FLIP ROBO

Housing Sales Price Prediction

Submitted by: **Haindavi Chakravarthi**

Internship: **23**

ACKNOWLEDGEMENT

To complete this project, I have gone through various case studies and project reports, how house price prediction is useful for the public & real-estate market. I have gone through some articles available in google.

First of all, I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity.

I wish to express our sincere thanks to the above people, without whom I would not have been able to complete this project.

INTRODUCTION

Business problem framing:

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them on at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia.

The company is looking at prospective properties to buy to enter the market. You are required to build a regression model using regularisation in order to predict the actual value of the prospective properties and decide whether to invest in them or not.

We are required to model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables.

They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns.

Conceptual background of the domain problem:

In the present project, we will build a predictive model to predict house price (price is a number from some defined range, so it will be regression task). For example, you want to sell a house and you don't know the price which you can take — it can't be too low or too high.

To find house price you usually try to find similar properties in your neighbourhood and based on gathered data you will try to assess your house price. We will do something similar, but with Machine Learning methods The objective of the project is to perform data visualization techniques to understand the insight of the data.

Machine learning often required to getting the understanding of the data and its insights. This project aims apply various python tools to get a visual understanding of the data and clean it to make it ready to apply machine learning and deep learning models on it.

- **Review of Literature:**

This study proposes a performance comparison between machine learning regression algorithms and Artificial Neural Network (ANN).

The regression algorithms used in this study are Multiple linear, Least Absolute Selection Operator (Lasso), Ridge, Random Forest. Moreover, this study attempts to analyse the correlation between variables to determine the most important factors that affect house prices There are two datasets used in this study which called public and local. They contain house prices.

The accuracy of the prediction is evaluated by checking the root square and root mean square error scores of the training model. The test is performed after applying the required pre-processing methods and splitting the data into two parts.

However, one part will be used in the training and the other in the test phase. I have also presented a binning strategy that improved the accuracy of the models.

This thesis attempts to show that Gradient Boosting Regressor when using the public dataset in training. The correlation graphs show the variables' level of dependency.

Motivation for the Problem:

Undertaken Housing prices are an important reflection of the economy, and housing price ranges are of great interest for both buyers and sellers.

In this project, house prices will be predicted given explanatory variables that cover many aspects of residential houses. The goal of this project is to create a regression model that are able to accurately estimate the price of the house given the features.

Analytical Problem Framing:

- **Mathematical/ Analytical Modelling of the Problem:**

A statistical model is a mathematical representation (or mathematical model) of observed data. When data analysts apply various statistical models to the data they are investigating, they are able to understand and interpret the information more strategically.

Rather than sifting through the raw data, this practice allows them to identify relationships between variables, make predictions about future sets of data, and visualize that data so that nonanalysts and stakeholders can consume and leverage it.

Multiple Linear Regression (MLR) is a supervised technique used to estimate the relationship between one dependent variable and more than one independent variables. Identifying the correlation and its cause-effect helps to make predictions by using these relations [4].

To estimate these relationships, the prediction accuracy of the model is essential; the complexity of the model is of more interest. However, Multiple Linear Regression is prone to many problems such as multicollinearity, noises, and overfitting, which effect on the prediction accuracy. Here I have used Linear Regression, RandomForestRegression & GradientBoostingRegressor.

Data Sources and their formats:

Our data comes from “House Prices: Prediction”. It contains 1168 training data points and 81 features that might help us predict the selling price of a house.

We will load the dataset into a Pandas data frame and continue with our model building we will pre-process the data by analyzing the variables and then we have a more general view on the top correlated features with the sale price.

The Dataset consists of 81 variables and their explanation is given below:

- **MSSubClass:** Identifies the type of dwelling involved in the sale.
- **MSZoning:** Identifies the general zoning classification of the sale.

- **LotFrontage:** Linear feet of street connected to property
- **LotArea:** Lot size in square feet
- **Street:** Type of road access to property
- **Alley:** Type of alley access to property
- **LotShape:** General shape of property
- **LandContour:** Flatness of the property
- **Utilities:** Type of utilities
- **LotConfig:** Lot configuration
- **LandSlope:** Slope of property
- **Neighborhood:** Physical locations within Ames city limits
- **Condition1:** Proximity to various conditions
- **Condition2:** Proximity to various conditions (if more than one is present)
- **BldgType:** Type of dwelling
- **HouseStyle:** Style of dwelling
- **OverallQual:** Rates the overall material and finish of the house
- **OverallCond:** Rates the overall condition of the house
- **YearBuilt:** Original construction date
- **YearRemodAdd:** Remodel date (same as construction date if no remodelling or additions)
- **RoofStyle:** Type of roof
- **RoofMatl:** Roof material
- **Exterior1st:** Exterior covering on house
- **Exterior2nd:** Exterior covering on house (if more than one material)
- **MasVnrType:** Masonry veneer type
- **MasVnrArea:** Masonry veneer area in square feet
- **ExterQual:** Evaluates the quality of the material on the exterior
- **ExterCond:** Evaluates the present condition of the material on the exterior
- **Foundation:** Type of foundation
- **BsmtQual:** Evaluates the height of the basement
- **BsmtCond:** Evaluates the general condition of the basement
- **BsmtExposure:** Refers to walkout or garden level walls
- **BsmtFinType1:** Rating of basement finished area

- **BsmtFinSF1:** Type 1 finished square feet
- **BsmtFinType2:** Rating of basement finished area (if multiple types)
- **BsmtFinSF2:** Type 2 finished square feet
- **BsmtUnfSF:** Unfinished square feet of basement area
- **TotalBsmtSF:** Total square feet of basement area
- **Heating:** Type of heating
- **HeatingQC:** Heating quality and condition
- **CentralAir:** Central air conditioning
- **Electrical:** Electrical system
- **1stFlrSF:** First Floor square feet
- **2ndFlrSF:** Second floor square feet
- **LowQualFinSF:** Low quality finished square feet (all floors)
- **GrLivArea:** Above grade (ground) living area square feet
- **BsmtFullBath:** Basement full bathrooms
- **BsmtHalfBath:** Basement half bathrooms
- **FullBath:** Full bathrooms above grade
- **HalfBath:** Half baths above grade
- **Bedroom:** Bedrooms above grade (does NOT include basement bedrooms)
- **Kitchen:** Kitchens above grade
- **KitchenQual:** Kitchen quality
- **TotRmsAbvGrd:** Total rooms above grade (does not include bathrooms)
- **Functional:** Home functionality (Assume typical unless deductions are warranted)
- **Fireplaces:** Number of fireplaces
- **FireplaceQu:** Fireplace quality
- **GarageType:** Garage location
- **GarageYrBlt:** Year garage was built
- **GarageFinish:** Interior finish of the garage
- **GarageCars:** Size of garage in car capacity
- **GarageArea:** Size of garage in square feet
- **GarageQual:** Garage quality
- **GarageCond:** Garage condition

• **PavedDrive:** Paved driveway

• **WoodDeckSF:** Wood deck area in square feet

• **OpenPorchSF:** Open porch area in square feet

• **EnclosedPorch:** Enclosed porch area in square feet

• **3SsnPorch:** Three season porch area in square feet

• **ScreenPorch:** Screen porch area in square feet

• **PoolArea:** Pool area in square feet

• **PoolQC:** Pool quality

• **Fence:** Fence quality

• **MiscFeature:** Miscellaneous feature not covered in other categories

• **MiscVal:** $Value of miscellaneous feature

• **MoSold:** Month Sold (MM)

• **YrSold:** Year Sold (YYYY)

• **SaleType:** Type of sale

• **SaleCondition:** Condition of sal

Data Preprocessing Done:

Data preprocessing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, greater is the reliance on the produced results. Incomplete, noisy, and inconsistent data are the properties of large real-world datasets. Data preprocessing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise and resolving inconsistencies.

• Incomplete data can occur for a number of reasons. Attributes of interest may not always be available, such as customer information for sales transaction data. Relevant data may not be recorded due to a misunderstanding, or because of equipment malfunctions.

• There are many possible reasons for noisy data (having incorrect attribute values):

The data collection instruments used may be faulty. There may have been human or computer errors occurring at data entry. Errors in data transmission can also occur. Incorrect data may also result from inconsistencies in naming conventions or data codes used, or inconsistent formats for input fields, such as date. There are a number of data preprocessing techniques available such as, 1. Data Cleaning 2. Data Integration 3. Data Transformation 4. Data Reduction Data cleaning can be applied to filling in missing values, remove noise, resolving inconsistencies, identifying and removing outliers in the data. Data integration merges data from multiple sources into a coherent data store, such as a data warehouse. Data transformations, such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurements. Data reduction can reduce the data size by eliminating redundant features, or clustering, for instance.

• Data Inputs- Logic- Output Relationships:

Data Science is the process of making some assumptions and hypothesis on the data, and testing them by performing some tasks. Initially we could make the following intuitive assumptions for each feature. We will start by creating a scatterplot matrix that will allow us to visualize the pair-wise relationships and correlations between the different features. It is also quite useful to have a quick overview of how the data is distributed and wheter it contains or not outliers. We are going to create now a correlation matrix to quantify and summarize the relationships between the variables.

• State the set of assumptions (if any) related to the problem under consideration:

This study will not cover all regression algorithms; instead, it is focused on the chosen algorithm, starting from the basic regression techniques to the advanced ones. Likewise, the artificial neural network that has many techniques and a wide area and several training methods that do not fit in this study.

• Identification of possible problem-solving approaches (methods):

This study has been organised through theoretical research and practical implementation of regression algorithms. The theoretical part relies on peer-reviewed articles to answer the research questions, which is going to be detailed. The practical part will be performed according to the design described below and detailed furthermore. The experiment is done to pre-process the data and evaluate the prediction accuracy of the models. The experiment has multiple stages that are required to get the prediction results. These stages can be defined as: - Pre-processing: both datasets will be checked and pre-processed using different methods. These methods have various ways of handling data. Thus, the pre-processing is done on multiple iterations where each time the accuracy will be evaluated with the used combination. - Data splitting: dividing the dataset into two parts is essential to train the model with one and use the other in the evaluation. The dataset will be split 75% for training and 25% for testing. - Evaluation: the accuracy of both datasets will be evaluated. Performance: alongside the evaluation metrics, the required time to train the model will be measured to show the algorithm vary in terms of time. - Correlation: correlation between the available features and house price will be evaluated using the Pearson Coefficient Correlation to identify whether the features have a negative, positive or zero correlation with the house price.

