FLIP ROBO

Housing Project- Predicting Sales Price

Submitted By:

Khushboo Khatri

ACKNOWLEDGMENT

I would like to appreciate Shwetank Mishra for taking time out to clear my doubts and help me final my project. I would like to extend my gratitude to FlipRobo to gave us the opportunity to work with them and DataTrained to taught us ML Model building, which is the key for this project.

Introduction

Problem Statement:

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.
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 at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below.
The company is looking at prospective properties to buy houses to enter the market. 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?

Business Goal:

You 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. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

Technical Requirements:

- Data contains 1460 entries each having 81 variables.
- Data contains Null values. You need to treat them using the domain knowledge and your own understanding.
- Extensive EDA has to be performed to gain relationships of important variable and price.
- Data contains numerical as well as categorical variable. You need to handle them accordingly.
- You have to build Machine Learning models, apply regularization and determine the optimal values of Hyper Parameters.

- You need to find important features which affect the price positively or negatively.
- Two datasets are being provided to you (test.csv, train.csv). You will train on train.csv dataset and predict on test.csv file.

Analytical Problem Framing

- **Mathematical/ Analytical Modeling of the Problem:** Our dataset set consist of total number of 81 columns including the salePrice, and other columns define the various factors of the property, including the condition and quality of material used at different parts of the property.
  To upload the dataset and to work on it, firstly we need to import important libraries, them load the files and then have a loom on the data present in it.

```python
# importing basic libraries
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# warnings
import warnings
warnings.filterwarnings('ignore')
```

```python
# getting test and train dataset
df_train=pd.read_csv(r'C:\Users\DELL\OneDrive\Documents\Project-Housing_splitted\train.csv')
df_test=pd.read_csv(r'C:\Users\DELL\OneDrive\Documents\Project-Housing_splitted\test.csv')
```

```python
pd.set_option('display.max_columns' ,None)       # To display all the columns value of dataset
df_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1
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

Here we use pd.set_option( ) in order to see all the columns of this large datset.

- **Data Sources and their Formats:**

The Housing Project consist of two csv file, one is for train data and other is for test data. We train our Model on train data, then use that model predict the sale price of test dataset. The dimension of train data and test data are as follows:

```
Train Dataset: (1168, 81)
Test Dataset: (292, 80)
```

We can see that for our test dataset we have only 80 columns while train dataset has 81 columns, th at one extra column is of SalePrice, which is given in train dataset in order to train our model. Here a glimse of all the information about dataset, including their non null values and datatype. The below description is for train dataset, we did perform similar function for test dataset. Let's have a look

```
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 
 30  BsmtQual       1138 non-null   object 
 31  BsmtCond       1138 non-null   object 
 32  BsmtExposure   1137 non-null   object 
 33  BsmtFinType1   1138 non-null   object 
 34  BsmtFinSF1     1168 non-null   int64  
 35  BsmtFinType2   1137 non-null   object 
 36  BsmtFinSF2     1168 non-null   int64  
 37  BsmtUnfSF      1168 non-null   int64  
 38  TotalBsmtSF    1168 non-null   int64  
 39  Heating        1168 non-null   object 
 40  HeatingQC      1168 non-null   object 
 41  CentralAir     1168 non-null   object 
 42  Electrical     1168 non-null   object 
 43  1stFlrSF       1168 non-null   int64  
 44  2ndFlrSF       1168 non-null   int64  
 45  LowQualFinSF   1168 non-null   int64  
 46  GrLivArea      1168 non-null   int64  
 47  BsmtFullBath   1168 non-null   int64  
 48  BsmtHalfBath   1168 non-null   int64  
 49  FullBath       1168 non-null   int64  
 50  HalfBath       1168 non-null   int64  
 51  BedroomAbvGr   1168 non-null   int64  
```

```
 52  KitchenAbvGr   1168 non-null   int64
 53  KitchenQual    1168 non-null   object
 54  TotRmsAbvGrd   1168 non-null   int64
 55  Functional     1168 non-null   object
 56  Fireplaces     1168 non-null   int64
 57  FireplaceQu    617 non-null    object
 58  GarageType     1104 non-null   object
 59  GarageYrBlt    1104 non-null   float64
 60  GarageFinish   1104 non-null   object
 61  GarageCars     1168 non-null   int64
 62  GarageArea     1168 non-null   int64
 63  GarageQual     1104 non-null   object
 64  GarageCond     1104 non-null   object
 65  PavedDrive     1168 non-null   object
 66  WoodDeckSF     1168 non-null   int64
 67  OpenPorchSF    1168 non-null   int64
 68  EnclosedPorch  1168 non-null   int64
 69  3SsnPorch      1168 non-null   int64
 70  ScreenPorch    1168 non-null   int64
 71  PoolArea       1168 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          237 non-null    object
 74  MiscFeature    44 non-null     object
 75  MiscVal        1168 non-null   int64
 76  MoSold         1168 non-null   int64
 77  YrSold         1168 non-null   int64
 78  SaleType       1168 non-null   object
 79  SaleCondition  1168 non-null   object
 80  SalePrice      1168 non-null   int64
dtypes: float64(3), int64(35), object(43)
```

Here we use info() to get a short summary of our dataframe, it tells us about number if columns, there indexs, their datatypes, total number of non null values present in the dataframe.

From here we observe that Alley, PoolQC, Fence, MiscFeature has more number of null values than the provided info, we can simply drop them, if it's the same condition for test dataset as well. Also column Id is merely an identification number, which is not of use in ML model building, so we will drop that as well.

Other columns with less null values are : LotFrontage, MasVnrType, MasVnrArea, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, FireplaceQu, GarageType, GarageYrBlt, GarageFinish, GarageQual, GarageCond. We need to treat them as per their data type.

If it is an Object datatype column, them we will impute it with the mode of the column, and If the datatype is int64 or float64 something like that means its continuous data type, and we can impute them either with median/ mean of the column.

Our Dataset consist of both Categorical Column and Continuous Column.

Categorical columns are those columns with dtype as Object, while continuous data are those with dtype as int64/float64 here in this case.

And our label is 'SalePrice'. As we need to predict the sale Price of the property.
Let's describe the each columns, as it helps us to identify what can be the content of it:

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 available

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 remodeling 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 sale

One liner description of each column has been described. Let’s move ahead with data preprocessing.

Data Preprocessing :

From info() we can see that certain columns had so many null values, that if we impute them there will be biasness, so its better to drop them, We also observe that ID column is there which also don’t add much value in ML model making, so we can drop them as well. For that we will use drop ()

```
# Drop Columns, Alley, PoolQC, Fence, MiscFeature
df_train.drop(columns=['Alley', 'PoolQC', 'Fence', 'MiscFeature','Id'], axis=1, inplace = True)
df_test.drop(columns=['Alley', 'PoolQC', 'Fence', 'MiscFeature','Id'], axis=1, inplace = True)


#checking shape
print(df_train.shape)
print(df_test.shape)
```

```
(1168, 76)
(292, 75)
```

Next, we need to separate categorical columns from continuous columns in the dataframe, this will help us to understand and analyse the dataset in a better manner.

```python
# Separating Categorical and continuous columns
cat_data=[]
num_data=[]
for column in df_train.columns:
    if df_train[column].dtype == object:
        cat_data.append(column)

    else:
        num_data.append(column)
print("categorical columns are : ",cat_data)
print("\nTotal number of Categorical Columns : ",len(cat_data))
print("\n\nContinuous columns are : ", num_data)
print("\n Total number of Numerical Columns : ", len(num_data))
```

```
categorical columns are :  ['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighbo
rhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnr
Type', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heat
ing', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'Ga
rageQual', 'GarageCond', 'PavedDrive', 'SaleType', 'SaleCondition']

Total number of Categorical Columns :  39


Continuous columns are :  ['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAd
d', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivAre
a', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'G
arageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolAre
a', 'MiscVal', 'MoSold', 'YrSold', 'SalePrice']

 Total number of Numerical Columns :  37
```

We can observe that we have total number of 39 categorical columns while 37 numerical columns. One thing to note that we are following same steps for test dataset as well.

Also, here we can observe that categorical data is further divided into Nominal data and ordinal data. Here, let's differentiate both the columns, as it will be helpful during EDA and encoding.

1. Nominal Columns: 'MSZoning', 'Street','LotShape', 'LandContour', 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st','Heating','CentralAir', 'Exterior2nd', 'MasVnrType', 'Foundation', 'Electrical', 'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'PavedDrive', 'SaleType', 'SaleCondition'

2. Ordinal data: 'ExterQual', 'ExterCond','BsmtQual','BsmtCond','BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'HeatingQC','KitchenQual','FireplaceQu','GarageQual', 'GarageCond'.

Now, we need to find unique values in each columns, let's start with numerical columns.

We will us nuinque() to identify number of unique values in each numerical column. This will help us to observe how the data can be.

We also perform somewhat similar function in categorical columns to see how our categorical columns are distributed between various classes of the features.

In case of categorical data we use value_counts() to count the values of different unique classes of categorical columns. And unique () is use to determine the unique values of the columns.

```
# checking number of unique values in each column (numerical columns)
df_train[num_data].nunique()
```

```
0]: MSSubClass        15
    LotFrontage      106
    LotArea          892
    OverallQual       10
    OverallCond        9
    YearBuilt        110
    YearRemodAdd      61
    MasVnrArea       283
    BsmtFinSF1       551
    BsmtFinSF2       122
    BsmtUnfSF        681
    TotalBsmtSF      636
    1stFlrSF         669
    2ndFlrSF         351
    LowQualFinSF      21
    GrLivArea        746
    BsmtFullBath       4
    BsmtHalfBath       3
    FullBath           4
    HalfBath           3
    BedroomAbvGr       8
    KitchenAbvGr       4
    TotRmsAbvGrd      12
    Fireplaces         4
    GarageYrBlt       97
    GarageCars         5
    GarageArea       392
    WoodDeckSF       244
    OpenPorchSF      176
    EnclosedPorch    106
    3SsnPorch         18
    ScreenPorch       65
    PoolArea           8
    MiscVal           20
    MoSold            12
```

Above shows value for numerical columns and the below one shows the values for categorica; columns.

