FLIP ROBO

Housing Sales Price Prediction

Submitted By:- Ojasav Sahu

Iternship :- 23

Acknowledgment:-

First of all I would like to thank all my mentors in Data Trained and FlipRobo Technologies for this opportunity. A house is usually the single largest purchase an individual will make in their lifetime. The power of machine learning provides us with the tools we need to look at a large data set and spit out a predicted value, which was our main goal in this project. 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.

Introduction

Business problem framing :

A house is usually the single largest purchase an individual will make in their lifetime. Such significant purchase warrants being well-informed about what a house's selling price should be; for the buyer, as well as the seller or real estate broker involved. The power of machine learning provides us with the tools we need to look at a large data set and spit out a predicted value, which was our main goal in this project. Using a dataset containing information on houses in Ames, Iowa, our team leveraged different machine learning techniques to predict sale prices based on both practical intuition and those observed through our exploratory data analysis and model fitting processes.

You are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know:

- *Which variables are important to predict the price of variable?*
- *How do these variables describe the price of the house?*

• Conceptual Background of the Domain Problem :

In this example 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 neighborhood 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.

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.

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 preprocessing 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 GradientBoosting Regressor when using the public dataset in training. The correlation graphs show the variables' level of dependency.

Analytical Problem Framing:

The given dataset for training the Machine Learning model consists of 1168 rows and 81 columns. Using this dataset, we will be training the Machine Learning models on 70% of the data and the models will be validated on 30% data.

Finally, we will predict the prices for the testing dataset consisting of 275 rows and 69 columns. The provided dataset has null values and we will be imputing the same carefully before we proceed with any pre-processing steps.

- **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 We're going to predict the sale price column ($ USD), let's start with it, Most of the density lies between 100k and 250k, but there appears to be a lot of outliers on the pricier side. Next, let's have a look at the greater living area (square feet) against the sale price. You might've expected that larger living area should mean a higher price. This chart shows you're generally correct. But what are those 2–3 "cheap" houses offering huge living area? One column you might not think about exploring is the "TotalBsmtSF" — Total square feet of the basement area. Intriguing, isn't it? The basement area seems like it might have a lot of predictive power for our model. Ok, last one. Let's look at "OverallQual" — overall material and finish quality. Of course, this one seems like a much more subjective feature, so it might provide a bit different perspective on the sale price. Everything seems fine for this one, except that when you look to the right things start getting much more nuanced. Will that "confuse" our model? We have a more general view on the top 8 correlated features with the sales price.

- **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.*

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 Pre-processing:-

Data pre-processing 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 pre-processing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise and resolving inconsistencies.

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.

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.

. 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

Importing the necessary libraries :

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Data Collection :

In [2]:
```
train = pd.read_csv(r"C:\Users\asus\Downloads\Project-Housing_splitted\train.csv")
pd.set_option("display.max_columns",None)
pd.set_option("display.max_rows",None)
train.head(10)
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Con
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	
5	1197	60	RL	58.0	14054	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	
6	561	20	RL	NaN	11341	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Sawyer	Norm	
7	1041	20	RL	88.0	13125	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	Sawyer	Norm	
8	503	20	RL	70.0	9170	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	Edwards	Feedr	
9	576	50	RL	80.0	8480	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	

Statistical description of the train data :

```
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

1. I can see that the total number of data are 1168 but in few columns, the data is missing.
2. I can see that there are few columns with min value, max value, 25%, 50% and 75% quartile values equal's are "0"
3. I can also see that there are few columns where the value of "Standard deviation" is higher than the "mean" value.

Information of the train data :

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1168 non-null   int64
 1   MSSubClass     1168 non-null   int64
 2   MSZoning       1168 non-null   object
 3   LotFrontage    954 non-null    float64
 4   LotArea        1168 non-null   int64
 5   Street         1168 non-null   object
 6   Alley          77 non-null     object
 7   LotShape       1168 non-null   object
 8   LandContour    1168 non-null   object
 9   Utilities      1168 non-null   object
 10  LotConfig      1168 non-null   object
 11  LandSlope      1168 non-null   object
 12  Neighborhood   1168 non-null   object
 13  Condition1     1168 non-null   object
 14  Condition2     1168 non-null   object
 15  BldgType       1168 non-null   object
 16  HouseStyle     1168 non-null   object
 17  OverallQual    1168 non-null   int64
 18  OverallCond    1168 non-null   int64
 19  YearBuilt      1168 non-null   int64
 20  YearRemodAdd   1168 non-null   int64
 21  RoofStyle      1168 non-null   object
 22  RoofMatl       1168 non-null   object
 23  Exterior1st    1168 non-null   object
 24  Exterior2nd    1168 non-null   object
 25  MasVnrType     1161 non-null   object
 26  MasVnrArea     1161 non-null   float64
 27  ExterQual      1168 non-null   object
 28  ExterCond      1168 non-null   object
 29  Foundation     1168 non-null   object
```

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
```