• Testing of Identified Approaches (Algorithms):

The algorithms used in this study have different properties that will be used during the implementation. The experiment is done with jupyter notebook Python as a programming language. However, in all algorithms, the data is split into four variables, namely, X_train, X_test, y_train, and y_test, by using train_test_split class from the library sklearn. model_selection. In addition, in all algorithms, the train_test_split class takes as parameters the independent variables, which is the data, the dependent variable, which is the SalePrice. The properties and design of each algorithm are as below: Regression Model Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Random Forest Regression Model A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Gradient boosting Gradient boosting is a

machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.

Importing the necessary libraries :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Data Collection :

```
train = pd.read_csv(r"C:/Users/Akanksha/Downloads/Project-Housing_splitted/train.csv")
```

```
pd.set_option("display.max_columns",None)
```

```
train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condi
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	

Null values of the train data :

```
train.isnull().sum()
```

```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage     214
LotArea           0
Street            0
Alley          1091
LotShape          0
LandContour       0
Utilities         0
LotConfig         0
LandSlope         0
Neighborhood      0
Condition1        0
Condition2        0
BldgType          0
HouseStyle        0
OverallQual       0
OverallCond       0
YearBuilt         0
YearRemodAdd      0
RoofStyle         0
RoofMatl          0
Exterior1st       0
Exterior2nd       0
MasVnrType        7
MasVnrArea        7
ExterQual         0
ExterCond         0
Foundation        0
BsmtQual         30
BsmtCond         30
BsmtExposure     31
BsmtFinType1     30
BsmtFinSF1        0
BsmtFinType2     31
BsmtFinSF2        0
BsmtUnfSF         0
TotalBsmtSF       0
Heating           0
HeatingQC         0
```

Statistical description of the train data :

```
pd.set_option("display.max_columns",None)
train.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1168.000000	1168.000000	954.00000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1161.000000	1168.000000	1168.000000
mean	724.136130	56.767979	70.98847	10484.749144	6.104452	5.595890	1970.930651	1984.758562	102.310078	444.726027	46.647260
std	416.159877	41.940650	24.82875	8957.442311	1.390153	1.124343	30.145255	20.785185	182.595606	462.664785	163.520016
min	1.000000	20.000000	21.00000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000
25%	360.500000	20.000000	60.00000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000
50%	714.500000	50.000000	70.00000	9522.500000	6.000000	5.000000	1972.000000	1993.000000	0.000000	385.500000	0.000000
75%	1079.500000	70.000000	80.00000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000
max	1460.000000	190.000000	313.00000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

Observation : 1) Here we can see that there are few columns with min value and 25% quartile values equal to "0"

2) Also we can see that there are few columns where the value of "Standard deviation" is higher than the "mean" value

3) Also there are few columns where min value,25% quartile value and 50% quartile value and 75% quartile values are "0"

Removing the unnecessary columns:

```
train = train.drop(columns = 'Id')
```

```
train.shape
```

```
(1168, 80)
```

Observation : Here we have successfully deleted the column

Removing the columns which have mostly null values :

```
train = train.drop(columns = ['Alley','MiscFeature','PoolQC'])
```

```
train.shape
```

```
(1168, 77)
```

Observation : Here we can see that we have successfully deleted the columns.

Filling the null values in the columns :

We will use Knn imputer to fill the null values in the following columns :

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, KNNImputer
```

```
knnimp = KNNImputer()
```

```
train[['LotFrontage']] = knnimp.fit_transform(train[['LotFrontage']])
```

Using ".fillna"method to fill null values :

```
train['MasVnrType'] = train['MasVnrType'].fillna(train['MasVnrType'].mode()[0])
```

```
train['MasVnrArea'] = train['MasVnrArea'].fillna(0)
```

```
train['BsmtQual'] = train['BsmtQual'].fillna('NA')
train['BsmtCond'] = train['BsmtCond'].fillna('NA')
train['BsmtExposure'] = train['BsmtExposure'].fillna('NA')
train['BsmtFinType1'] = train['BsmtFinType1'].fillna('NA')
train['BsmtFinType2'] = train['BsmtFinType2'].fillna('NA')
```

```
train['FireplaceQu'] = train['FireplaceQu'].fillna('NA')
```

```
train['GarageType'] = train['GarageType'].fillna('NA')
train['GarageYrBlt'] = train['GarageYrBlt'].fillna(0)
train['GarageFinish'] = train['GarageFinish'].fillna('NA')
train['GarageQual'] = train['GarageQual'].fillna('NA')
train['GarageCond'] = train['GarageCond'].fillna('NA')
```

```
train['Fence'] = train['Fence'].fillna('NA')
```

Checking the null-values again:

```
train.isnull().sum()
```

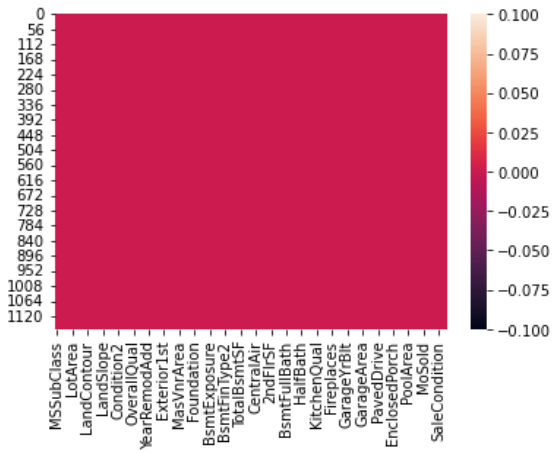
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
LotShape         0
LandContour      0
Utilities        0
LotConfig        0
LandSlope        0
Neighborhood     0
Condition1       0
Condition2       0
BldgType         0
HouseStyle       0
OverallQual      0
OverallCond      0
YearBuilt        0
YearRemodAdd     0
RoofStyle        0
RoofMatl         0
Exterior1st      0
Exterior2nd      0
MasVnrType       0
MasVnrArea       0
ExterQual        0
ExterCond        0
Foundation       0
BsmtQual         0
BsmtCond         0
BsmtExposure     0
BsmtFinType1     0
BsmtFinSF1       0
BsmtFinType2     0
BsmtFinSF2       0
BsmtUnfSF        0
TotalBsmtSF      0
Heating          0
HeatingQC        0
CentralAir       0
Electrical       0
```

Observation : Here we have seen that we have successfully treated all the null values present in the columns of the respective data.

Plotting the heatmap for the null values :

```
sns.heatmap(train.isnull())
```

```
<AxesSubplot:>
```



Observation : Here we can see that there are no null values in the columns according to the heatmap plotted

Key Metrics for success in solving problem under consideration:

Will start by creating a scatterplot matrix that will allow us to visualize the pair-wise relationships and correlations between the different features and the correlation map is also plotted.

Before checking the correlation, we have encoded the data using ordinal encoder to convert categorical data into numerical data.

Using the ordinal encoder to encode the categorical data :

```
from sklearn.preprocessing import OrdinalEncoder
```

```
encoder = OrdinalEncoder()
for i in train.columns:
    if train[i].dtypes == 'object':
        train[i] = encoder.fit_transform(train[i].values.reshape(-1,1))
```

```
data_corr = train.corr()
data_corr['SalePrice'].sort_values(ascending = False)
```

```
SalePrice        1.000000
OverallQual      0.789185
GrLivArea        0.707300
GarageCars       0.628329
GarageArea       0.619000
TotalBsmtSF      0.595042
1stFlrSF         0.587642
FullBath         0.554988
TotRmsAbvGrd     0.528363
YearBuilt        0.514408
YearRemodAdd     0.507831
MasVnrArea       0.460535
Fireplaces       0.459611
Foundation       0.374169
BsmtFinSF1       0.362874
OpenPorchSF      0.339500
2ndFlrSF         0.330386
LotFrontage      0.323779
WoodDeckSF       0.315444
HalfBath         0.295592
GarageYrBlt      0.265622
LotArea          0.249499
GarageCond       0.249340
CentralAir       0.246754
Electrical       0.234621
PavedDrive       0.231707
SaleCondition    0.217687
BsmtUnfSF        0.215724
BsmtFullBath     0.212924
HouseStyle       0.205502
Neighborhood     0.198942
RoofStyle        0.192654
GarageQual       0.192392
RoofMatl         0.159865
BedroomAbvGr     0.158281
Fence            0.143922
Functional       0.118673
ExterCond        0.115167
Exterior1st      0.108451
Condition1       0.105820
PoolArea         0.103280
ScreenPorch      0.100284
Exterior2nd      0.097541
BsmtCond         0.084121
MoSold           0.072764
BsmtFinType2     0.069657
3SsnPorch        0.060119
Street           0.044753
Condition2       0.033956
LandContour      0.032836
LandSlope        0.015485
MasVnrType       0.007732
BsmtFinSF2      -0.010151
BsmtHalfBath    -0.011109
MiscVal         -0.013071
LowQualFinSF    -0.032381
YrSold          -0.045508
SaleType        -0.050851
LotConfig       -0.060452
MSSubClass      -0.060775
OverallCond     -0.065642
BldgType        -0.066028
FireplaceQu     -0.076951
BsmtFinType1    -0.099860
Heating         -0.100021
EnclosedPorch   -0.115004
KitchenAbvGr    -0.132108
```

```
MSZoning         -0.133221
LotShape         -0.248171
BsmtExposure     -0.267635
HeatingQC        -0.406604
GarageType       -0.415370
GarageFinish     -0.424922
KitchenQual      -0.592468
BsmtQual         -0.601307
ExterQual        -0.624820
Utilities              NaN
Name: SalePrice, dtype: float64
```