```
ince categorical column has object datatype we will print all of the object data types and their unique values.

 column in df_train.columns:
if df_train[column].dtype == object:    #checking datatype for each column if it is 'object'
    print(str(column) + ' : ' + str(df_train[column].unique()))   #unique() gives all the unique value of that column
    print(df_train[column].value_counts())  # value_counts() count the number belongs to different class in that column
    print("\n ____________________________________________________________________\n")
```

```
ConLw      4
Oth        3
CWD        3
Con        2
Name: SaleType, dtype: int64


 ____________________________________________________________________


SaleCondition : ['Normal' 'Partial' 'Abnorml' 'Family' 'Alloca' 'AdjLand']
Normal     945
Partial    108
Abnorml     81
Family      18
Alloca      12
AdjLand      4
Name: SaleCondition, dtype: int64
```

Since our data has been separated as categorical and continuous columns, let’s gop ahead and perform some EDA and Feature Engineering, before Imputing the missing columns and encoding the categorical columns.

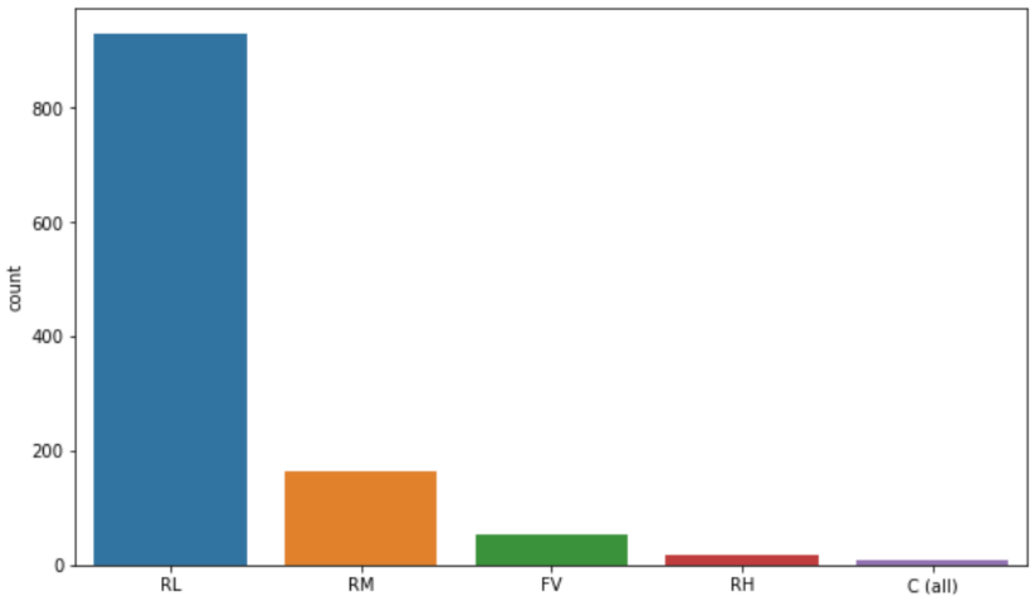
EDA:

We will start with univariate analysis, this will help use to understood how data is distributed on each columns. And help us to get some insights about the data.

In EDA we generally draw countplot for categorical columns and distplot for continuous columns.

```
# countplot for categorical columns
plt.figure(figsize=(10,6))
sns.countplot('MSZoning', data=df_train)
```

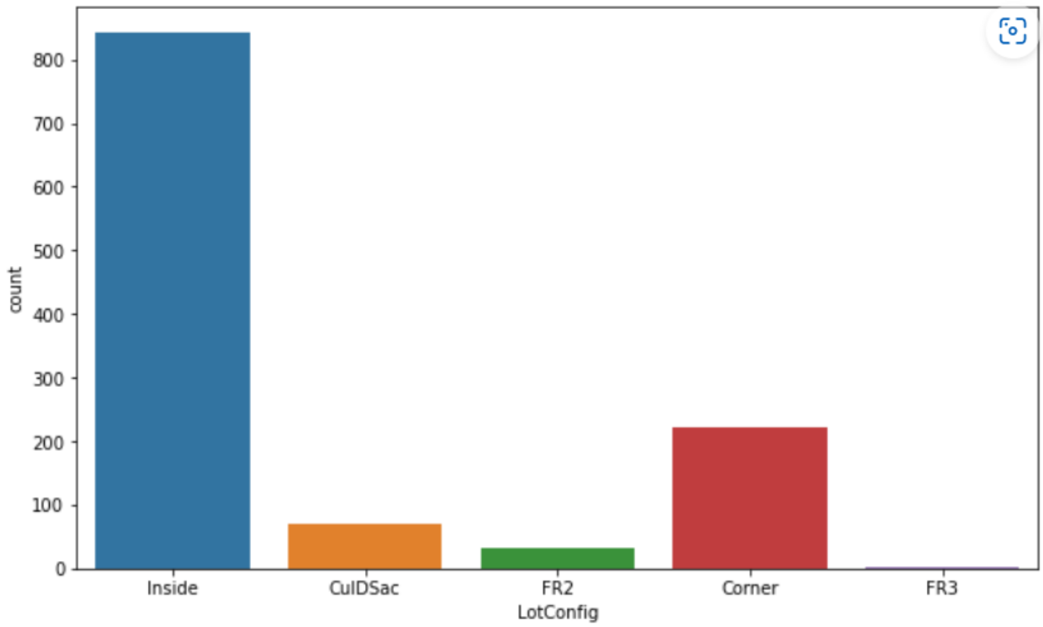
```
<AxesSubplot:xlabel='MSZoning', ylabel='count'>
```



We can observe that Residential low density zoning has highest number of counts then rest of them, that might be reason why people in Australia prefer low density residential area.

```
plt.figure(figsize=(10,6))
sns.countplot('LotConfig', data=df_train)
```

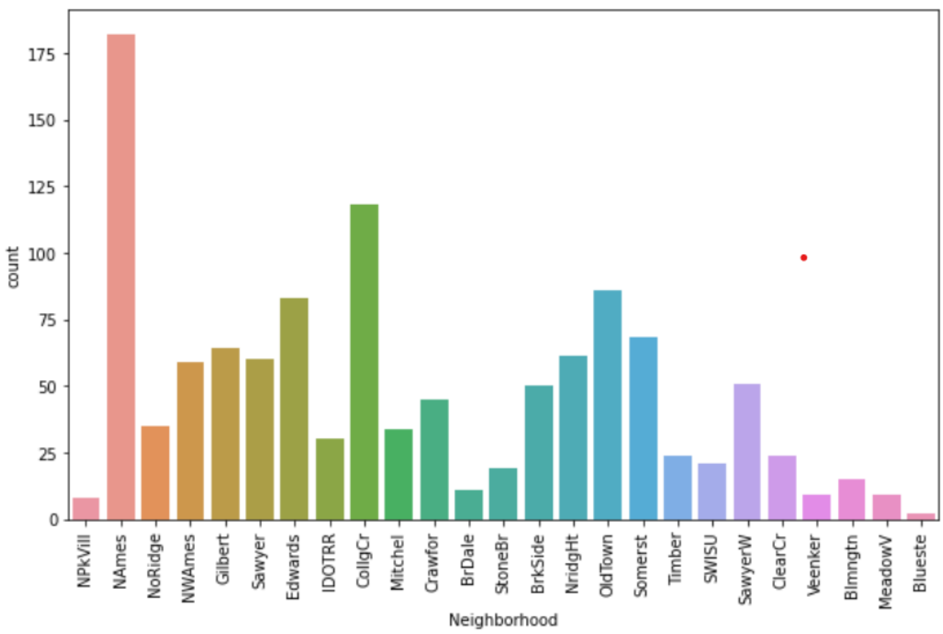
<AxesSubplot:xlabel='LotConfig', ylabel='count'>



We can observe that Generally people prefer inside plots followed by corner.

```
plt.figure(figsize=(10,6))
sns.countplot('Neighborhood', data=df_train)
plt.xticks(rotation=90)
plt.show
```

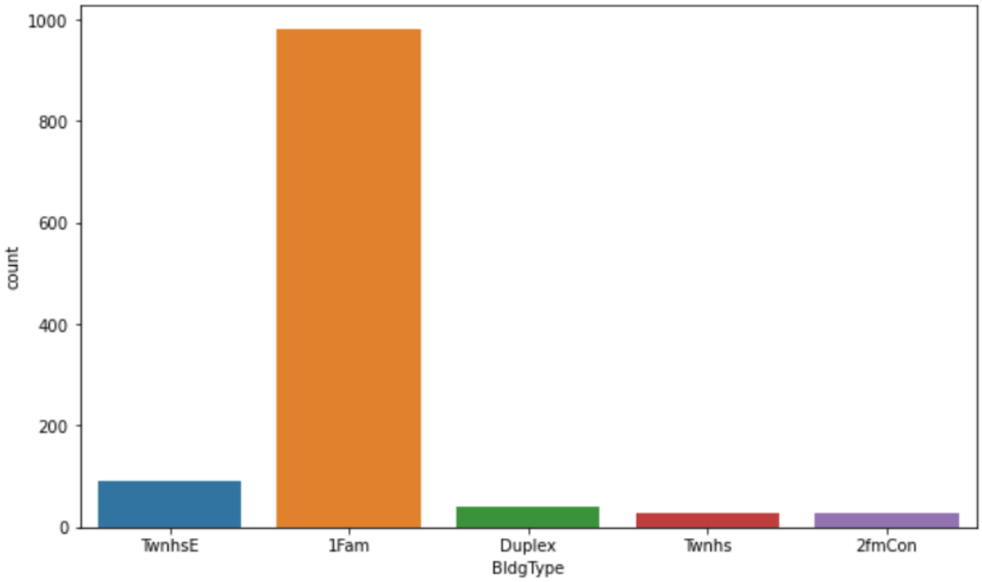
]: <function matplotlib.pyplot.show(close=None, block=None)>



Looks Like N Ames neighbourhood seems to be the most popular among the rest. The values of property there definitely gives good value.