Removing the unnecessary columns:

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

Removing the columns which have mostly null values :

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

Filling the null values in the columns :

I'll be using 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']])
```

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.

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')
```

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))
```

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

Checking the correlation of the columns with the label :

```
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
```

Printing the highly correlated values :

```
print("Highly Correlated variables with the SalePrice\n",
"""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""")
```

```
Highly Correlated variables with the SalePrice
 OverallQual      0.789185
```

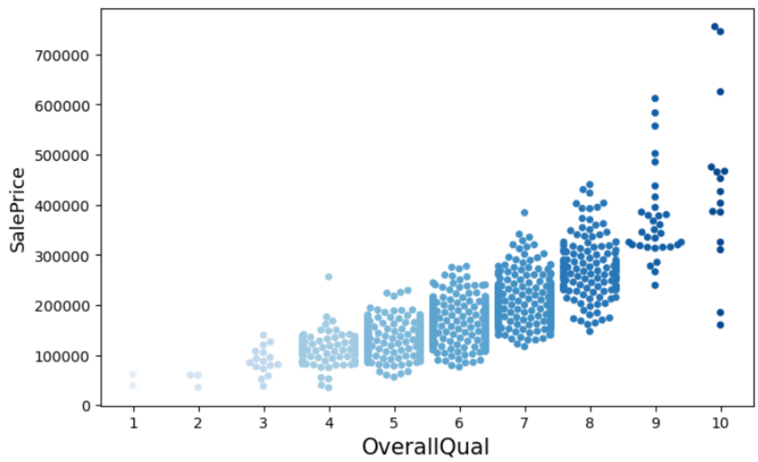
Visualization :

I will visualize the highly correlated values with the target variable :

OverallQual :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'OverallQual',y = 'SalePrice', data = train, palette = 'Blues')
plt.xlabel('OverallQual', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

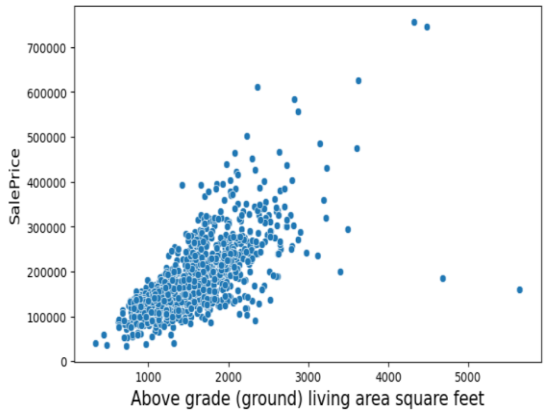
```
Text(0, 0.5, 'SalePrice')
```



GrLivArea :

```
plt.figure(figsize = (8,5),dpi=100)
sns.scatterplot(x = 'GrLivArea',y = 'SalePrice', data = train, palette = 'Blues')
plt.xlabel('Above grade (ground) living area square feet', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

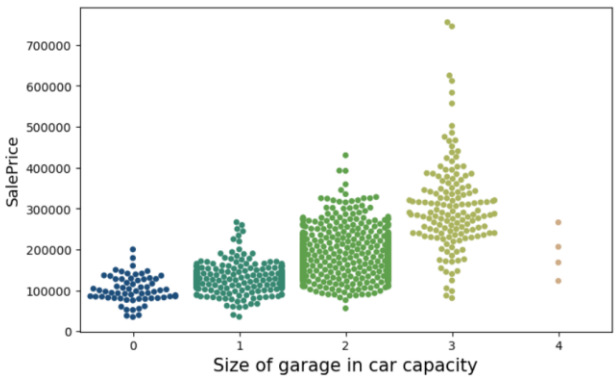


I can see that the density in the scatterplot looks linear and the high density is from 1000 to 2000 at the sales price from 1,00,00 to 3,00,00 and there is no density in the attribute 5000 and alos at the sales price 7,00,000

GarageCars :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'GarageCars',y = 'SalePrice', data = train, palette = 'gist_earth')
plt.xlabel('Size of garage in car capacity', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

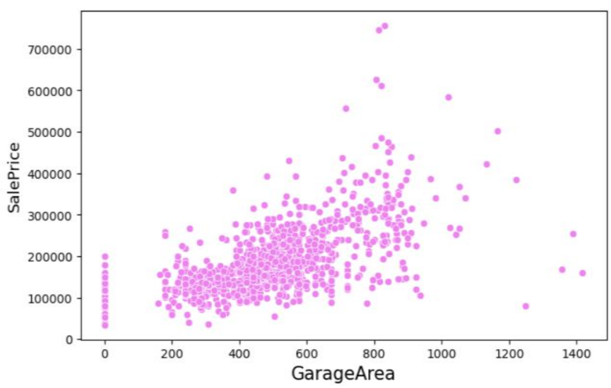


I can see that the high density is present in the attribute 2 and the high dense category is present at the sales price between 1,00,000 to 3,00,000.