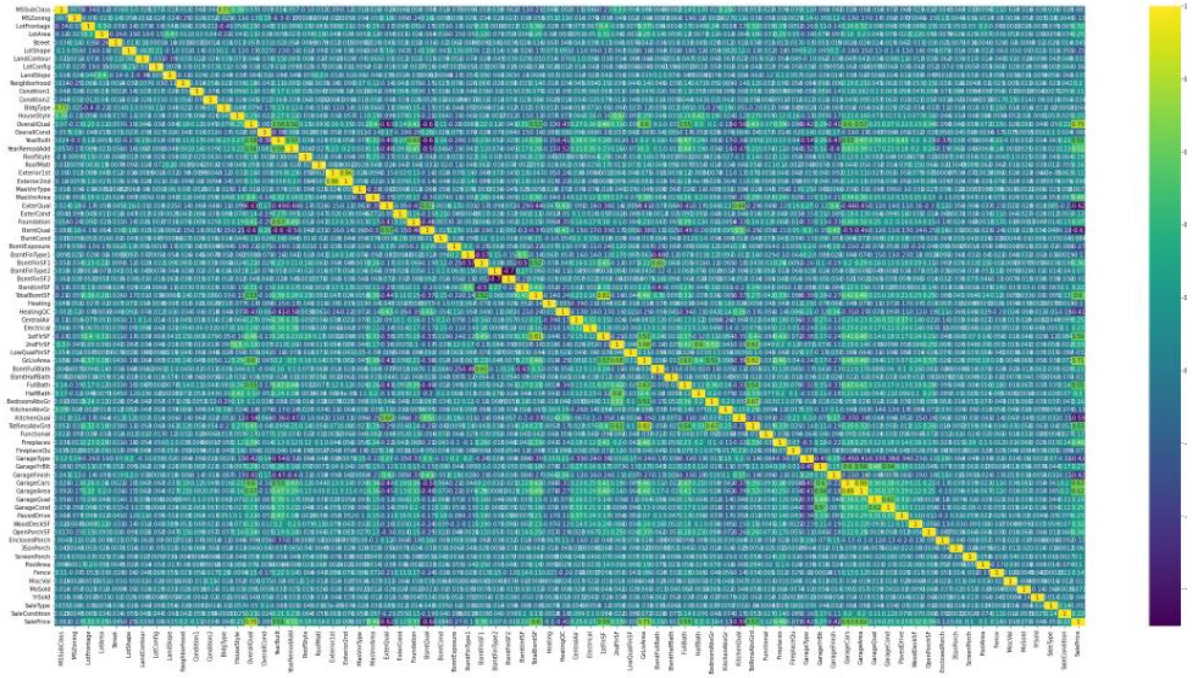
```
Highly Correlated variables with the SalePrice
 OverallQual      0.789185
GrLivArea        0.707300
GarageCars       0.628329
GarageArea       0.619000
TotalBsmtSF      0.595042
1stFlrSF         0.587642
FullBath         0.554988
TotRmsAbvGrd     0.528363
YearBuilt        0.514408
YearRemodAdd     0.507831
MasVnrArea       0.460535
Fireplaces       0.459611
Foundation       0.374169
BsmtFinSF1       0.362874
OpenPorchSF      0.339500
2ndFlrSF         0.330386
LotFrontage      0.319416
WoodDeckSF       0.315444
HalfBath         0.295592
GarageYrBlt      0.265622
LotArea          0.249499
GarageCond       0.249340
CentralAir       0.246754
LotShape        -0.248171
BsmtExposure    -0.267635
HeatingQC       -0.406604
GarageType      -0.415370
GarageFinish    -0.424922
KitchenQual     -0.592468
BsmtQual        -0.601307
ExterQual       -0.624820
```

Correlation table :

Here we will be going to check the multicollinearity issue between any of the variables.

```
Correlation = train.corr()
plt.figure(figsize = (40,20))
sns.heatmap(Correlation, annot = True, cmap = 'viridis')
```

```
<AxesSubplot:>
```



Here OverallQual is highly correlated with target feature of sale price by 82%.

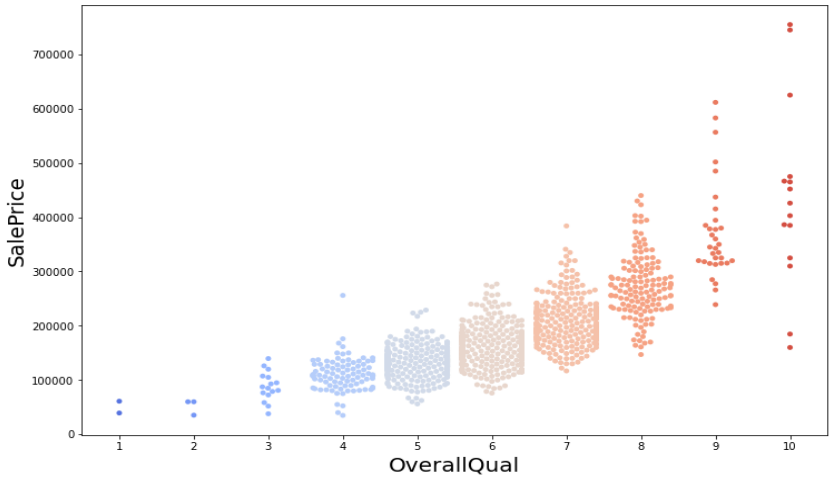
Visualization :

Now we will visualize the highly correlated values with the target variable :

OverallQual :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'OverallQual',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('OverallQual', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

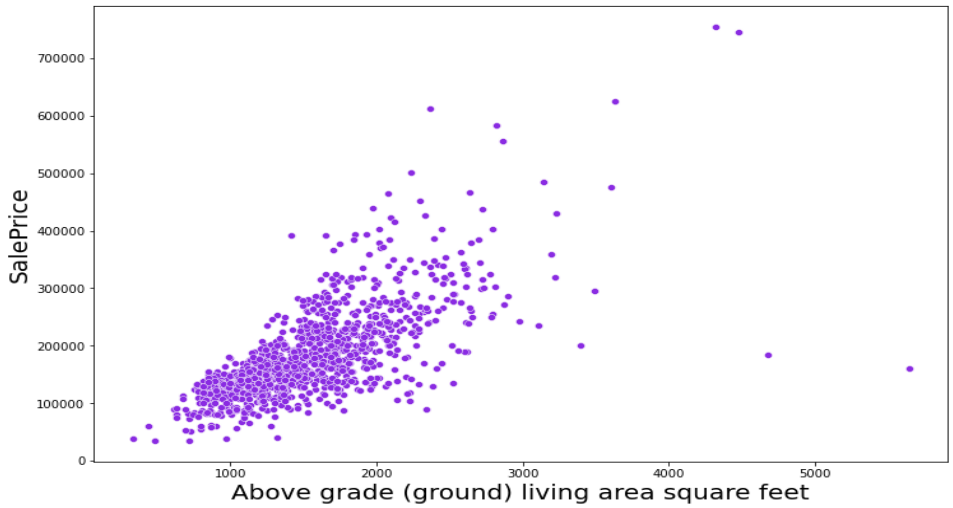


Observation : Here we have less density in the attribute 1 of the column "overallQual" at the sales price 1,00,000 and high density is from the attribute 5 to 8 .

GrLivArea :

```
plt.figure(figsize = (12,8))
sns.scatterplot(x = 'GrLivArea',y = 'SalePrice', data = train, color = 'blueviolet')
plt.xlabel('Above grade (ground) living area square feet', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

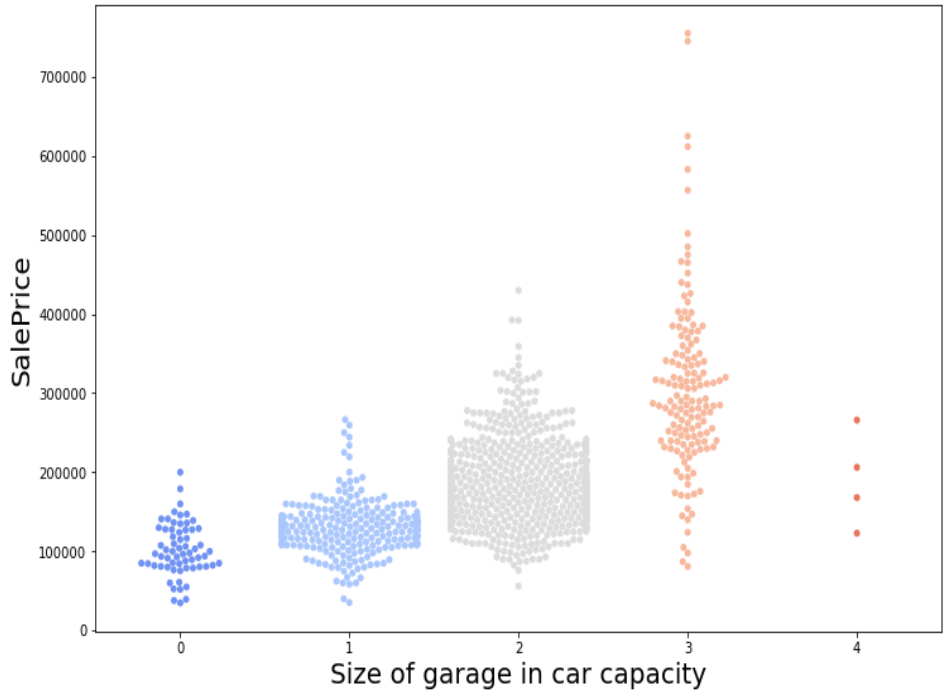


Observation : Here we can see that the density in the scatterplot looks linear and the high density is from 1000 to 2000 at the sales price from 1,00,00 to 3,00,00 and there is no density in the attribute 5000 and alos at the sales price 7,00,000

GarageCars :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'GarageCars',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('Size of garage in car capacity', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

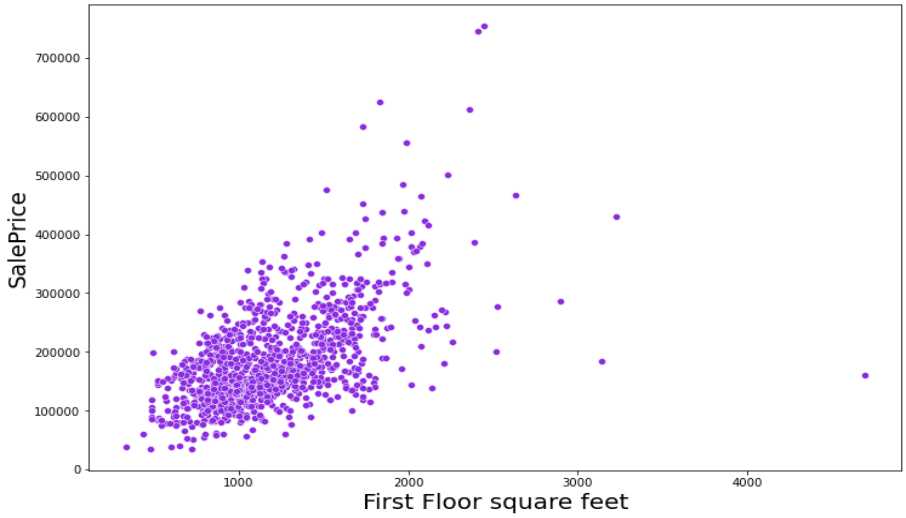


observation : Here we can see that the high density is present in the attribute 2 and the high dense category is present at the sales price between 1,00,000 to 3,00,000.

1stFlrSF :