```
plt.figure(figsize=(10,6))
sns.countplot('BldgType', data=df_train)
```

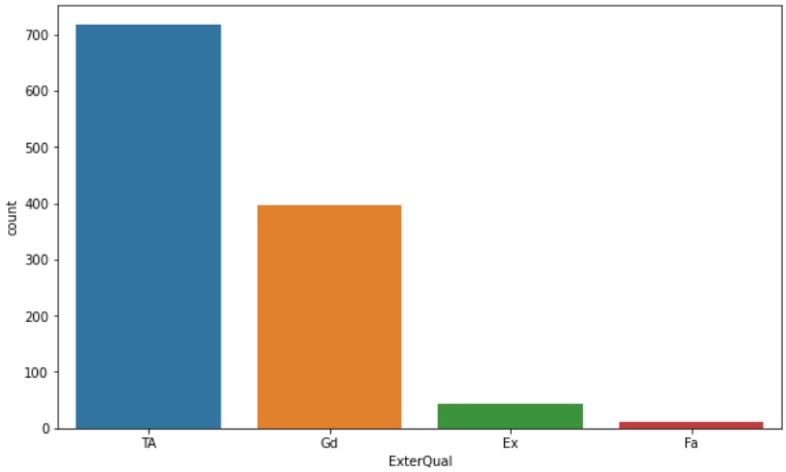
]: <AxesSubplot:xlabel='BldgType', ylabel='count'>



Single family detached house are preferred once over the rest. We can see that kind of property are high on sales as well.

```
plt.figure(figsize=(10,6))
sns.countplot('ExterQual',data=df_train)
```

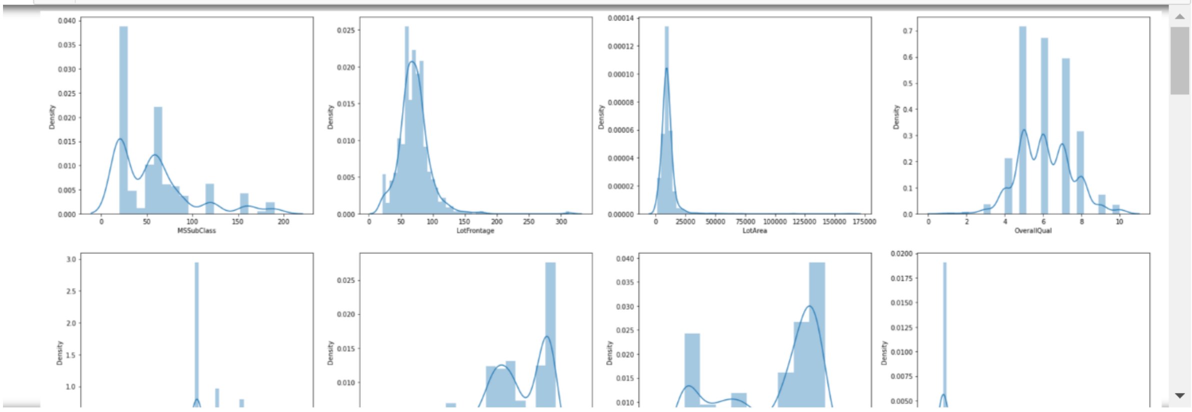
8]: <AxesSubplot:xlabel='ExterQual', ylabel='count'>



from above observation TA stands for typical/average, Gd means good, Ex means excellent, Fa means fair exterior quality.

We did plot some more categorical count plots to see how the data looks like. Then we plot distplot for all the numerical columns using one set of code. In that way we are able to see the distribution of data for all the numerical columns, in single plot.

```python
# let's see how data is distributed among each column. we are not including SalesPrice here
plt.figure(figsize=(30,60), facecolor='white')
plotnumber=1
for column in df_train[num_data]:
    if plotnumber<=36:
        ax=plt.subplot(9,4,plotnumber)
        sns.distplot(df_train[column])
    plotnumber+=1
plt.show()
```

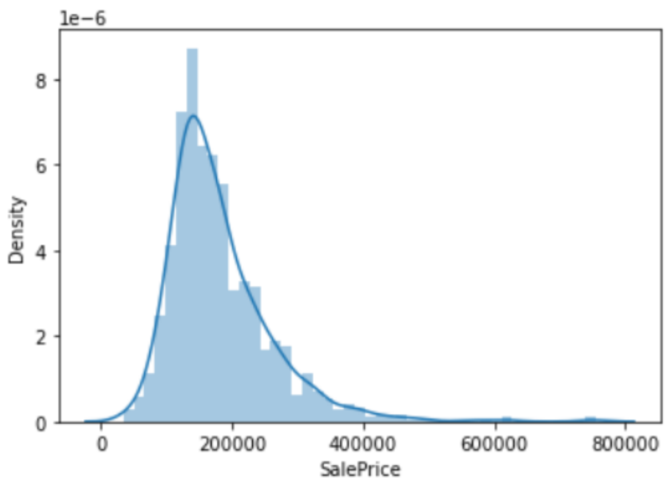


From here we can observe that some of the numerical columns has discrete value while others has continuous values.

We also noticed that few of the columns has outliers, as we can see that their distribution lines extend beyond normal distribution curves. For discrete data having multiple peaks are column.

```python
sns.distplot(df_train['SalePrice'])
```

```
<AxesSubplot:xlabel='SalePrice', ylabel='Density'>
```



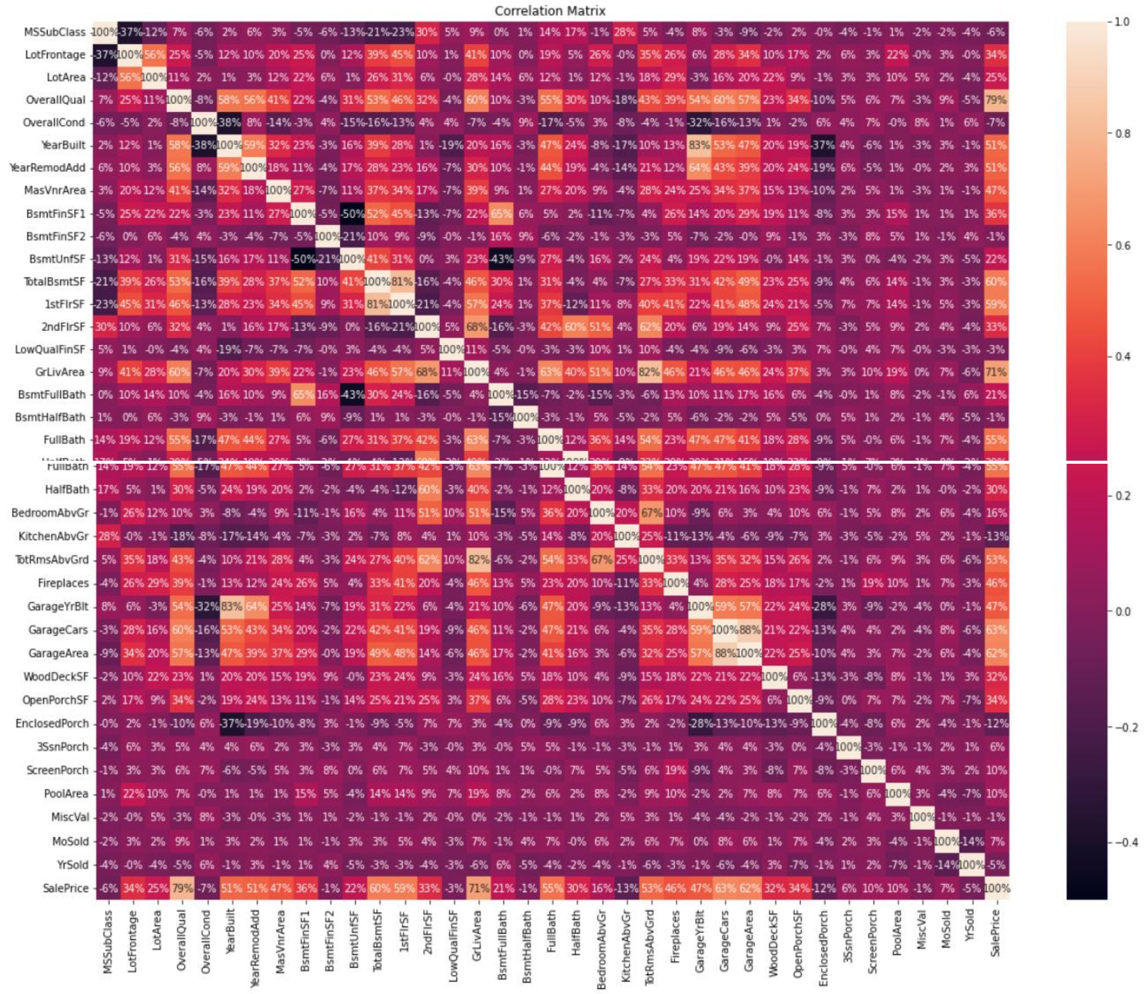
From above observation we can see how data is distributed in our label column, in this case it is SalePrice.

Now Let's start Multivariate analysis, as it helps to see logical relation between various columns.

Let's start with correlation matrix , as it helps at this point to understand relationship between various numerical features and columns. It also helps us to see till how much degree the feature is correlated with label.

One thing to note here that we are also able to see correlation between various independent features as well.