GarageArea :

```
plt.figure(figsize = (8,5),dpi=100)
sns.scatterplot(x = 'GarageArea',y = 'SalePrice', data = train, color = 'violet')
plt.xlabel('GarageArea', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

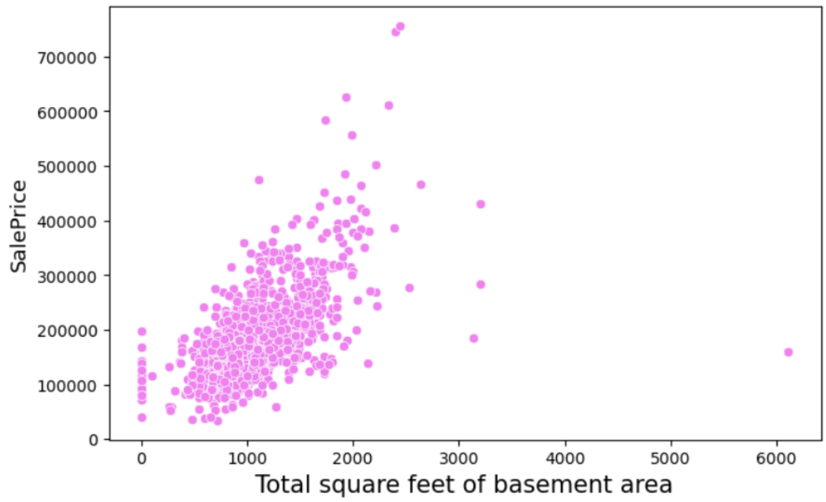


I can see that the high density is present in the attributes between 200 to 600 at the sales price between 1,00,000 to 3,00,000 and the least density is present at the highest points of the columns.

TotalBsmtSF :

```
plt.figure(figsize = (8,5),dpi=100)
sns.scatterplot(x = 'TotalBsmtSF',y = 'SalePrice', data = train, color='violet')
plt.xlabel('Total square feet of basement area', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

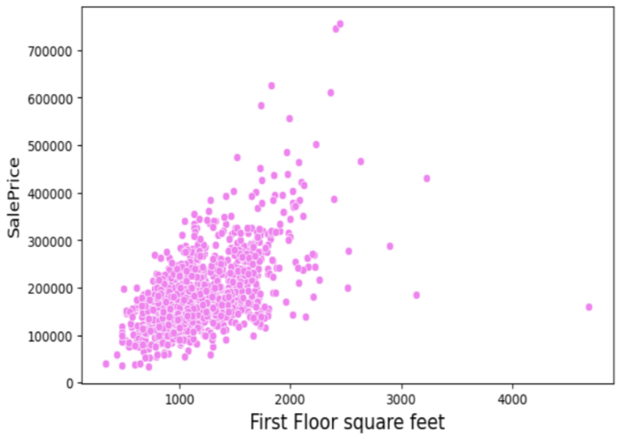


I can see that the high density is present below 2000 and at the sales price between 1,00,000 to 3,00,000