```
plt.figure(figsize = (12,8))
sns.scatterplot(x = '1stFlrSF',y = 'SalePrice', data = train, color = 'blueviolet')
plt.xlabel('First Floor square feet', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

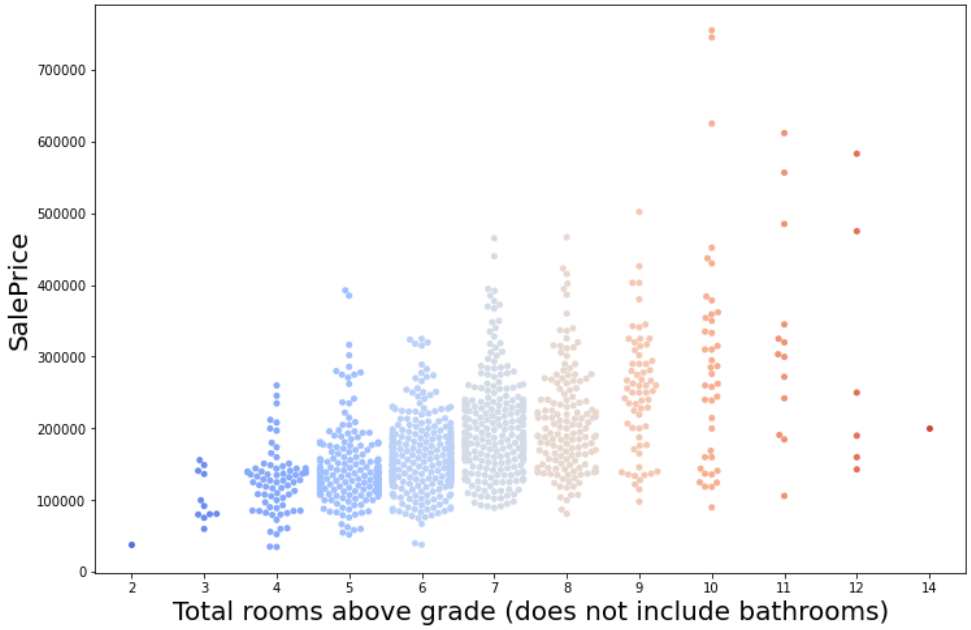


Observation : Here we can see that the high density is present below 2000 in the variable column and in the sales price it is between 1,00,000 to 3,00,000

TotRmsAbvGrd :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'TotRmsAbvGrd',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('Total rooms above grade (does not include bathrooms)', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

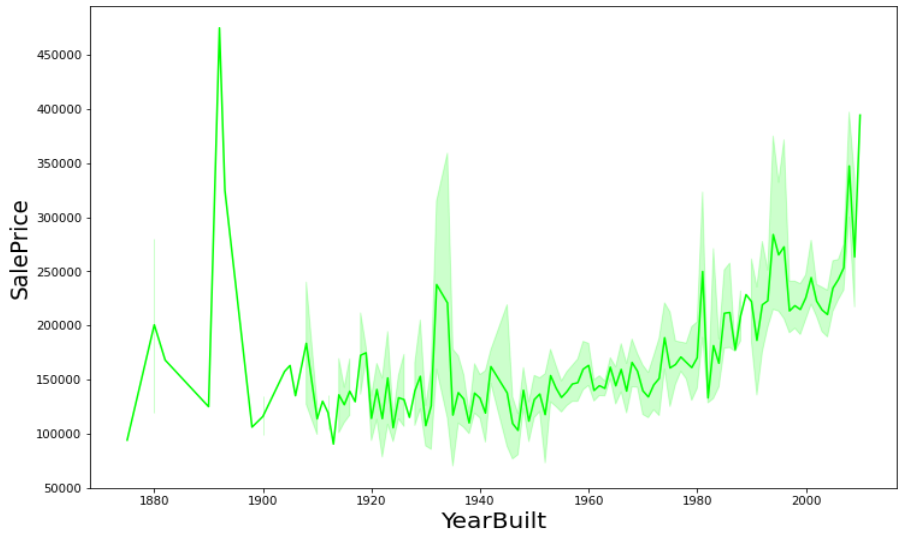


Observation : Here we can see that the high density is present between the attributes 4 to 8 in the variable column and in the sales price it is between 1,00,000 to 3,00,000

YearBuilt :

```
plt.figure(figsize = (12,8))
sns.lineplot(x = 'YearBuilt',y = 'SalePrice', data = train, color = 'lime')
plt.xlabel('YearBuilt', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

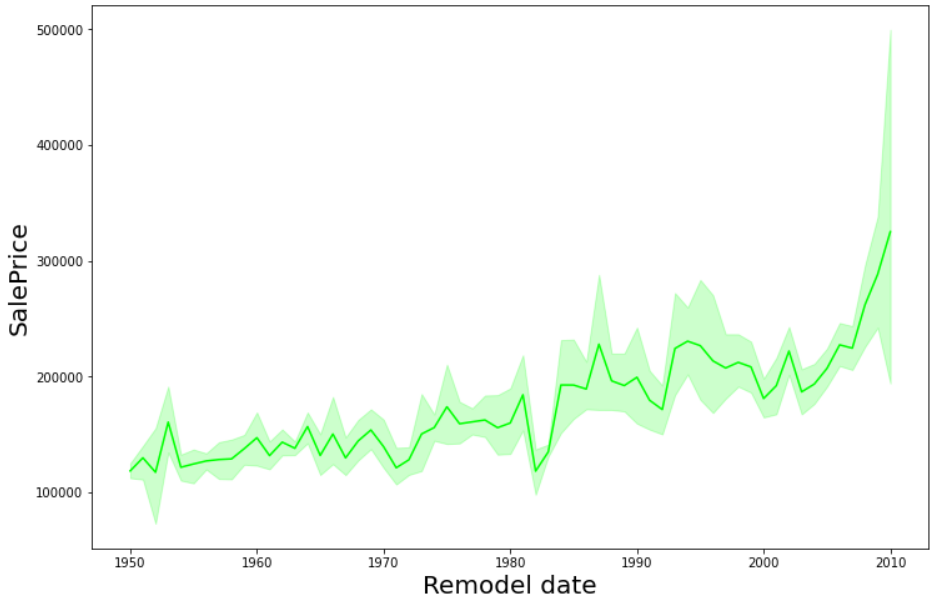


Observation : Here we can see that the high sales price is in the years between 1880 to 1900 reached the highest sales price and the density is highest in the variable column between 1920 to 2000 between the sales prices 1,00,000 to 3,00,000

YearRemodAdd :

```
plt.figure(figsize = (12,8))
sns.lineplot(x = 'YearRemodAdd',y = 'SalePrice', data = train, color = 'lime')
plt.xlabel('Remodel date', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

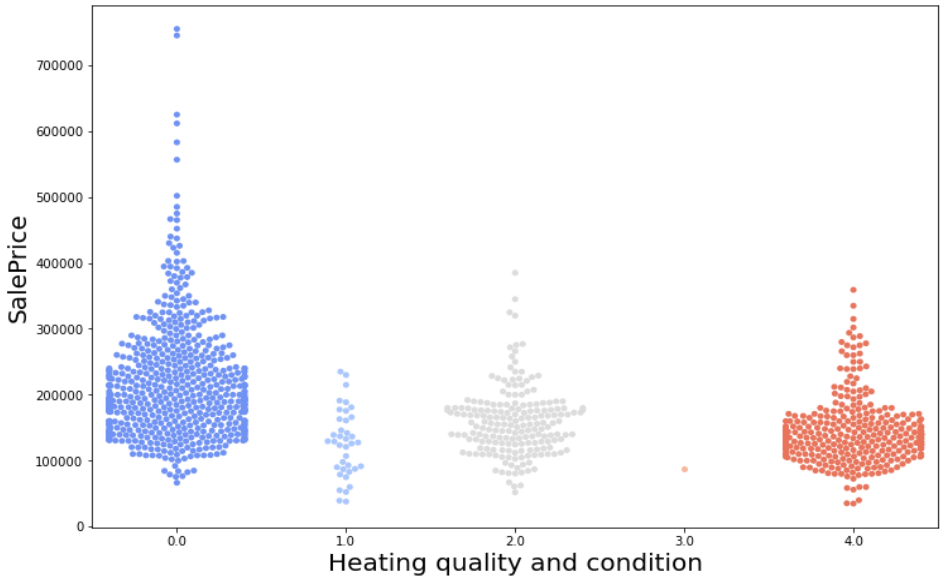


Observation : Here we can see that the density of the distribution is varying between the attributes in the variable column randomly and ranging between 1,00,000 to 3,00,000 in sales price

HeatingQC :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'HeatingQC',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('Heating quality and condition', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