```python
# Let's plot heatmap we visualise the correlation of continuous features and  our target variable.

plt.figure(figsize=(20,15))
sns.heatmap(df_train.corr(), annot= True, fmt='.0%')
plt.title("Correlation Matrix")
plt.show()
```

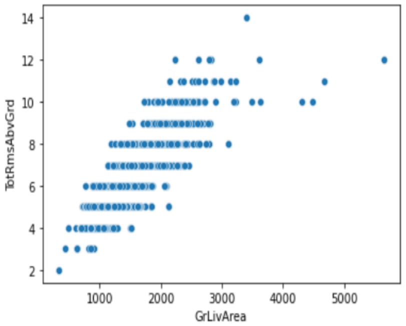


Observations:

- We can observe that BsmtFinSF2, BsmtHalfBath, MiscVal are least correlated features followed by LowQualFinSF. We can drop them.
- While OverallQual, followed by GrLivArea are highest correlated features.
- We can also observe that some of the features are highly correlated with each other as well, like : GarageCars-GarageArea, GrLivArea-TotRmsAbvGrd, GarageYrBlt-YearBuilt.
- Let's confirm their correlation using scatterplot ,then we need to drop one out of the two, so that we don't see multicolinearity in our data

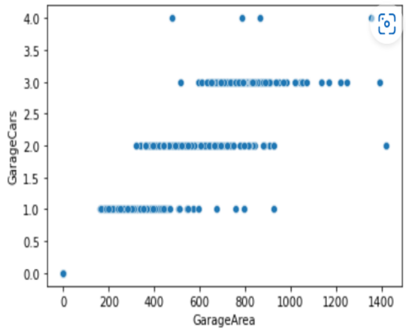
```
sns.scatterplot('GrLivArea','TotRmsAbvGrd', data=df_train)
```

```
<AxesSubplot:xlabel='GrLivArea', ylabel='TotRmsAbvGrd'>
```




```
sns.scatterplot('GarageArea','GarageCars', data=df_train)
```

```
<AxesSubplot:xlabel='GarageArea', ylabel='GarageCars'>
```



We can observe that GrLivArea- TotRmsAbvGrd are positively correlated with each other, we can drop one of them in order to avoid multicollinearity.

From GarageArea-GarageCars we don't find any strong correlation, although value of GarageArea increase with GarageCars, however it is not certain, as for GarageCars4, we don't see similar relation.

Now let's drop the un important columns and also the columns which shows multicollinearity.

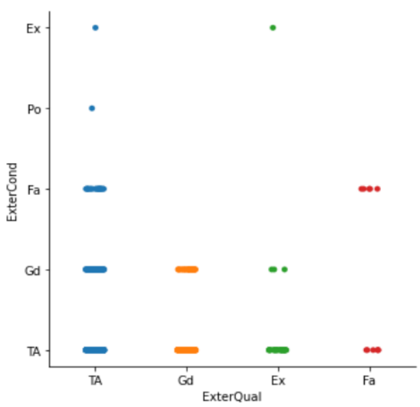
```
# Let's drop the columns which are least correlated and shows multicolinearity.
# from train dataset
df_train.drop(columns= ['BsmtFinSF2', 'BsmtHalfBath', 'MiscVal','LowQualFinSF', 'TotRmsAbvGrd'], axis=1, inplace=True)

#From test dataset
df_test.drop(columns=['BsmtFinSF2', 'BsmtHalfBath', 'MiscVal','LowQualFinSF', 'TotRmsAbvGrd'], axis=1, inplace=True)

print("The dimension of Train dataset : ",df_train.shape)
print("The dimension of test dataset : ",df_test.shape)
```

```
The dimension of Train dataset :  (1168, 71)
The dimension of test dataset :  (292, 70)
```

In EDA we generally plot catplot to show relation between ordinal columns. Let's have a look.

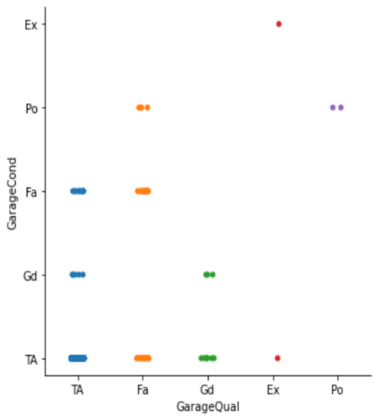
```
sns.catplot(x='ExterQual', y='ExterCond', data=df_train)
```

```
<seaborn.axisgrid.FacetGrid at 0x1fac78ef370>
```




```
sns.catplot(x='GarageQual', y='GarageCond', data=df_train)
```

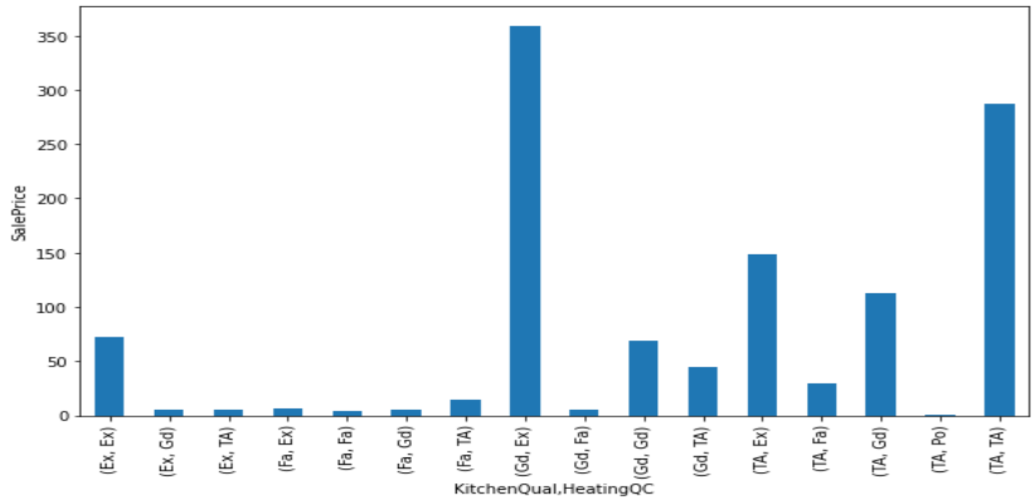
```
<seaborn.axisgrid.FacetGrid at 0x1fac654bfd0>
```



we can see that there is less difference between condition and quality of both Exterior and garage.

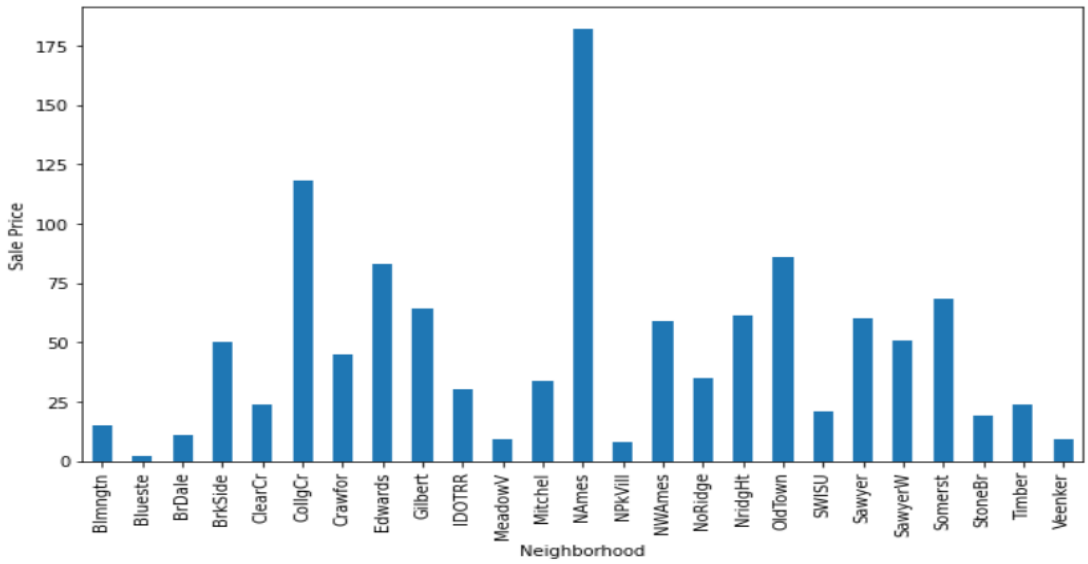
We also draw some groupby plots to see how they are related with SalePrice.

```
plt.figure(figsize=(10,6))
df_train.groupby(['KitchenQual','HeatingQC']).SalePrice.count().plot.bar(ylim=0)
plt.ylabel('SalePrice')
plt.show()
```



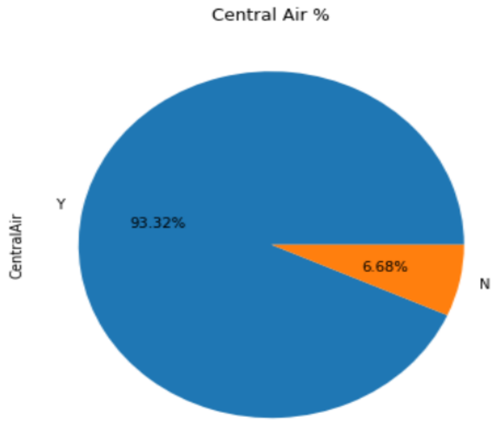
From above observation we can say that both Kitchen Quality and Heating Quality are related to SalePrice, we can see that for Good Quality Kitchen and Excellent Heating Quality got highest sales price.

```
plt.figure(figsize=(10,6))
df_train.groupby('Neighborhood').SalePrice.count().plot.bar(ylim=0)
plt.ylabel('Sale Price')
plt.show()
```



We already establish that North Ames Neighborhood got highest SalePrice, It is the most popular neighborhood to live in. While Blueste Neighborhood has lowest SalePrice.

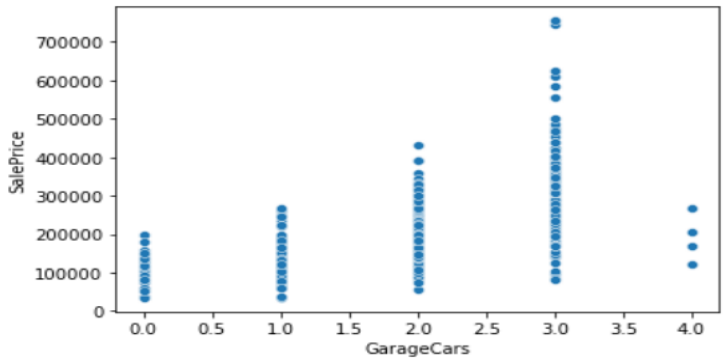
```
plt.figure(figsize=(10,6))
(df_train['CentralAir'].value_counts()*100.0 /len(df_train)).plot.pie(autopct='%.2f%%')
plt.title("Central Air %")
plt.show()
```



We can Observe that Central Air system is present in most of the properties.