1stFlrSF :

```
plt.figure(figsize = (8,5),dpi=100)
sns.scatterplot(x = '1stFlrSF',y = 'SalePrice', data = train, color = 'violet')
plt.xlabel('First Floor square feet', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

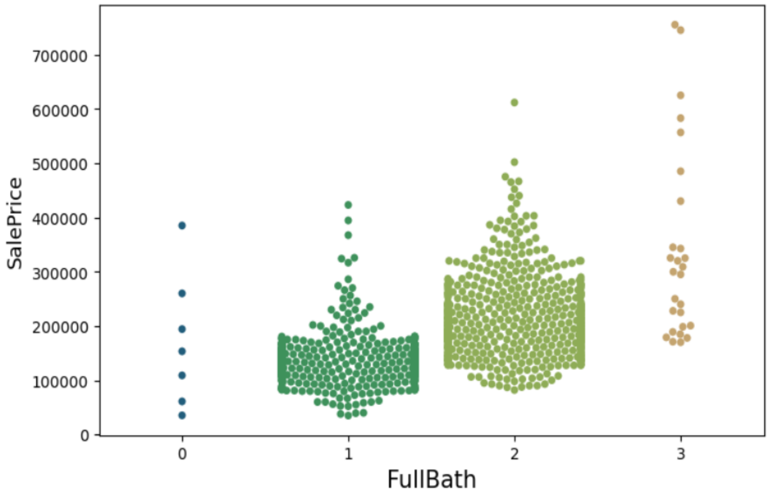


I can see that the high density is present below 2000 in the variable column and in the sales price it is between 1,00,000 to 3,00,000.

FullBath :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'FullBath',y = 'SalePrice', data = train, palette = 'gist_earth')
plt.xlabel('FullBath', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

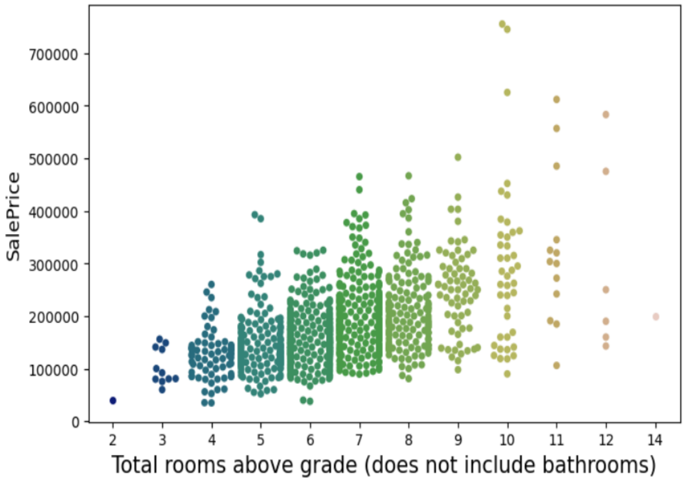


I can see that the high density is present in the column at the attributes 1 and 2 and in the sales price it is between 1,00,000 to 3,00,000

TotRmsAbvGrd :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'TotRmsAbvGrd',y = 'SalePrice', data = train, palette = 'gist_earth')
plt.xlabel('Total rooms above grade (does not include bathrooms)', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

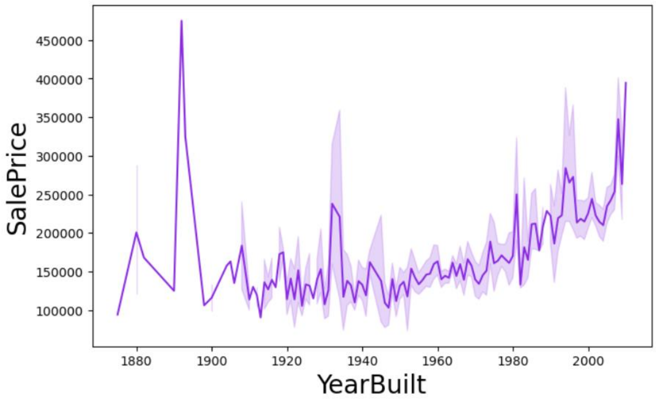


I can see that the high density is present between the attributes 4 to 8 in the variable column and in the sales price it is between 1,00,000 to 3,00,000

YearBuilt :

```
plt.figure(figsize = (8,5),dpi=100)
sns.lineplot(x = 'YearBuilt', y = 'SalePrice', data = train, color = 'blueviolet')
plt.xlabel('YearBuilt', fontsize = 20)
plt.ylabel('SalePrice', fontsize = 20)
```

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

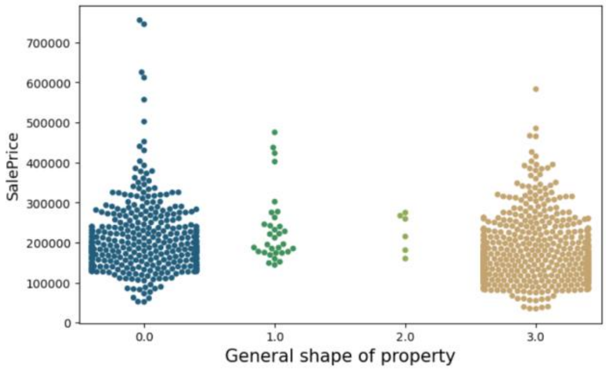


I can see that the high sales price is in the years between 1880 to 1900 reached the highest sales price and the density is highest in the variable column between 1920 to 2000 between the sales prices 1,00,000 to 3,00,000

LotShape :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'LotShape',y = 'SalePrice', data = train, palette = 'gist_earth')
plt.xlabel('General shape of property', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

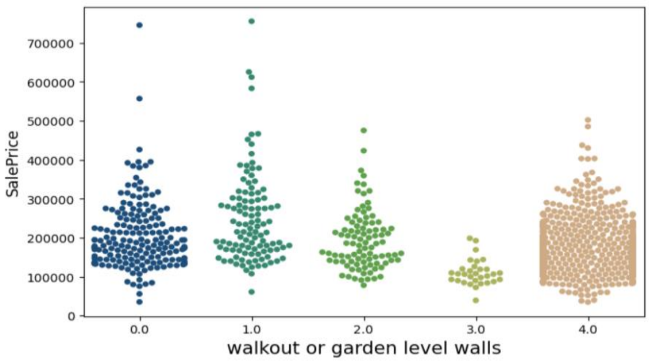


I can see that the highest density is for the attribute 1.0 followed by 0.0 for the variable and ranges between 1,00,000 to 3,00,000 in the label column sales price.

BsmtExposure :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'BsmtExposure',y = 'SalePrice', data = train, palette = 'gist_earth')
plt.xlabel('walkout or garden level walls', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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

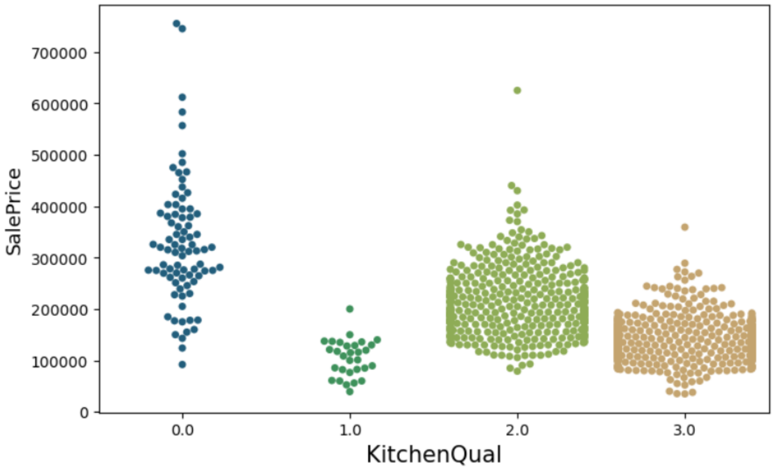


I can see that the highest density is for the attribute 4.0 followed by 0.0 in the variable column ranging from 1,00,000 to 3,50,000 in the label column sales price.

KitchenQual :