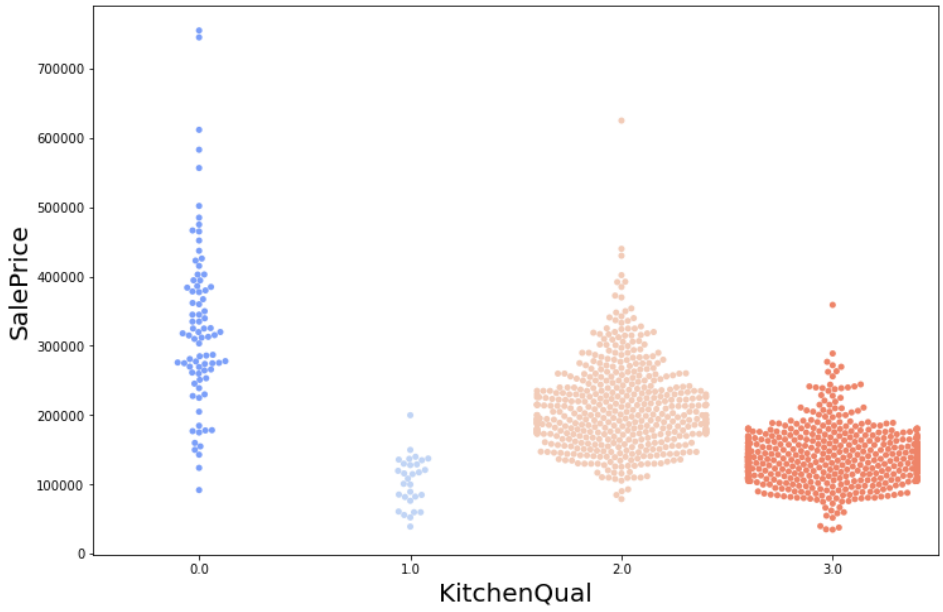


Observation : Here we can see that the density of the variable is high for the attribute 0.0 followed by 4.0 ranging from 1,00,000 to 3,00,000

KitchenQual :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'KitchenQual',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('KitchenQual', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

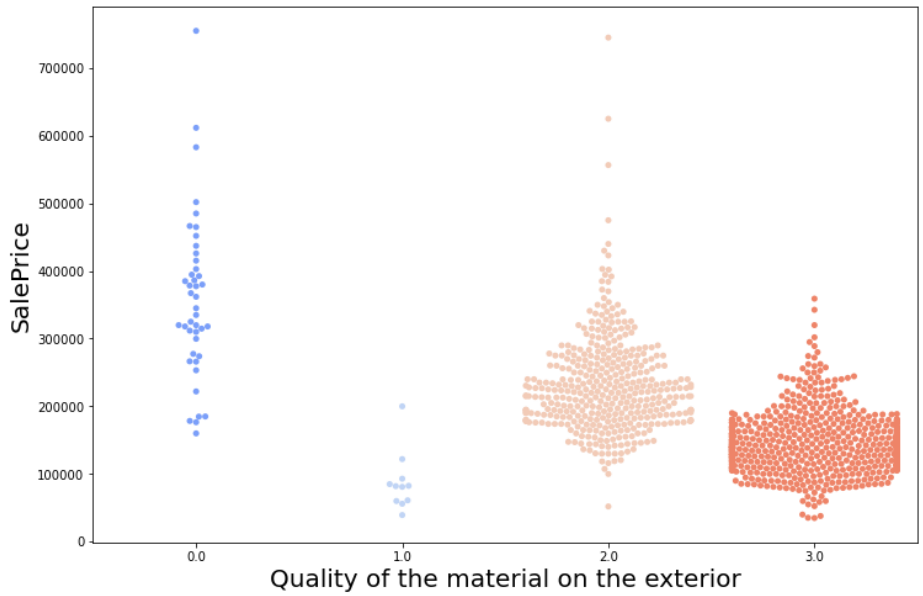


Observation : Here we can see that the attributes 2.0 and 3.0 have high density distribution in variable column ranging from 1,00,000 to 3,80,000 in label column sales price

ExterQual :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'ExterQual',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('Quality of the material on the exterior', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```

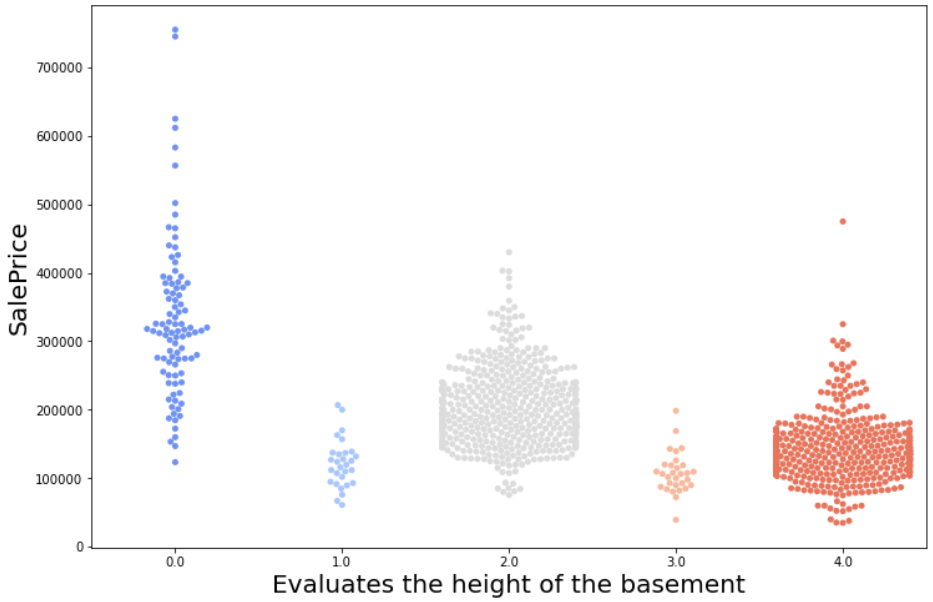


Observation : Here we can see that the attributes 2.0 and 3.0 have high density distribution at the range between 1,00,000 to 3,50,000

BsmtQual :

```
plt.figure(figsize = (12,8))
sns.swarmplot(x = 'BsmtQual',y = 'SalePrice', data = train, palette = 'coolwarm')
plt.xlabel('Evaluates the height of the basement', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