```
sns.scatterplot('GarageCars','SalePrice', data=df_train)
```

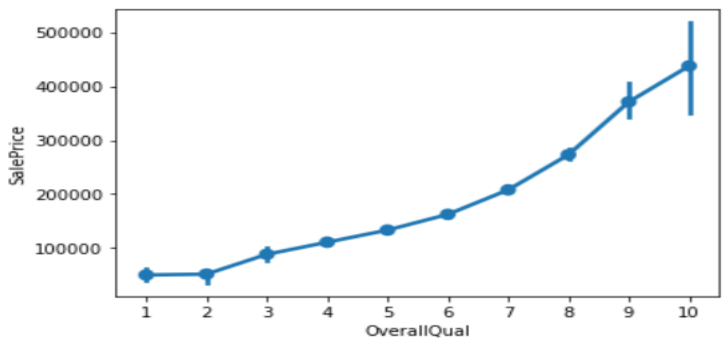
]: <AxesSubplot:xlabel='GarageCars', ylabel='SalePrice'>



From above we can observe that GarageCars 3 has highest sale price.

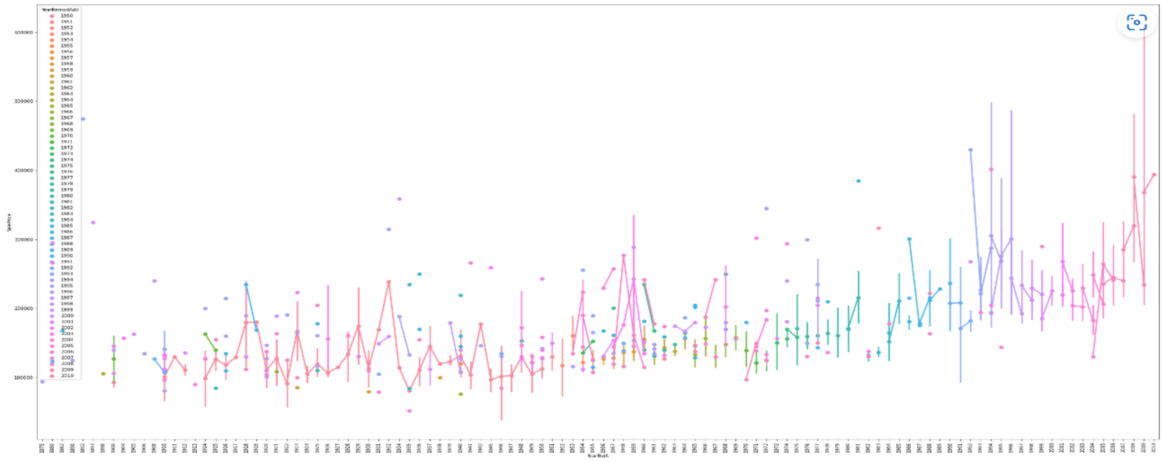
```
sns.pointplot(x='OverallQual', y='SalePrice', data=df_train)
```

]: <AxesSubplot:xlabel='OverallQual', ylabel='SalePrice'>



We can SalePrice increases as overall quality increases.

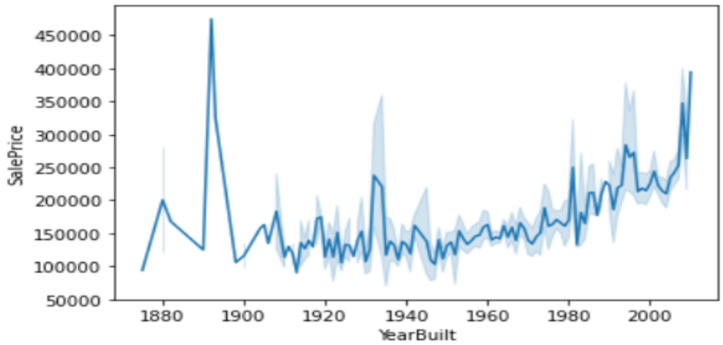
```python
plt.figure(figsize=(40,20))
sns.pointplot(x='YearBuilt', y='SalePrice', hue= 'YearRemodAdd', data=df_train)
plt.xticks(rotation=90)
plt.show()
```



We can observe that the sale price for remodel houses are higher, The house Built in year 2010 or Remodel from 1990-2010 gets good sale Price.

```python
sns.lineplot(x='YearBuilt', y='SalePrice', data=df_train)
```

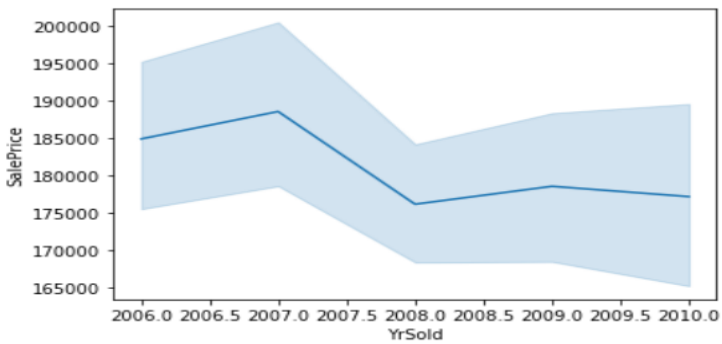
```
]: <AxesSubplot:xlabel='YearBuilt', ylabel='SalePrice'>
```



We can observe from above plot that the sales price were highest for ancient 1890-1900 well build well maintained property, then the sale price drop and gradually increases with YearBuilt.

```python
sns.lineplot(x='YrSold', y='SalePrice', data=df_train)
```

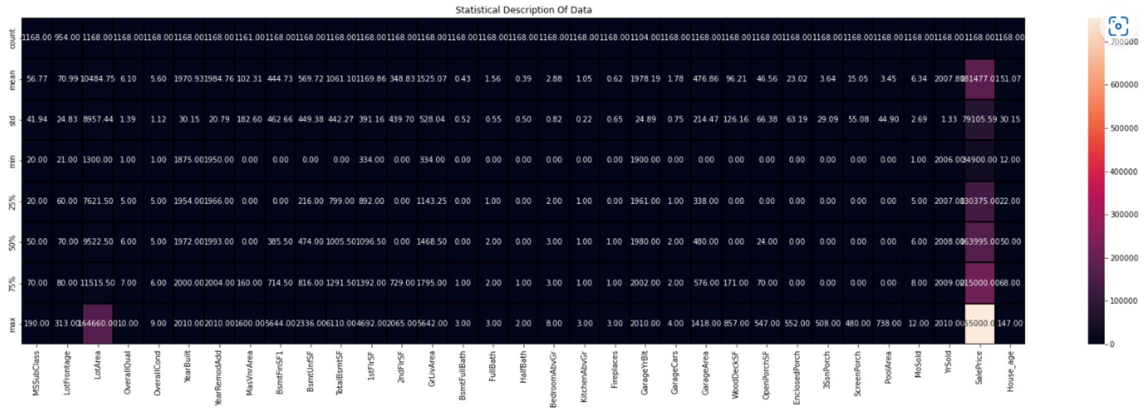
```
]: <AxesSubplot:xlabel='YrSold', ylabel='SalePrice'>
```



We can observe that the sale price of houses increases from 2006-2007 then decreases in 2008 may be because of recession. then they never hit high again.

Let's Visualise the statistical Analysis of Train dataset.

```
# Let's Visualize it using heat map
plt.figure(figsize=(30,8))
sns.heatmap(df_train.describe(), annot= True,linewidths=0.2,linecolor='black', fmt='.2f')
plt.title("Statistical Description Of Data")
plt.show()
```



We can see that our numerical data, looks good. 25% data represents q1 while 75% represents q3. we also got the min and max data of every column. While looking at the mean and std we can say that we need to scale our data before ML model building.

Handling Missing Values:

As we observe the missing values belongs to same columns in both test and train dataset, so its better to merge both the dataset, to impute those missing values and later we will do Encoding also in this merge data.

Following are the steps involve in Handling the missing values :

1. Separate Categorical and Numerical columns
2. We will impute null values of categorical column using Mode of the columns.
3. We will impute numerical columns by mean of the columns, as we know that mean value is sensitive for outliers, we need to first take care of them, then will impute numerical column with their mean value.

Merging test and train dataset

```
df_final= df_train.append(df_test)                    # merging the two dataset

df_final.tail()
# checking tail just to make sure they are merged as test data has no SalePrice column so it shows Nan in place
```

7]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2
287	20	RL	78.0	10206	Pave	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	Norm
288	20	RL	57.0	9245	Pave	IR2	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	Norm
289	20	RL	NaN	11241	Pave	IR1	Lvl	AllPub	CulDSac	Gtl	NAmes	Norm	Norm
290	50	RM	50.0	5000	Pave	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	Feedr	Norm
291	160	RM	21.0	1953	Pave	Reg	Lvl	AllPub	Inside	Gtl	BrDale	Norm	Norm

The final merge dataset name is df_final, all the imputation and encoding is done in this dataset.

First let's check the outliers as we know mean values are sensitive for outliers.

```
#checking for outlier, using IQR

num_data= ['LotFrontage','MasVnrArea', 'GarageYrBlt']
for feature in df_final[num_data]:
    q1 = df_final[feature].quantile(0.25)
    q3 = df_final[feature].quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (IQR*1.5)
    upper_limit = q3 + (IQR*1.5)
    df_final.loc[df_final[feature]<lower_limit,feature] = lower_limit
    df_final.loc[df_final[feature]>upper_limit,feature] = upper_limit
```

As we take care of outliers , now we can impute the numerical and categorical columns.

```
# Imputing Numerical Columns with there mean.

for column in df_final[num_data]:
    df_final[column]=df_final[column].fillna(df_final[column].mean())




# Imputing categorical Columns with their mode.
cat_col= [ 'MasVnrType','BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',
          'BsmtFinType2', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']

for col in df_final[cat_col]:
    df_final[col]= df_final[col].fillna(df_final[col].mode()[0])

#checking for null
df_final[null_column].isnull().sum()
```

We can see that all the null values are imputed, and we are good to proceed further with encoding the dataset.

Encoding :

For encoding the categorical columns, we encode nominal data with Label encoder and Ordinal data using ordinal encoder.