```
plt.figure(figsize = (8,5),dpi=100)
sns.swarmplot(x = 'KitchenQual',y = 'SalePrice', data = train, palette = 'gist_earth')
plt.xlabel('KitchenQual', fontsize = 15)
plt.ylabel('SalePrice', fontsize = 13)
```

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



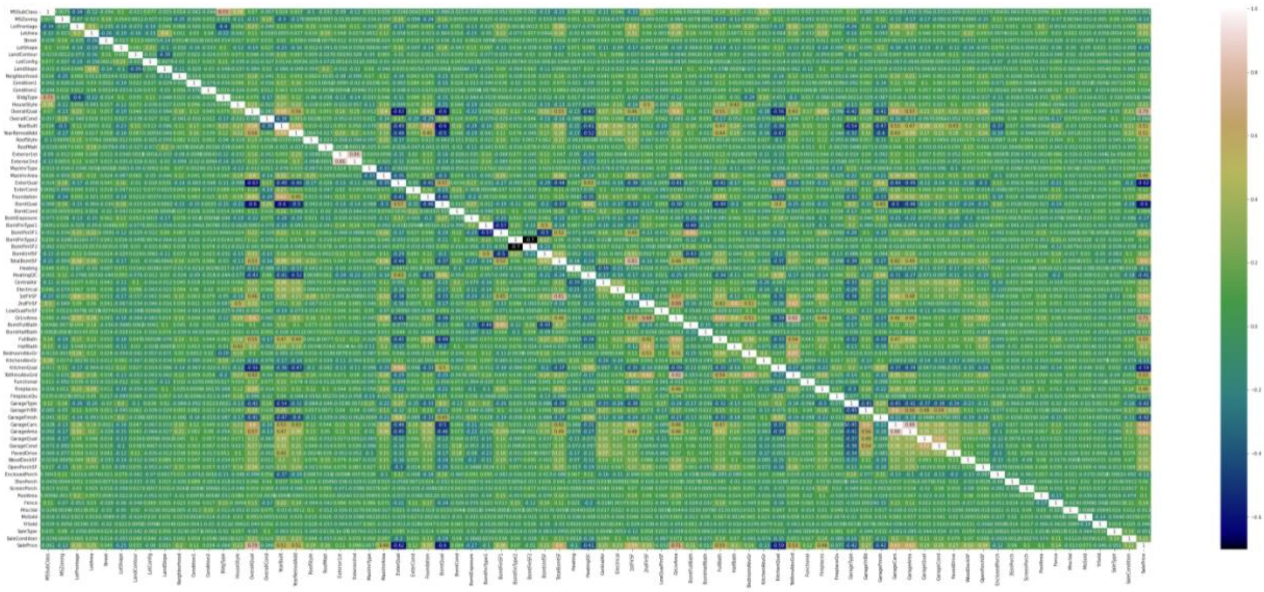
I can see that the attributes 2.0 and 3.0 have high density distribution in variable column ranging from 1,00,000 to 3,80,000 in label column sales price.

Correlation table :

Now, I'll be checking the multicollinearity issue between any of the variables.