```
Text(0, 0.5, 'SalePrice')
```



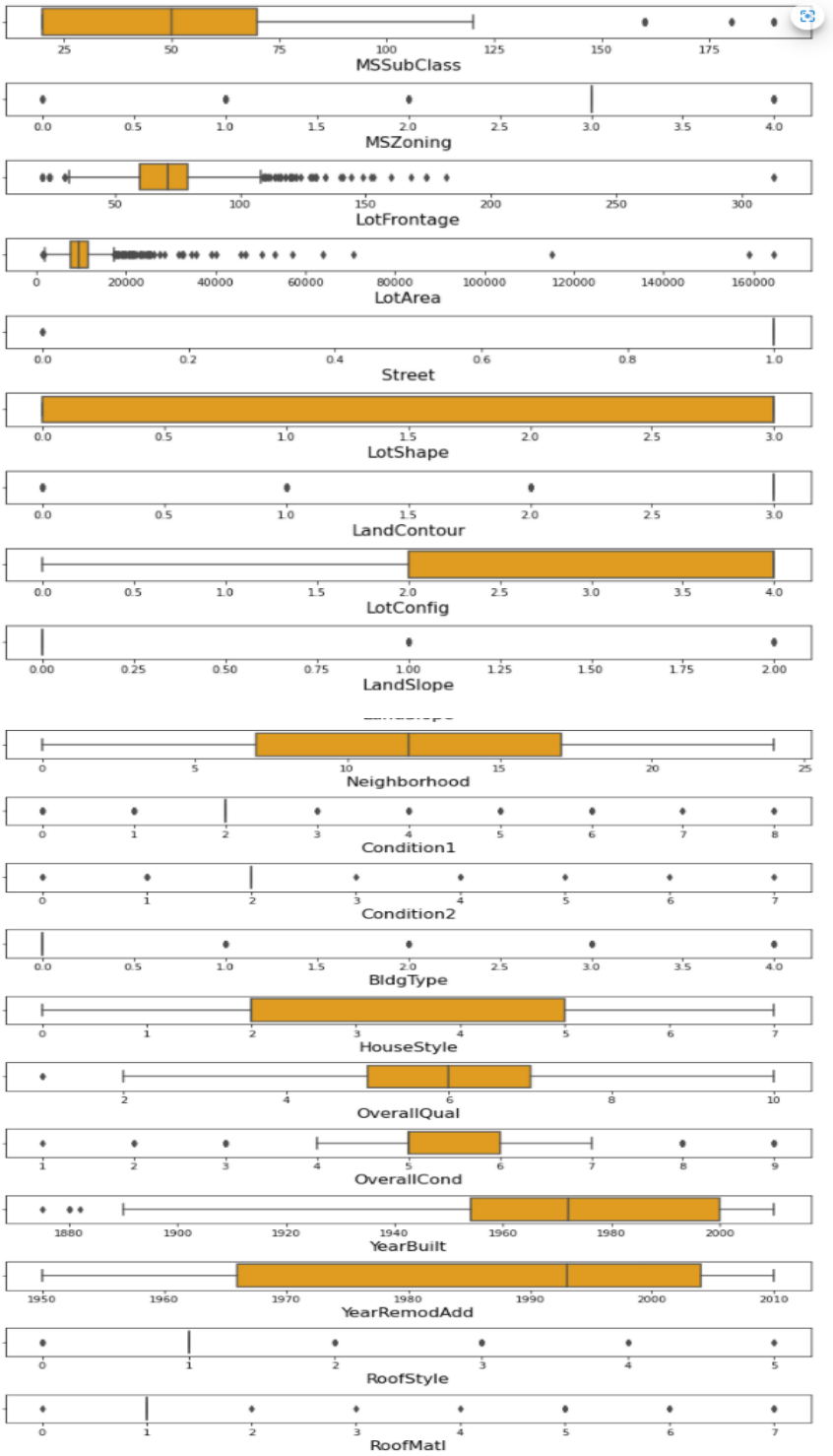
Observation : Here we can see that the high density is for the attribute 2.0 followed by 4.0 in the variable column which ranges from 1,00,000 to 3,00,000 in the label column

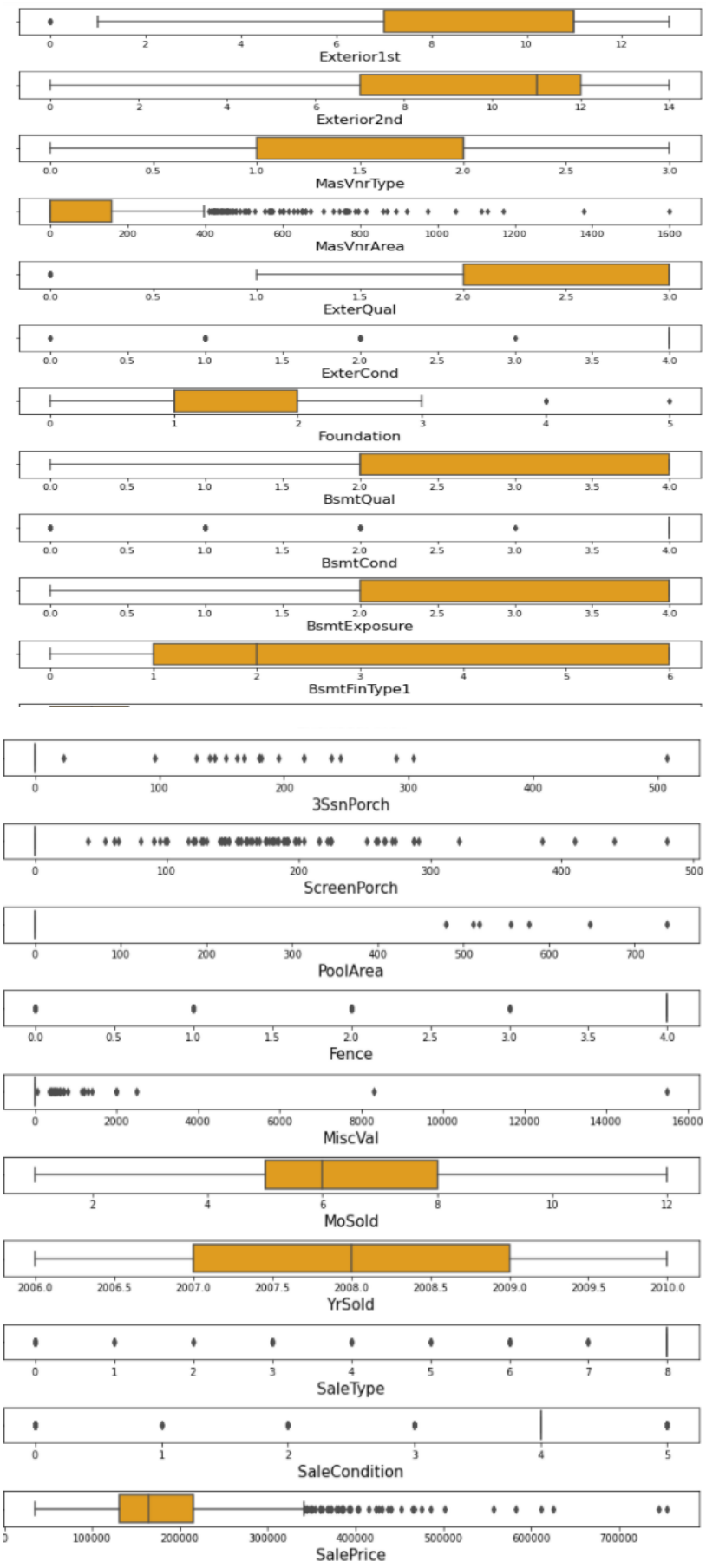
Here we are going to delete the column "utilities" as the variable has no correlation at all with the label column :

```
: train = train.drop(columns = 'Utilities')
```

Checking the outliers :

```
plt.figure(figsize = (10,90))
pltnum = 1
for i in train:
    if pltnum <= 80:
        plt.subplot(80,1, pltnum)
        sns.boxplot(train[i], color = 'orange')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```




Exterior1st
Exterior2nd
MasVnrType
MasVnrArea
ExterQual
ExterCond
Foundation
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
3SsnPorch
ScreenPorch
PoolArea
Fence
MiscVal
MoSold
YrSold
SaleType
SaleCondition
SalePrice

Observation : Here we can see that most of the columns have outliers of which few of them are with many number of outliers and few with few outliers which are far away from eachother and so treating these outliers is required because if we donot treat these outliers this may effect our model accuracy.

Treating the outliers using Z-Score method :

```
from scipy.stats import zscore
```

```
z = np.abs(zscore(train[['LotFrontage','LotArea','MasVnrArea','GarageArea','OpenPorchSF','BsmtFinSF2','BsmtUnfSF','EnclosedPorch'
z.head()
```

	LotFrontage	LotArea	MasVnrArea	GarageArea	OpenPorchSF	BsmtFinSF2	BsmtUnfSF	EnclosedPorch	ScreenPorch
0	0.000000	0.620616	0.558343	0.171944	2.387850	0.285392	0.864410	0.364375	0.273377
1	1.070631	0.600903	0.558343	0.672371	2.417992	4.749787	1.053642	0.364375	3.795117
2	0.936867	0.063075	0.558343	0.101973	1.257525	0.285392	0.700654	0.364375	0.273377
3	1.516514	0.141424	2.076985	0.322517	1.136957	0.285392	1.267363	0.364375	0.273377
4	0.000000	0.686902	0.133430	0.243217	0.701705	0.285392	0.475801	0.364375	0.273377

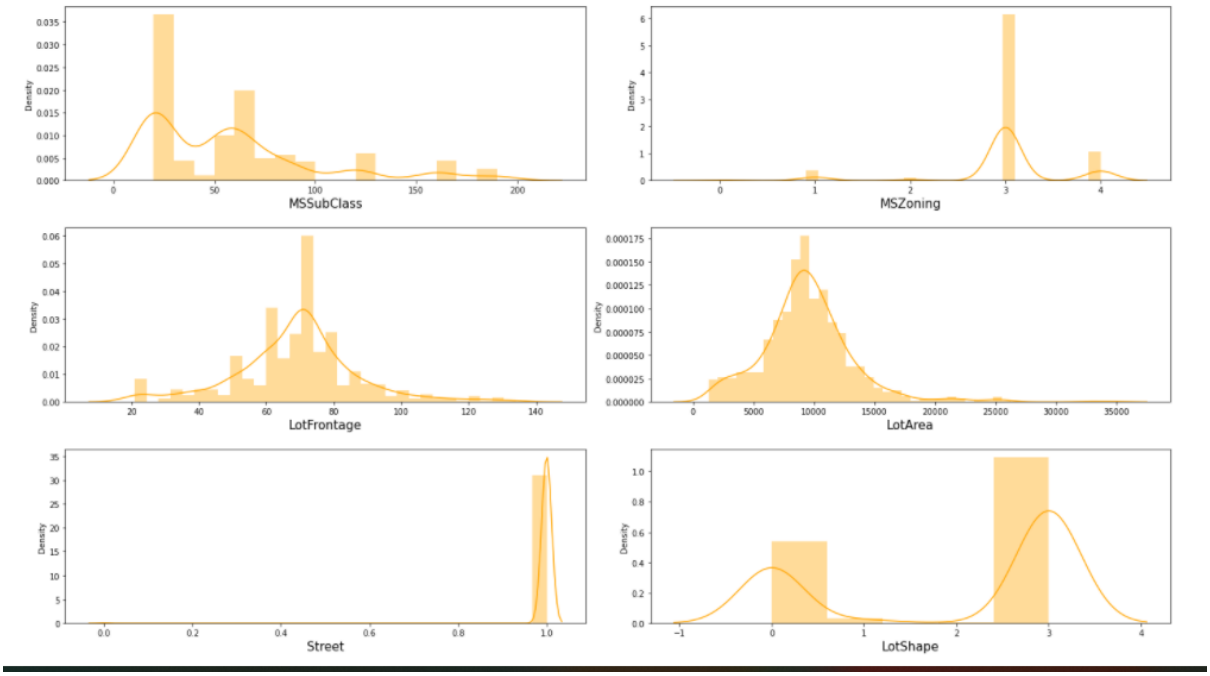
```
train_new = train[(z<3).all(axis = 1)]
print(train.shape)
print(train_new.shape)
```