```
# Let's encode nominal data using Label Encoder.
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()

for column in df_final[nominal_data]:
    df_final[column]= encoder.fit_transform(df_final[column])

# checking the data
df_final[nominal_data].head()
```

2]:

HouseStyle	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	Foundation	Heating	CentralAir	Electrical	Functional	GarageType	GarageFinish	I
2	1	1	9	10	2	1	1	1	4	6	1	1	
2	0	5	13	14	2	2	1	1	4	6	1	2	
5	1	1	8	8	2	2	1	1	4	6	1	2	
2	3	1	9	10	1	1	1	1	4	6	1	1	
2	1	1	5	5	3	1	1	1	4	6	1	0	

Our Nominal data has been encoded now let's encode our Ordinal columns

```
# for ordinal data let's use ordinal encoder
from sklearn.preprocessing import OrdinalEncoder
ord_enc= OrdinalEncoder()

for col in df_final[Ordinal_data]:
    df_final[col]=ord_enc.fit_transform(df_final[Ordinal_data])

# checking data
df_final[Ordinal_data].head()
```

3]:

	ExterQual	ExterCond	BsmtQual	BsmtCond	BsmtExposure	HeatingQC	KitchenQual	FireplaceQu	GarageQual	GarageCond
0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
1	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
2	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
3	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
4	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0

Since, the dataset is free from null values and properly encoded, it's time to separate both the dataset again, before going further for checking the skewness and outliers removal.

```
# separating train dataset and test data set

data_train = df_final.iloc[:1168,:]
data_test = df_final.iloc[1168:,:]
```

```
print(data_train.shape)
print(data_test.shape)
```

```
(1168, 71)
(292, 71)
```

```
# we need to drop SalePrice column in data_test
data_test.drop('SalePrice', axis=1, inplace= True)
data_test.shape
```

```
]: (292, 70)
```

Here we drop 71st column in test dataset as it was the SalePrice column and having all null values for test dataset. And It was like this only in original dataset.

We also Observe that column Utilities has only one value through out the column, hence it won't add any value in the Model Building, so we will drop that as well.

After Dropping that column , will check for correlation one more time, as it helps in Understanding the relevant columns for the SalePrice prediction.

```
# Drop Utilities, as it has only one unique value filled in the entire column
data_train.drop('Utilities', axis=1, inplace=True)
data_test.drop('Utilities', axis= 1, inplace= True)

print(data_train.shape)
print(data_test.shape)
```

```
(1168, 70)
(292, 69)
```

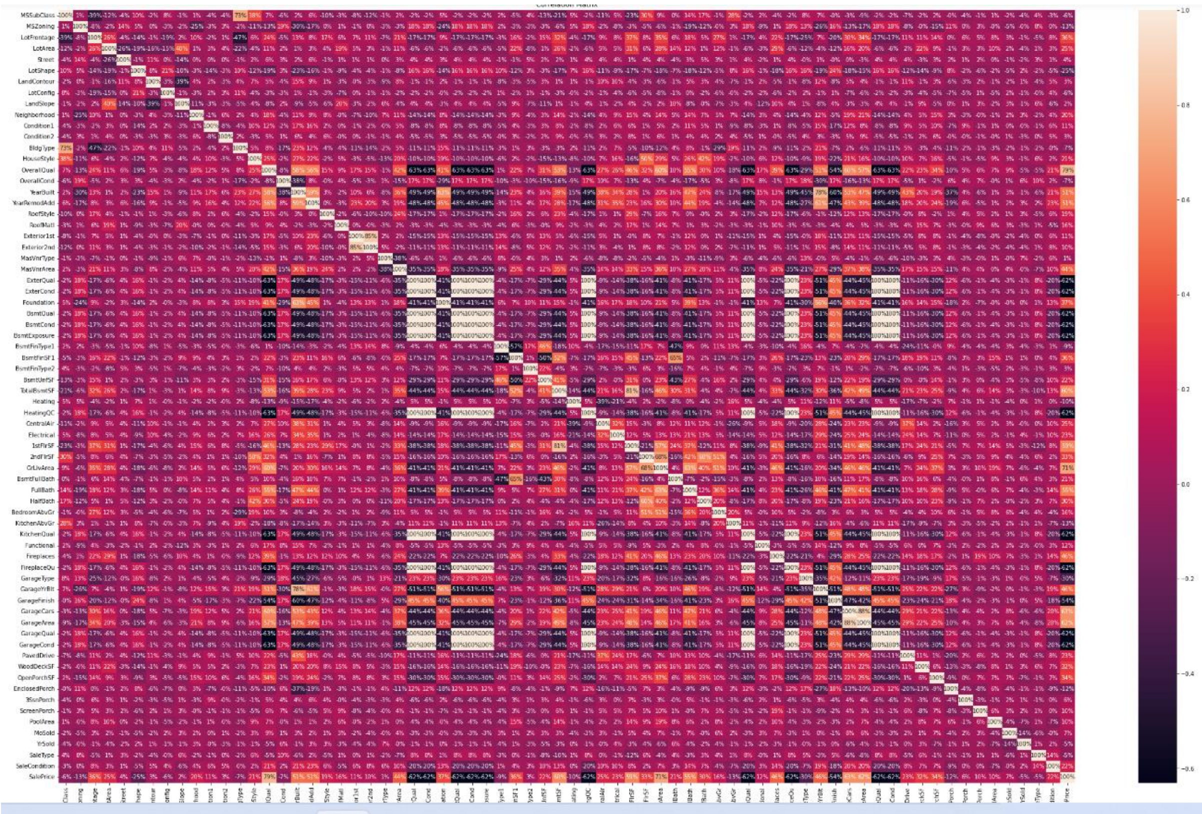
```
data_train.corr()
```

]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Conditior
MSSubClass	1.000000	0.007478	-0.390952	-0.124151	-0.035981	0.104485	-0.021387	0.076880	-0.014930	0.013918	-0.0424
MSZoning	0.007478	1.000000	-0.082734	-0.023328	0.140215	0.053655	0.001175	-0.027246	-0.023952	-0.251833	-0.0256
LotFrontage	-0.390952	-0.082734	1.000000	0.261540	-0.040382	-0.143292	-0.006954	-0.190129	0.017470	0.096343	-0.0225
LotArea	-0.124151	-0.023328	0.261540	1.000000	-0.263973	-0.189201	-0.159038	-0.152063	0.395410	0.010707	0.0295
Street	-0.035981	0.140215	-0.040382	-0.263973	1.000000	-0.012941	0.105226	0.000153	-0.141572	0.001420	0.0021
...	...	...	...	...	...	...	...	...	...	...	
MoSold	-0.016015	-0.051646	0.031347	0.015141	-0.008860	-0.050418	-0.023872	0.019084	0.030526	0.023378	0.0018
YrSold	-0.038595	-0.004964	-0.005859	-0.035399	-0.019635	0.021421	0.009499	-0.009817	-0.005352	0.026181	-0.0041
SaleType	0.035050	0.079854	-0.046102	0.005421	0.025920	-0.015161	-0.041763	-0.002039	0.056004	-0.023081	-0.0071
SaleCondition	-0.028981	0.004501	0.077491	0.034236	0.014176	-0.054905	0.047715	0.043692	-0.061461	0.042340	0.0590
SalePrice	-0.060775	-0.133221	0.364858	0.249499	0.044753	-0.248171	0.032836	-0.060452	0.015485	0.198942	0.1058

70 rows × 70 columns

Let's visualize it, as it gives us clear picture of the correlation.



Observations:

- We can observe that MSSubClass is 0% related to sale price, othe columns which are least related to Sale Price are : LandContour, LandSlope, Condition2, MasVnrType, BsmtFinType2, Street.

- Let's go ahead and drop them as well.

- We can also observe that GarageQual, GarageCond, ExterQual, ExterCond, BsmtQual, BsmtCond, BsmtExposure, HeatingQC, KitchenQual, FirplaceQu are also highly correlated. However these are Categorical columns, so we must run chi squared test to confirm it. We will do it if it is necessary.

- We also able to observe that TotalBsmtSf and 1stFlrSF are also correlated. Similarly Exterior1st and Exterior2nd also show high correlation

We can either drop these categorical columns or will kept it for once, and see if it really affect the over all more, cause deriving multicollinearity connection of categorical columns using Correlation matrix is not really recommended.

Let’s start with dropping those columns which are least below 3% correlated with Label i.e. “SalePrice”

```
data_train.drop(columns=['MSSubClass','LandContour', 'LandSlope', 'Condition2', 'MasVnrType', 'BsmtFinType2', 'Street'],
data_test.drop(columns=['MSSubClass','LandContour', 'LandSlope', 'Condition2', 'MasVnrType', 'BsmtFinType2', 'Street'], a

print(data_train.shape)
print(data_test.shape)

```

```
(1168, 63)
(292, 62)
```

So far our dataset looks good, now Let’s move ahead for skewness and outlier detection and removal, we will perform this step only in Train dataset, as test data is the one where we need to do the prediction of SalePrice, so does not make any sense to follow skewness and outlier steps there.

```
data_train.skew()
```

We will take a threshold value of +/-1 here, and any continuous column with skewness value greater than one will be considered as skewed, and we will remove skewness using Power Transformer.

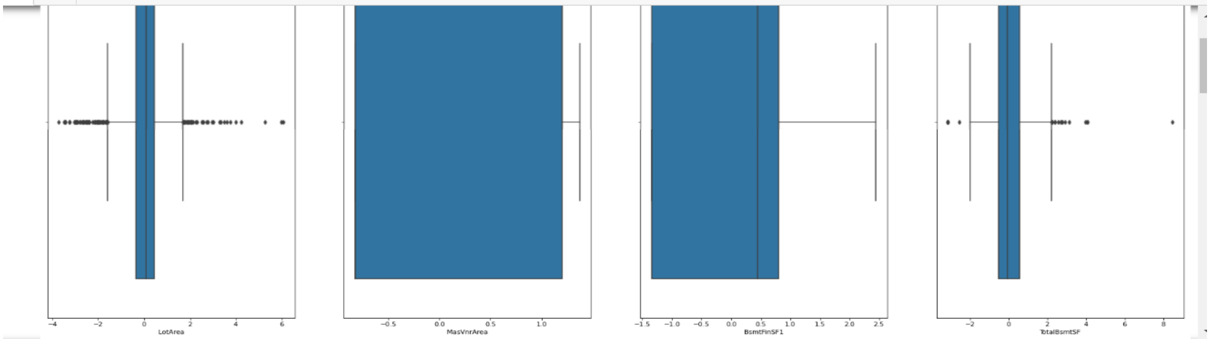
One thing to note here is that, we don’t perform skewness and outlier detection and removal on categorical columns and Label, so here we will leave all the categorical columns as it is, and our Label which although is continuous data but, we don’t transform it as well.