```
Correlation = train.corr()
plt.figure(figsize = (55,22),)
sns.heatmap(Correlation, annot = True, cmap = 'gist_earth')
```

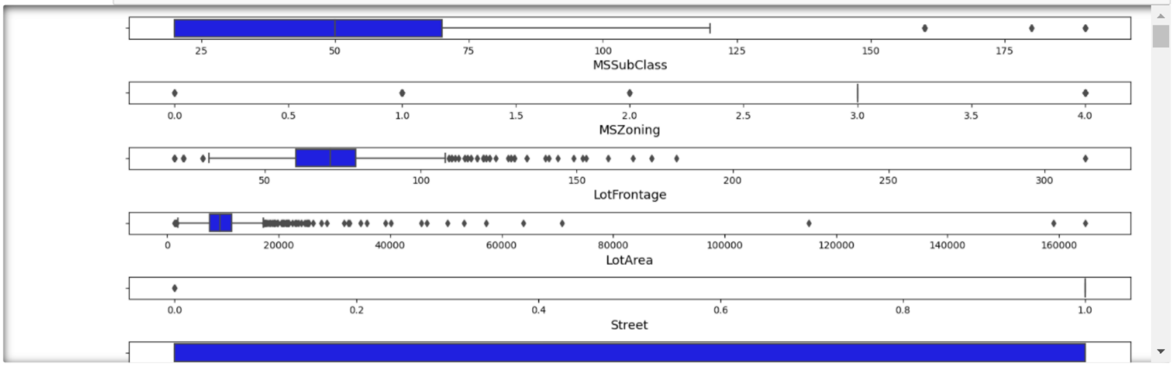
```
<AxesSubplot:>
```



I can see that there are few columns with more than 80% of correlation and few of them are with 65% to 80% of correlation with the label coloumn.

Checking the outliers :

```
In [53]: plt.figure(figsize = (15,75),dpi=100)
pltnum = 1
for i in train:
    if pltnum <= 80:
        plt.subplot(80,1, pltnum)
        sns.boxplot(train[i], color = 'Blue')
        plt.xlabel(i, fontsize = 13)
    pltnum+=1
plt.tight_layout()
```



I can see that most of the columns have outliers so treating these outliers is necessary.

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

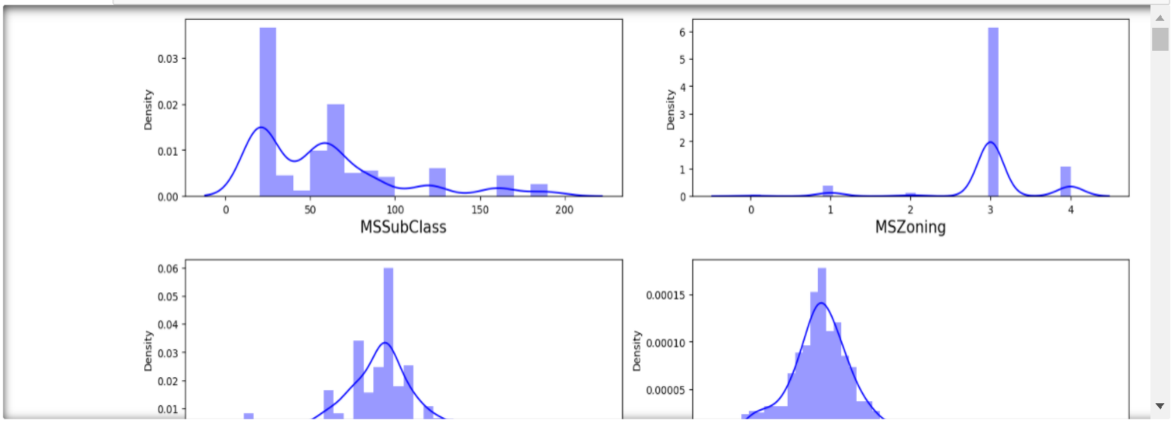
```
train_new = train[(z<3).all(axis = 1)]
print(train.shape)
print(train_new.shape)
```

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

I can see that the number of records change ie., reduced after using z-score method.

Checking the skewness of the data through visualization:

```
In [57]: plt.figure(figsize = (15,120),dpi=100)
pltnum = 1
for i in train_new:
    if pltnum <= 80:
        plt.subplot(40,2, pltnum)
        sns.distplot(train_new[i], color = 'blue')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```



I can see that there are a number of columns with a lot of skewness, some are left skewed and some are right skewed and few of them also have uniform distribution and few are not at all in distribution.