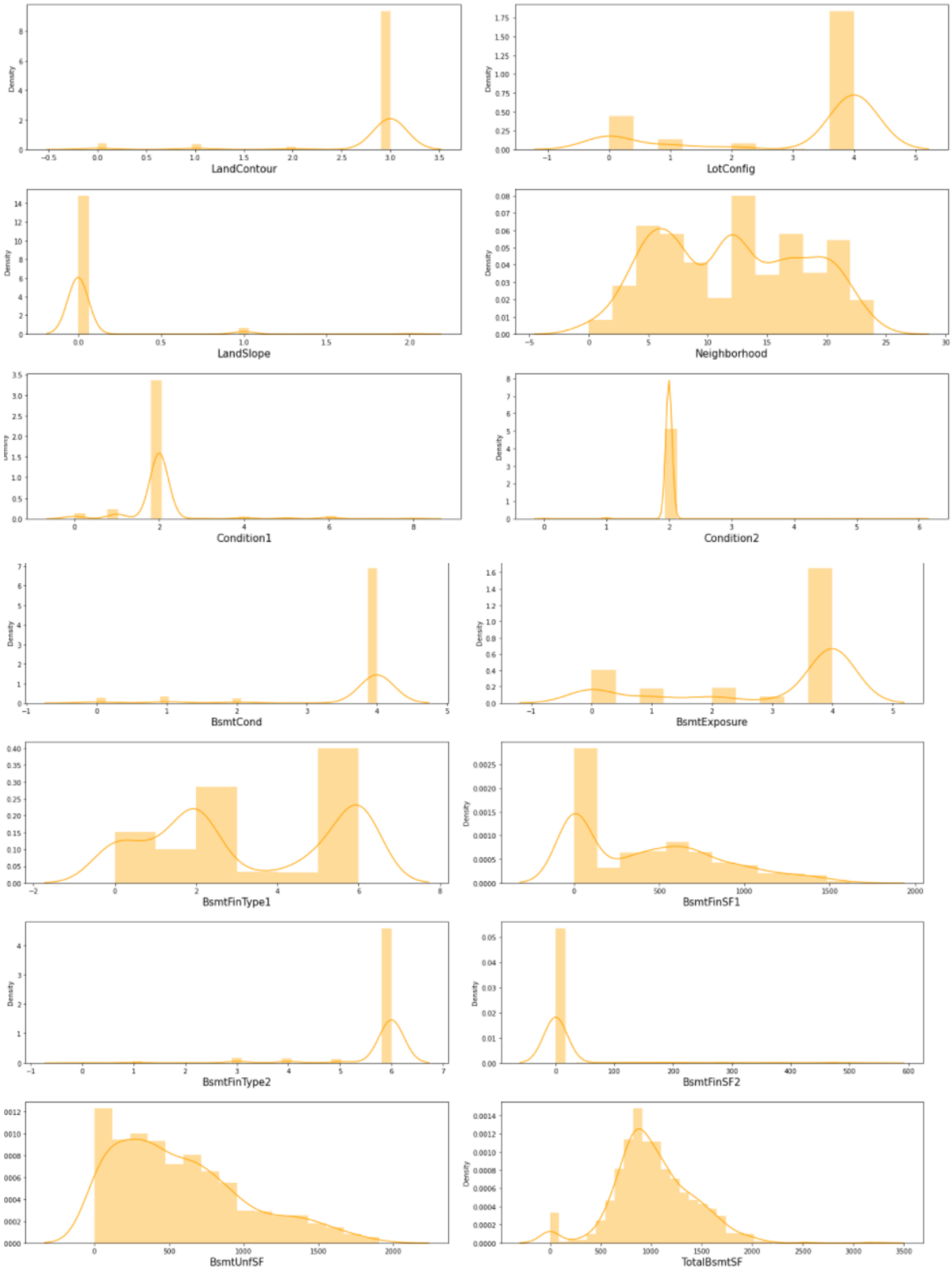
```
(1168, 76)
(988, 76)
```

Observation : Here we can see that the number of records change ie., reduced after using z-score method.

Checking the skewness of the data through visualization:

```
plt.figure(figsize = (20,150))
pltnum = 1
for i in train_new:
    if pltnum <= 80:
        plt.subplot(40,2, pltnum)
        sns.distplot(train_new[i], color = 'orange')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```





Observation : here we can see that there are a number of columns with a lot of skewness, few are left skewed and few are right skewed and few of them have uniform distribution and few are not at all in distribution , we have to treat the skewness in all these columns and if we dont then it may effect our model accuracy.

Splitting the data :

```
x = train_new.drop(columns = 'SalePrice')
y = train_new['SalePrice']
```

Scaling the data using standard scaler :

```
from sklearn.preprocessing import StandardScaler
```

```
scal = StandardScaler()
sc = scal.fit_transform(x)
x = pd.DataFrame(sc, columns = x.columns)
```

Treating the skewness using power transform method :

```
from sklearn.preprocessing import LabelEncoder,power_transform
```

```
train_tr = power_transform(x, method = 'yeo-johnson')
x = pd.DataFrame(train_tr, columns = x.columns)
```

Deleting the unnecessary columns :

```
x = x.drop(columns = ['MiscVal','PoolArea','ScreenPorch','3SsnPorch','EnclosedPorch','BsmtFinSF2','Exterior2nd'])
```

Checking the best random state to controlthe overfitting of the model :

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
rs = 0
for i in range(0,300):
    x_train,x_val, y_train,y_val = train_test_split(x,y,test_size = 0.3, random_state = i)
    lg = LinearRegression()
    lg.fit(x_train,y_train)
    val_pred = lg.predict(x_val)
    tr_score = lg.score(x_train,y_train)
    val_score = lg.score(x_val,y_val)
    if round(tr_score*100,1)==round(val_score*100,1):
        if i>rs:
            rs = i
print('the best random state for the data set is', rs)
```

```
the best random state for the data set is 54
```

Splitting the dataset with best random state :

```
x_train,x_val,y_train,y_val = train_test_split(x,y, test_size = 0.3, random_state = rs)
```

I'm testing the results with the below algorithms.

1. Linear Regression
2. Random Forest Regressor
3. Extra Trees Regressor
4. Extra Trees Regressor
5. K-Nearest Neighbours Regressor

In order to test the model, I'm using Mean Absolute Error and R2 score, further in order to verify the model's fit, I'm using cross val score to identify the best model.

Linear Regression :

```
linear = LinearRegression()
linear.fit(x_train,y_train)
linear_pred = linear.predict(x_val)
linear_score = linear.score(x_val,y_val)
linear_score
```

```
0.8858295492506131
```

```
linear_rmse = mean_absolute_error(y_val, linear_pred)
print('The recoreded mean absolute error for the Linear Regression is: ', linear_rmse)
```

```
The recoreded mean absolute error for the Linear Regression is:  16360.272609304564
```

Random Forest Regressor :

```
from sklearn.ensemble import RandomForestRegressor
```

```
rfr  = RandomForestRegressor()
rfr.fit(x_train,y_train)
rfr_pred = rfr.predict(x_val)
rfr_score = rfr.score(x_val,y_val)
rfr_score
```

```
0.8960849533389239
```

```
rfr_rmse = mean_absolute_error(y_val, rfr_pred)
print('The recoreded mean absolute error for the Random Forest Regression is: ', rfr_rmse)
```

```
The recoreded mean absolute error for the Random Forest Regression is:  16262.13107744108
```

Extra trees Regressor :

```
from sklearn.ensemble import ExtraTreesRegressor
```

```
et  = ExtraTreesRegressor()
et.fit(x_train,y_train)
et_pred = et.predict(x_val)
et_score = et.score(x_val,y_val)
et_score
```

```
0.8900968460647097
```

```
et_rmse = mean_absolute_error(y_val, et_pred)
print('The recoreded  mean absolute error for the ExtraTrees Regression is: ', et_rmse)
```

```
The recoreded  mean absolute error for the ExtraTrees Regression is:  15006.738653198656
```

KNN Regressor :

```
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn import metrics
```

```
knn=KNN()
knn.fit(x_train,y_train)

#prediction
predknn=knn.predict(x_val)
print('R2_Score:',metrics.r2_score(y_val,predknn))
```

```
R2_Score: 0.8367038162034272
```

```
# Mean Absolute Error (MAE)
print(metrics.mean_absolute_error(y_val, predknn))

# Mean Squared Error (MSE)
print(metrics.mean_squared_error(y_val, predknn))

# Root Mean Squared Error (RMSE)
print(np.sqrt(metrics.mean_squared_error(y_val, predknn)))
```

```
18694.833670033673
684479478.3046465
26162.558710964156
```

Ridge Regression :

```
from sklearn.linear_model import Ridge, RidgeCV
```

```
ridgecv = RidgeCV(alphas = np.arange(0.001,0.1,0.01), normalize = True)
ridgecv.fit(x_train, y_train)
```

```
RidgeCV(alphas=array([0.001, 0.011, 0.021, 0.031, 0.041, 0.051, 0.061, 0.071, 0.081,
       0.091]),
        normalize=True)
```

```
alpha = ridgecv.alpha_
alpha
```

```
0.09099999999999998
```

```
ridge_reg = Ridge(alpha)
ridge_reg.fit(x_train, y_train)
```

```
Ridge(alpha=0.09099999999999998)
```

```
ridge_reg.score(x_val, y_val)
```

```
0.8858487825612413
```

```
rid_pred = ridge_reg.predict(x_val)
```

```
rid_mae = mean_absolute_error(y_val, rid_pred)
print('The recoreded mean absolute error for the Ridge Regression is: ', rid_mae)
```

```
The recoreded mean absolute error for the Ridge Regression is:  16357.73727630437
```

Crossvalidation Scores :

```
cv = cross_val_score(linear,x,y,scoring ='r2', cv = 5)
cv =cv.mean()
cv
```

```
0.8578455991198158
```

```
cv1 = cross_val_score(rfr,x,y,scoring ='r2', cv = 5)
cv1 =cv1.mean()
cv1
```

```
0.8695330035399623
```

```
cv2 = cross_val_score(et,x,y,scoring ='r2', cv = 5)
cv2 =cv2.mean()
cv2
```

```
0.8612282554602441
```

```
cv3 = cross_val_score(ridge_reg,x,y,scoring ='r2', cv = 5)
cv3 =cv3.mean()
cv3
```

```
0.8578754062597206
```

Hyper parameter tuning :

Here we can see that the Random Forest model has the highest R2 Score :

```
from sklearn.model_selection import GridSearchCV
```

```
params ={'n_estimators':[100,200,300,400],
        'max_depth':[13,15,17,19],
        'min_samples_split':[3,4,5,6],
        'criterion':['mse','mae']}
```

```
gcv = GridSearchCV(RandomForestRegressor(), params, cv =5, n_jobs = -1)
gcv.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'criterion': ['mse', 'mae'],
                         'max_depth': [13, 15, 17, 19],
                         'min_samples_split': [3, 4, 5, 6],
                         'n_estimators': [100, 200, 300, 400]})
```

```
gcv.best_params_
```

```
{'criterion': 'mse',
 'max_depth': 15,
 'min_samples_split': 3,
 'n_estimators': 100}
```

```
final = RandomForestRegressor(criterion = 'mae',max_depth = 15, min_samples_split = 3, n_estimators = 100)
final.fit(x_train,y_train)
final_pred = final.predict(x_val)
final_score = final.score(x_val,y_val)
final_score
```

```
0.8953384772458912
```