```
# Let's do Power Transform to remove skewness.
from sklearn.preprocessing import PowerTransformer
pwrTrans=PowerTransformer(method='yeo-johnson')
feature_conti=[ 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'TotalBsmtSF',
               '1stFlrSF', 'GrLivArea', 'KitchenAbvGr', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch',
               '3SsnPorch', 'ScreenPorch', 'PoolArea']
data_train[feature_conti]=pwrTrans.fit_transform(data_train[feature_conti].values)
```

We only transform those columns whose skewness level is greater that threshold.

Now let’s visualize for Outlier will remove Outliers again only from continuous data only excluding Label.

```
# let's visualise outliers using boxplot. we are not including SalesPrice here
plt.figure(figsize=(30,60), facecolor='white')
plotnumber=1
for column in data_train[feature_conti]:
    if plotnumber<=13:
        ax=plt.subplot(4,4,plotnumber)
        sns.boxplot(data_train[column])
    plotnumber+=1
plt.show()
```




```
There might be outliers in few of the columns like LotArea, TotalBsmtSF, 1sttFlrSF, GrLivArea. Rest data has some high or
low values which we dont want to iterate, to avoid any major data loss.
```

As we can see the observation, we will only remove outliers from these columns. And for that we will use Z score Method.

```
# for Outliers Dection and removal we will use Z- Score Method
from scipy import stats
feature= ['LotArea', 'TotalBsmtSF', '1stFlrSF', 'GrLivArea']
df_out=pd.DataFrame(data_train[feature])
z= np.abs(stats.zscore(df_out))
threshold = 3
print(np.where(z>3))
```

```
(array([  34,   48,   48,   48,   52,   52,   54,   60,   86,   96,  113,
        119,  124,  137,  141,  159,  174,  226,  231,  243,  245,  249,
        267,  305,  305,  356,  361,  361,  361,  361,  370,  420,  432,
        491,  504,  517,  537,  558,  592,  592,  592,  592,  600,  656,
        679,  689,  691,  698,  706,  735,  760,  831,  834,  865,  884,
        899,  902,  908,  915,  935, 1035, 1038, 1042, 1056, 1067, 1082,
       1107, 1117, 1123, 1126, 1147, 1148, 1164], dtype=int64), array([1, 1, 2, 3, 2, 3, 1, 1, 1, 1, 0, 0, 1, 1, 3, 1, 0, 1,
0, 1, 0, 3,
       1, 1, 2, 0, 0, 1, 2, 3, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 2, 3, 0, 0,
       0, 0, 3, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 1], dtype=int64))
```

```
# Removing outliers from the data frame, and storing final value in df_out Dataframe.
df_out= data_train[(z<3).all(axis=1)]
df_out.shape
```

```
2]: (1105, 63)
```

As now our data is free from skewness and outliers, let's move ahead with further processing.

As we already remove most of the unwanted features from our dataset, let's perform SelectKBest Technique in order to determine most valuable columns in the dataset.

Selecting Best feature by using SelectKBest Feature Selection Method.

SelectKBest use f_classif function to find best feature, where, f_classif uses ANOVA test

```
from sklearn.feature_selection import SelectKBest, f_regression
```

```
# separting features and label
X= data_train.drop('SalePrice', axis=1)
y=data_train.SalePrice
```

```
best_features= SelectKBest(score_func=f_regression, k=60)
fit= best_features.fit(X,y)
df_scores=pd.DataFrame(fit.scores_)
df_columns=pd.DataFrame(X.columns)

#concatenate dataframes
feature_scores=pd.concat([df_columns, df_scores], axis=1)

feature_scores.columns=['Feature_Name', 'Score']  # name output columns

print(feature_scores.nlargest(60, 'Score'))    #print best 50 features
```

Number of Features to pick depends on us, here we try to see first 60 features.

Let's have a look on top few.

Below we can see the feature name and there respective K scores, higher the K score better for the Model Prediction, Our list is already in descending order of K score.

	Feature_Name	Score
9	OverallQual	1925.310146
34	GrLivArea	1087.553977
47	GarageCars	760.625799
18	ExterQual	746.729944
19	ExterCond	746.729944
21	BsmtQual	746.729944
22	BsmtCond	746.729944
23	BsmtExposure	746.729944
29	HeatingQC	746.729944
40	KitchenQual	746.729944
43	FireplaceQu	746.729944
49	GarageQual	746.729944
50	GarageCond	746.729944
48	GarageArea	724.282299
27	TotalBsmtSF	609.160955
32	1stFlrSF	571.388374
36	FullBath	518.998338
46	GarageFinish	472.788235
11	YearBuilt	419.564317
12	YearRemodAdd	405.199690
42	Fireplaces	312.273172
45	GarageYrBlt	308.306592
53	OpenPorchSF	294.576763
20	Foundation	189.817339
17	MasVnrArea	185.507969
2	LotArea	184.722134
1	LotFrontage	179.055902
33	2ndFlrSF	142.869824
52	WoodDeckSF	124.631337
44	GarageType	114.871599
37	HalfBath	111.632993
3	LotShape	76.525762
30	CentralAir	75.597514
31	Electrical	67.924069
51	PavedDrive	66.152095
61	SaleCondition	58.002352
25	BsmtFinSF1	57.255449
26	BsmtUnfSF	56.910469
35	BsmtFullBath	55.372874
8	HouseStyle	51.412890
5	Neighborhood	48.049600
13	RoofStyle	44.944903
54	EnclosedPorch	37.184869
14	RoofMatl	30.580617
38	BedroomAbvGr	29.962224
0	MSZoning	21.067891
41	Functional	16.655755
15	Exterior1st	13.628658
6	Condition1	13.204578
39	KitchenAbvGr	13.054225
57	PoolArea	12.933881
28	Heating	11.782809
16	Exterior2nd	11.547601
24	BsmtFinType1	9.977084
58	MoSold	6.206432
56	ScreenPorch	6.137160
7	BldgType	5.105667
10	OverallCond	5.045938
4	LotConfig	4.276670
55	3SsnPorch	3.682955

We can see all the Important features with there scores, we can select as per our perspective.

Now Let's go ahead and Scale our dataset, before Model Building.

Scaling :

We will do scaling so that our dataset lies in same scale, it helps in better performance of the Algorithm . Here we are using StandardScalar method in order to scale our data, this will do both standardization and normalization of the dataset.

But Scaling can only perform on Features not on label, so we need to first separate Label and features and then will perform scaling on Features data only.

```
from sklearn.preprocessing import StandardScaler

# Scaling the data using StandardScaler.
scalar= StandardScaler()
X_scaled=scalar.fit_transform(X)
```

For Checking Multicollinearity one can perform VIF (Variance Inflation Factor), Now let's move ahead with Model Building.

As its is a regression Model, where we need to predict the SalePrice of the property, There are certain Regression Algorithm which might work for our Dataset. We will start with Linear Regression, then Regularization technique(Lasso) and then Ensemble Technique(randomForest Regressor) let's see which algorithm best suit our dataset.

Model Building:

For Model Building we need to split our train data into train set and test set, using train_test_split. In most of the cases we choose either 20% or 25% for test set and on remaining we will train our model.

```
# import libraries

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error


# Spliting data into train and test, keep 20% for test purpose
X_train,X_test, y_train, y_test= train_test_split(X_scaled, y, test_size=0.20)
```

Here we also import scoring metrics, which is necessary for determining the performance of our model.

Higher the r2 score the better the model, similarly lower the mean absolute error and root mean squared error , the better the model. Let's start applying different algorithm.

Linear Regression

```
from sklearn.linear_model import LinearRegression
# ML Model
LR= LinearRegression()

#fit
LR.fit(X_train,y_train)

#predict
y_pred= LR.predict(X_test)
pred=LR.predict(X_train)

print("--------- Train score---------")
LR_train_MAE= round(mean_absolute_error(y_train, pred), 2)
LR_train_avg_MAE= LR_train_MAE/(max(y)-min(y))
LR_train_R2 = round(r2_score(y_train, pred), 4)
LR_train_RMSE=(np.sqrt(mean_squared_error(y_train, pred))/(max(y)-min(y)))

print(f" R^2 Score : {LR_train_R2}\n")
print(f" MAE score avg : {LR_train_avg_MAE}\n")
print(f" RMSE score avg : {LR_train_RMSE}\n")


#score variables
LR_R2= round(r2_score(y_test, y_pred), 4);
LR_MAE=(mean_absolute_error(y_test, y_pred)/(max(y)-min(y)))
LR_RMSE=(np.sqrt(mean_absolute_error(y_test,y_pred)))/(max(y)-min(y))


print("---------------Test Score-------------")
print(f" R^2 Score : {LR_R2}\n")
print(f" MAE score avg : {LR_MAE}\n")
print(f" RMSE score avg : {LR_RMSE}\n")

```

We determine RMSE using mean squared error, by taking its square root.

Here we are taking the average of both mean absolute error and root mean squared error ny dividing there values divided by total number of values.

Let's see how our scores look like for Linear Regression.

```
--------- Train score---------
 R^2 Score : 0.8376

 MAE score avg : 0.029531134564643798

 RMSE score avg : 0.04563071225432996

---------------Test Score-------------
 R^2 Score : 0.6842

 MAE score avg : 0.032894521868587

 RMSE score avg : 0.00021372994974007832
```

```
# cross validation
from sklearn.model_selection import cross_val_score
LR= LinearRegression()
scores= cross_val_score(LR, X_train, y_train, scoring='r2', cv=10)
LR_CS=scores.mean()
print("Cross validation score is : ", LR_CS)
```

```
Cross validation score is :  0.7447966389837148
```

We did cross validate our score, in order to see if get any improvement.

LASSO (Regularization Technique)

```python
from sklearn.linear_model import Lasso
#model
LS=Lasso(alpha=0.05)
#fit
LS.fit(X_train,y_train)

#predict
y_pred= LS.predict(X_test)
pred=LS.predict(X_train)

print("--------- Train score---------")
LS_train_R2= round(r2_score(y_train, pred), 4)
LS_train_MAE=(mean_absolute_error(y_train,pred))/(max(y)-min(y))
LS_train_RMSE=(np.sqrt(mean_squared_error(y_train,pred)))/(max(y)-min(y))

print(f" R^2 Score : {LS_train_R2}\n")
print(f" MAE avg score : {LS_train_MAE}\n")
print(f" RMSE avg score : {LS_train_RMSE}\n")


#score variables
LS_MAE= (mean_absolute_error(y_test, y_pred))/(max(y)-min(y))   # we are calculating avg MAE
LS_R2= round(r2_score(y_test, y_pred), 4)
LS_RMSE=(np.sqrt(mean_squared_error(y_test,y_pred)))/(max(y)-min(y))
                         # calculating avg RMSE
print("\n------------Test Score--------------\n")
print(f" R^2 Score : {LS_R2}\n")
print(f" Mean Absolute Error avg : {LS_MAE}\n")
print(f" Root Mean Squared Error avg: {LS_RMSE}\n")

```

We import the library necessary for the algorithm, followed by fit the model then prediction and finally scoring. Let's have a look at our result, then will perform cross validation.