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)
```

Checking the skewness of the data :

```
In [61]: x.skew()

Out[61]: MSSubClass        1.375023
MSZoning         -1.766845
LotFrontage       0.097522
LotArea           1.179904
Street          -22.192273
LotShape         -0.686097
LandContour      -3.256074
LotConfig        -1.202986
LandSlope         5.132759
Neighborhood      0.087352
Condition1        3.205597
Condition2        8.967532
BldgType          2.166364
HouseStyle        0.309815
OverallQual       0.029123
OverallCond       0.590487
YearBuilt        -0.580255
YearRemodAdd     -0.512087
RoofStyle         1.732402
```

I can see that the columns are with mixture of object,int and float datatypes and I have to see to that the columns which are continous variables should have correlation between the range of +0.5 to -0.5

Treating the skewness using power transform method :

In [62]:
```python
from sklearn.preprocessing import LabelEncoder,power_transform
```

In [63]:
```python
train_tr = power_transform(x, method = 'yeo-johnson')
x = pd.DataFrame(train_tr, columns = x.columns)
```

Checking the skewness of the columns :

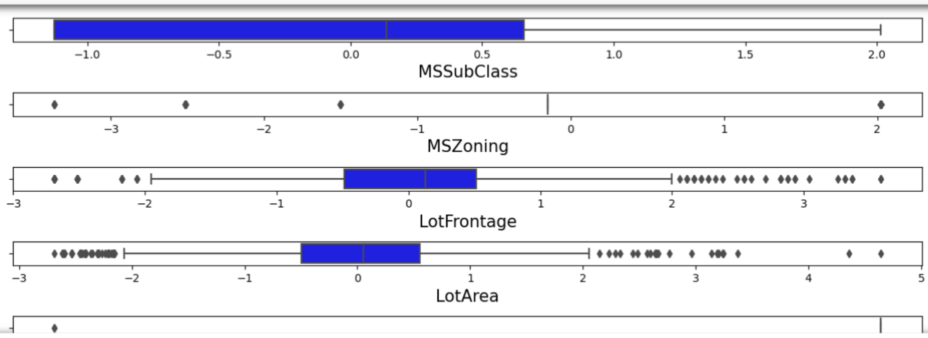
In [64]:
```python
x.skew()
```

Out[64]:
```
MSSubClass        0.220369
MSZoning          0.082024
LotFrontage       0.053543
LotArea           0.013169
Street          -22.192273
LotShape         -0.660805
LandContour      -2.763873
LotConfig        -1.061913
LandSlope         4.310859
Neighborhood      0.012074
Condition1       -0.494157
Condition2        0.282554
BldgType          1.734402
HouseStyle       -0.020586
OverallQual      -0.036828
OverallCond      -0.321125
YearBuilt        -0.134509
YearRemodAdd     -0.210365
RoofStyle        -1.167373
```

I can see that in most of the columns the skewness is reduced.

Re-checking the outliers of the data :

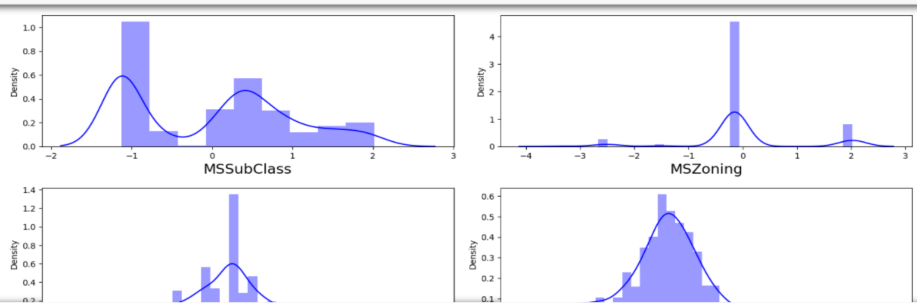
In [65]:
```python
plt.figure(figsize = (12,75),dpi=100)
pltnum = 1
for i in x:
    if pltnum <= 80:
        plt.subplot(80,1, pltnum)
        sns.boxplot(x[i], color = 'blue')
        plt.xlabel(i, fontsize = 15)
    pltnum+=1
plt.tight_layout()
```



I can see that most of the columns have change in their outliers and few columns have outliers in them but that don't have much impact on the label and thereby on model accuracy and so I'll be deleting few of the columns.

Re-checking the skewness of the data :

In [66]:
```python
plt.figure(figsize = (15,120),dpi=100)
pltnum = 1
for i in x:
    if pltnum <= 80:
        plt.subplot(40,2, pltnum)
        sns.distplot(x[i], color = 'blue')
        plt.xlabel(i, fontsize = 17)
    pltnum+=1
plt.tight_layout()
```



I can see that skewness of most of the columns have changed but few don't and few of them have different pattern skewness which are not related with our label column and so I can delete the columns which are unnecessary.

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)
```

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
```

```
0.8968345889491466
```

```
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:  16259.437138047138
```

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
```

```
0.891021210739994
```

```
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:  15145.161649831649
```

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
```

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
```

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
```

```
0.8690564691597671
```

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

```
0.8566554040160117
```

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