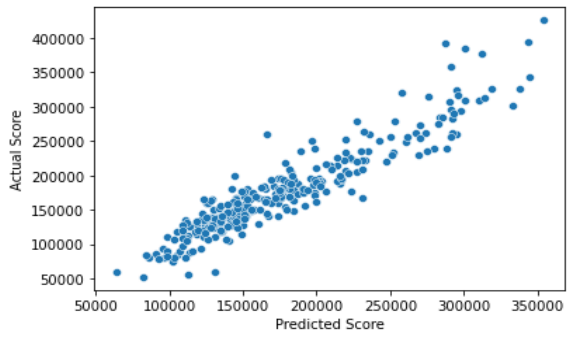
```
final_mae = mean_absolute_error(y_val,final_pred)
print("The mean absolute error for the final model is ", final_mae)
```

```
The mean absolute error for the final model is  16019.478619528621
```

Visualizing the basic Random Forest model and Hyper parameter tuned Random Forest model :

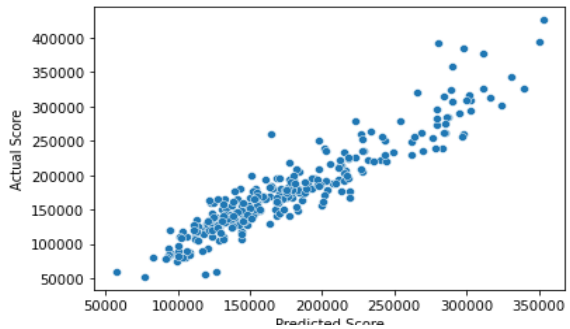
```
sns.scatterplot(x = final_pred, y = y_val)
plt.xlabel('Predicted Score')
plt.ylabel('Actual Score')
```

```
Text(0, 0.5, 'Actual Score')
```




```
sns.scatterplot(x = rfr_pred, y = y_val)
plt.xlabel('Predicted Score')
plt.ylabel('Actual Score')
```

```
Text(0, 0.5, 'Actual Score')
```



Predicting the values for the test data using the values of the trained model :

Preprocessing steps are same as train data :

```
test = pd.read_csv(r"C:\Users\Akanksha\Downloads\Project-Housing_splitted\test.csv")
test.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Cond
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	Corner	Gtl	StoneBr	Norm	
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	StoneBr	Norm	
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl	Crawfor	Norm	
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	Somerst	Feedr	

Removing the unnecessary columns :

```
test = test.drop(columns = ['MiscVal','PoolArea','ScreenPorch','3SsnPorch','EnclosedPorch','BsmtFinSF2',
                            'Exterior2nd','Alley','MiscFeature','PoolQC','Id'])
```

```
from sklearn.impute import IterativeImputer, KNNImputer
```

Feature Engineering :

```
test[['LotFrontage','LotArea']] = knnimp.fit_transform(test[['LotFrontage','LotArea']])
test['MasVnrType'] = test['MasVnrType'].fillna(test['MasVnrType'].mode()[0])
test['MasVnrArea'] = test['MasVnrArea'].fillna(0)
test['BsmtQual'] = test['BsmtQual'].fillna('NA')
test['BsmtCond'] = test['BsmtCond'].fillna('NA')
test['BsmtExposure'] = test['BsmtExposure'].fillna('NA')
test['BsmtFinType1'] = test['BsmtFinType1'].fillna('NA')
test['BsmtFinType2'] = test['BsmtFinType2'].fillna('NA')
test['FireplaceQu'] = test['FireplaceQu'].fillna('NA')
test['GarageType'] = test['GarageType'].fillna('NA')
test['GarageYrBlt'] = test['GarageYrBlt'].fillna(0)
test['GarageFinish'] = test['GarageFinish'].fillna('NA')
test['GarageQual'] = test['GarageQual'].fillna('NA')
test['GarageCond'] = test['GarageCond'].fillna('NA')
test['Fence'] = test['Fence'].fillna('NA')
```

```
test['Electrical'] = test['Electrical'].fillna(test['Electrical'].mode()[0])
```

Encoding the data:

```
encoder = OrdinalEncoder()
for i in test.columns:
    if test[i].dtypes == 'object':
        test[i] = encoder.fit_transform(test[i].values.reshape(-1,1))
```

```
z = np.abs(zscore(test[['LotFrontage','LotArea','MasVnrArea','GarageArea','OpenPorchSF']]))
z.head()
```

	LotFrontage	LotArea	MasVnrArea	GarageArea	OpenPorchSF
0	0.913244	0.263894	0.522510	1.038573	0.059897
1	0.485245	0.363030	0.623319	0.511068	0.715738
2	0.686462	0.089636	0.623319	0.306719	1.580750
3	0.393535	0.101809	0.623319	1.061944	0.715738
4	0.913244	0.297033	0.199362	1.000555	0.441985

```
test_new = test[(z<3).all(axis = 1)]
print(test.shape)
print(test_new.shape)
```

```
(292, 69)
(275, 69)
```

Scaling the data :

```
scal = StandardScaler()
sc = scal.fit_transform(test_new)
test_new = pd.DataFrame(sc, columns = test_new.columns)
```

Using the power transform :

```
tr = power_transform(test_new, method = 'yeo-johnson')
test_new = pd.DataFrame(tr, columns = test_new.columns)
```

Dropping the unnecessary columns :

```
test_new = test_new.drop(columns = 'Utilities')
```

Predicted values :

```
predicted_data = rfr.predict(test_new)
```

```
predicted_data
```

```
array([318192.37, 238550.  , 250698.52, 188752.  , 199111.  ,  87478.5 ,
       142927.19, 318791.18, 249962.58,  80511.83, 142180.16, 126224.08,
       277175.85, 125311.14, 118497.58, 128005.1 , 174299.59, 199536.3 ,
       160912.1 , 149275.37, 158070.82,  79321.5 , 102265.76, 127624.1 ,
       177120.55, 147104.  , 165761.35, 101634.27, 153763.27, 191784.13,
       217787.61, 159578.  , 106945.84, 162140.35, 188858.24, 109397.54,
       154056.  , 150844.72, 107246.74, 315057.66, 203919.55, 182583.8 ,
       131604.66, 128192.5 , 134114.87, 103162.  , 209544.03, 301643.42,
       140682.1 , 181160.24, 110165.72,  98061.16, 128192.5 , 136746.  ,
       192311.3 , 112788.5 , 249204.42, 112619.16, 161872.  , 133351.13,
       145267.6 , 195361.36, 102888.72, 150623.91, 204853.42, 136966.5 ,
       152972.  , 293535.81, 151822.15, 167299.13, 138345.12, 143741.65,
       243218.  , 294145.12, 200107.13, 279856.89, 149179.55, 227513.45,
       137770.85, 143958.15, 169587.02, 258705.72, 110248.89, 314223.89,
       153440.  , 181968.51, 225803.05, 134406.  , 126596.76, 128861.57,
       201576.14, 158812.24, 233768.81, 175063.75, 358345.04, 123374.18,
       258198.86, 100626.68, 116615.66, 154405.01, 205053.88, 144469.06,
       279706.16, 138949.25, 183939.75, 210866.49, 173449.82, 156721.5 ,
       256783.74, 122990.1 , 109117.62, 129001.27, 188629.14, 146419.59,
       109765.24, 118018.72, 201226.79, 284894.93, 128831.49, 139952.25,
       196650.9 , 124370.3 ,  79565.83, 113276.42, 135700.39, 221047.9 ,
       148840.65, 156865.26, 167790.24, 309057.73, 208011.12, 114988.22,
       278926.37, 122098.25, 129317.43,  90494.69, 238077.47, 185120.89,
       118113.85, 100529.98, 201164.7 , 154893.74, 138808.  , 186469.53,
       117010.28, 101077.04, 174525.57, 175647.99, 125739.65, 150598.92,
       195667.18, 145371.12, 186382.45, 121560.  , 115113.26, 250309.51,
       205488.04, 198184.55, 125179.  , 228134.77, 158282.37, 138406.87,
       263889.29, 134931.65, 123377.95, 118150.58, 141962.74, 144522.95,
       212531.49, 163419.72, 254635.59, 167834.99, 380244.12, 325465.37,
       225344.95, 102354.42, 169168.66, 145951.82, 114776.08, 215272.08,
       177336.67,  93161.22, 130087.5 ,  71543.83, 196560.74, 118958.32,
       150666.26, 144670.  , 110760.5 , 280412.25, 267553.56, 129028.45,
       124650.54, 242584.8 , 148799.  , 122569.55, 165543.39, 109272.5 ,
       174566.4 , 134305.04, 191113.92, 100950.64, 250826.79, 145480.  ,
       130063.55, 127442.04, 175115.94, 212714.82, 210199.27, 262738.38,
       138111.  , 178404.5 , 316078.96, 228464.47, 103376.91, 317325.77,
       178033.37, 108812.76, 154768.  ,  97246.16, 144545.  , 115860.85,
       182158.8 , 152596.05, 269046.11, 181189.8 , 213799.18, 145872.4 ,
```

Saving the best model :

```
import joblib
joblib.dump(final,'RealEstatePrediction.pkl')
```

```
['RealEstatePrediction.pkl']
```

CONCLUSION

• Key Findings and Conclusions of the Study:

So, our Aim is achieved as we have successfully ticked all our parameters as mentioned in our Aim Column. It is seen that circle rate is the most effective attribute in predicting the house price and that the Random Forest Regressor is the most effective model for our Dataset with accuracy score 89.5%.

• Learning Outcomes of the Study in respect of Data Science:

Linear Regression is a machine learning algorithm based on supervised learning.

• It performs a regression task. Regression models a target prediction value based on independent variables.

• It is mostly used for finding out the relationship between variables and forecasting. A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging.

• Bagging, in the Random Forest method, involves training each decision tree on a different data sample where sampling is done with replacement.

• The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

• Limitations of this work and Scope for Future Work:

The request contains a list of features, that matches the public dataset's features, that is desired to be available when the data is sent.

There is no guarantee that the data will be available in time nor contains the exact requested list of features. Thus, there might be a risk that the access will be denied or delayed. If so, the study will be accomplished based only on the public dataset.

Moreover, this study will not cover all regression algorithms; instead, it is focused on the chosen algorithm, starting from the basic regression techniques to the advanced ones.