```
--------- Train score---------
 R^2 Score : 0.8455

 MAE avg score : 0.02822751518524118

 RMSE avg score : 0.04252410065353866


-------------Test Score---------------

 R^2 Score : 0.6989

 Mean Absolute Error avg : 0.03424357411634604

 Root Mean Squared Error avg: 0.06365088135581802
```

```python
# cross validation
LS= Lasso()
scores= cross_val_score(LS, X_train, y_train, scoring='r2', cv=10)
LS_CS=scores.mean()
print("Cross validation score is : ", LS_CS)
```

```
Cross validation score is :  0.7955948228456167
```

The scores indeed improves from Linear Regression, now let's perform the ensemble technique, as it is most robust model, did not much affected by Multicollinearity.

RandomForestRegressor (Ensemble Technique)

```python
from sklearn.ensemble import RandomForestRegressor

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20, random_state=42)
#Model
RFR=RandomForestRegressor()

#fit
RFR.fit(X_train,y_train)

#predict
y_pred= RFR.predict(X_test)
pred=RFR.predict(X_train)

print("--------- Train score---------")
RFR_train_R2= round(r2_score(y_train, pred), 4)
RFR_train_MAE=(mean_absolute_error(y_train,pred))/(max(y)-min(y))
RFR_train_RMSE=(np.sqrt(mean_squared_error(y_train,pred)))/(max(y)-min(y))

print(f" R^2 Score : {RFR_train_R2}\n")
print(f" MAE avg score : {RFR_train_MAE}\n")
print(f" RMSE avg score : {RFR_train_RMSE}\n")

#score variables
RFR_R2= round(r2_score(y_test, y_pred), 4)
RFR_MAE=(mean_absolute_error(y_test,y_pred))/(max(y)-min(y))
RFR_RMSE=(np.sqrt(mean_squared_error(y_test,y_pred)))/(max(y)-min(y))
print("---------------Test Score-------------")
print(f" R^2 Score : {RFR_R2}\n")
print(f" MAE avg score : {RFR_MAE}\n")
print(f" RMSE avg score : {RFR_RMSE}\n")
```

Let’s have a look at its scores, and cross validation.

```
--------- Train score---------
 R^2 Score : 0.9765

 MAE avg score : 0.009356877658260048

 RMSE avg score : 0.016582062295456418

---------------Test Score-------------
 R^2 Score : 0.7949

 MAE avg score : 0.02703764392884654

 RMSE avg score : 0.05253289193672044
```

```python
# cross validation
RFR= RandomForestRegressor()
scores= cross_val_score(RFR, X_train, y_train, scoring='r2', cv=10)
RFR_CS=scores.mean()
print("Cross validation score is : ", RFR_CS)
```

```
Cross validation score is :  0.8522088460318565
```

We can observe that this model fits best for our dataset, Cross validation even improves the score significantly, let's perform hyper parameter tuning for random forest regressor.

Hyper parameter Tuning for random forest regressor

We will perform RandomizedSearchCV, here as it is fast, and we can rerun it, until desired set of best parameters is obtain.

One downside is, it won't consider all the given parameters at once , it picks certain set of parameter and give us the best result.

```
# Hyper Parameter Tunning

from sklearn.model_selection import RandomizedSearchCV

# We go for randomizedsearchCV as it is fast then GridSearchCv
# Create the random grid

random_grid = {'n_estimators': range(100,1200,100),
               'max_features':['auto', 'sqrt'] ,
               'max_depth': range(5,30,5),
               'min_samples_split': [2, 5, 10, 15, 100],
               'min_samples_leaf':[1, 2, 5, 10] }

#grid_search=GridSearchCV(estimator=RFR, param_grid= random_grid, cv=5)

#grid_search.fit(X_train,y_train)

#grid_search.best_estimator_

rnd_srch=RandomizedSearchCV(RandomForestRegressor(), cv=5, param_distributions= random_grid)

rnd_srch.fit(X_train,y_train)

rnd_srch.best_estimator_
```

```
]:
RandomForestRegressor
RandomForestRegressor(max_depth=20, max_features='auto', min_samples_leaf=2,
                      n_estimators=500)
```

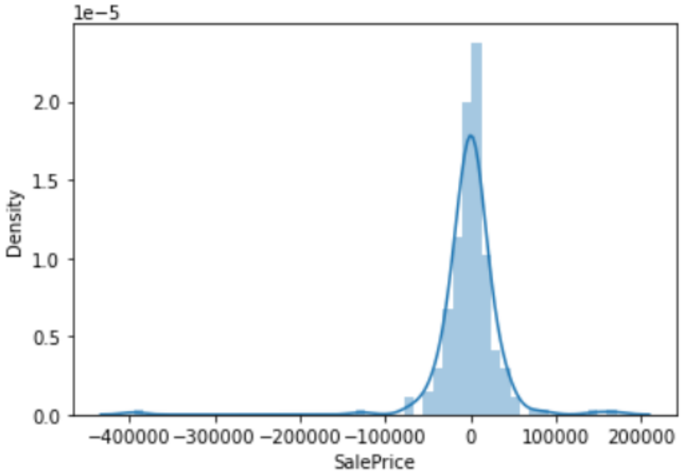
```
# we will use these best parameters in Random forest algorithm and check if accuracy is increasing.

RFR=RandomForestRegressor(max_depth=20,n_estimators=500,
                          min_samples_leaf=2, max_features='auto')
RFR.fit(X_train,y_train)
prediction=RFR.predict(X_test)
print("Post tuning scores")

#score variables
R2= round(r2_score(y_test, prediction), 4)
MAE=(mean_absolute_error(y_test,prediction))/(max(y)-min(y))
RMSE=(np.sqrt(mean_squared_error(y_test,prediction)))/(max(y)-min(y))
print("---------------Test Score-------------")
print(f" R^2 Score : {R2}\n")
print(f" MAE avg score : {MAE}\n")
print(f" RMSE avg score : {RMSE}\n")
```

```
Post tuning scores
---------------Test Score-------------
 R^2 Score : 0.803

 MAE avg score : 0.026914715022637824

 RMSE avg score : 0.05148469382580221
```

We can see our Score improves, r2score can further improve if you run this couple more time, as each time it peaks different set and gives different results.

Let's Visualize how our randomforest model is performing.

```
sns.distplot(y_test-y_pred)
plt.show()
```

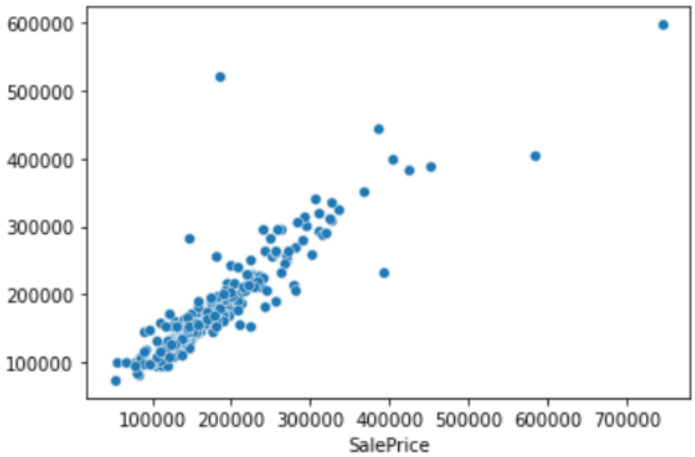


Both prediction and y_test data lies almost on same area.

We can also perform Scatter plot to get a clear idea.

```
sns.scatterplot(y_test,prediction)
```

2]: <AxesSubplot:xlabel='SalePrice'>



From the above we can say both are highly correlated, i.e our prediction lies almost in the same line as of test label.

We can finally say that RandomForest Regressor is our final model. Let's go ahead and save the model.

Saving the best Model.

For saving the model we use pickle technique. We need to save this model at our disk as we need to use this model for Test dataset sale Price prediction.

```
# saving best performing model and saving the model to disk
import pickle
filename= "Housing_Project-Price_Prediction.sav"
pickle.dump(RFR, open(filename, 'wb'))
```

Our model is saved, now let's load this model again, in other to predict the price for test data set.

data_test is our dataset in which we need to do SalePrice Prediction.

Loading the Model for Prediction

```
#load the model from disk.
loaded_model= pickle.load(open(filename,'rb'))

Sale_Price_Pred= loaded_model.predict(data_test)
```

```
Sale_Price_Pred
```

Sale_Price_Pred is our Predicted Sales Price for our test dataset. You can see that values in the Jupyter Notebook.

Conclusions :

As we know that real life dataset some with lots of missing information and are skewed to some degree, so it is very necessary to clean the data, and make it suitable as it can fit for the ML model building Algorithms.

As the number of columns in our dataset are very high, its very important to understand the data very carefully, so to avoid any data or information law, we did drop few of the columns in the process, with legitimate reasons.

One can perform VIF approach in order to avoid multicollinearity, as it sometimes hampers with the results.

One can select even less number of columns in Feature selection and go for Building the model using selected best columns only, this will definitely saves lot of time and energy in return money for the organization.

Overall, Our dataset works best with RandomForrest Regressor Algorithm, we get decent r2 score and even minimum error. And if one can remove multicollinearity than might be our model scores even better.

There are always scopes for improvement in every model, And one can do as much work as possible in finalization.