```
0.8578754062597206
```

Hyper parameter tuning :

I 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': 5,
 'n_estimators': 200}
```

```
final = RandomForestRegressor(criterion = 'mse',max_depth = 15, min_samples_split = 5, n_estimators = 200)
final.fit(x_train,y_train)
final_pred = final.predict(x_val)
final_score = final.score(x_val,y_val)
final_score
```

```
0.89197416732902
```

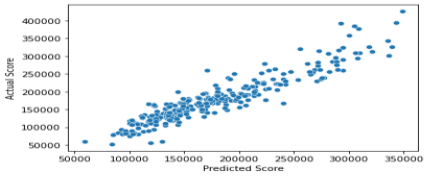
```
final_mse = mean_absolute_error(y_val,final_pred)
print("The mean absolute error for the final model is ", final_mse)
```

```
The mean absolute error for the final model is  16260.56203703704
```

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

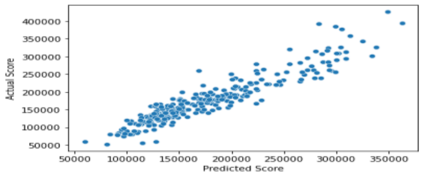
```
sns.scatterplot(x = final_pred, y = y_val)
plt.xlabel('Predicted Score')
plt.ylabel('Actual Score')
```

```
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')
```



I can see that the basic Random Forest model is performinglittle better than the hyper tuned model based on the mean absolute error

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

```
test = pd.read_csv(r"C:\Users\asus\Downloads\Project-Housing_splitted\test.csv")
test.head(10)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Con
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	
5	650	180	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	MeadowV	Norm	
6	1453	180	RM	35.0	3675	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Edwards	Norm	
7	152	20	RL	107.0	13891	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NridgHt	Norm	
8	427	80	RL	NaN	12800	Pave	NaN	Reg	Low	AllPub	Inside	Mod	SawyerW	Norm	
9	776	120	RM	32.0	4500	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Mitchel	Norm	

```
test.isnull().sum()
```

```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage      45
LotArea           0
Street            0
Alley           278
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        1
MasVnrArea        1
ExterQual         0
ExterCond         0
Foundation        0
BsmtQual          7
BsmtCond          7
BsmtExposure      7
BsmtFinType1      7
```

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
```

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

```
array([329124.23, 234312.51, 253591.47, 187904.36, 204827.72,  88178.22,
       139764.35, 323586.92, 243181.02,  80977.83, 141891.6 , 129003.5 ,
       308780.85, 123154.63, 119136.5 , 125001.08, 177953.63, 205911.14,
       166849.3 , 150909.37, 156590.32,  81367.22, 107731.76, 125807.35,
       178833.12, 141953.54, 162858.7 , 104937.03, 154087.98, 192258.28,
       218131.69, 159466.  , 106413.84, 165604.3 , 188013.  , 109122.  ,
       156625.5 , 150450.99, 107175.94, 319542.95, 208233.14, 184729.99,
       128752.51, 128813.84, 132491.75, 107351.71, 207257.3 , 301540.37,
       136634.33, 181330.22, 109368.68, 102230.66, 131165.16, 141393.52,
       193474.89, 113078.37, 249584.16, 109860.11, 164818.03, 131837.95,
       147788.33, 194912.49, 100853.11, 152008.99, 207284.38, 136262.6 ,
       161047.2 , 294301.21, 153395.48, 166820.7 , 141486.29, 145782.44,
       236567.55, 295784.83, 196822.48, 286406.84, 150358.67, 220275.29,
       136977.58, 134402.4 , 174535.24, 262200.4 , 110509.7 , 323373.99,
       153025.75, 183364.98, 229571.14, 132144.6 , 135847.17, 125439.9 ,
       200029.29, 158698.31, 235150.28, 180279.85, 349438.35, 120218.04,
       254378.38,  99828.69, 119320.6 , 149958.66, 206855.9 , 144829.74,
       276005.11, 132931.  , 186066.74, 214502.34, 171732.  , 159776.5 ,
       249260.23, 124590.65, 109831.08, 130637.55, 191285.52, 141557.25,
       106827.66, 116142.61, 200567.75, 284502.17, 130433.15, 142102.  ,
```

Saving the best model :

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

```
['Prediction.pkl']
```

Conclusion:-

I have successfully built a model using multiple models and found that the Random Forest Regressor model. Below are the details of the model's metrics predicting the dataset . R2 score is `16259.437138047138` and Mean Absolute error is `16259.437138047138`.

Limitations of this work and Scope for Future Work:-

- The amount of data is very less, it would be better to have more data to predict the sale price more accurately.
- There are more outliers in the provided data and I was unable to remove all the outliers becau se I could lose data. With more data more outliers can be removed from the dataset.
- Other than these above limitations, I couldn't find more scope for improvement.

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