FLIP ROBO

HOUSING PRICE PREDICTION

Submitted by:

R. Rajesh kannan

ACKNOWLEDGEMENT

Data Sources are from our client US-based housing company named **Surprise Housing** has decided to enter the Australian market. To get profit, the company has collected a data set of the sale of houses in Australia and provide to us to predict the sales value by train the model

Other Resources are Project Use case and Data Description of each variable.

Introduction

Business Problem Framing:

US-based housing company Surprise Housing has decided to enter the Australian real estate market.

To get profit the company uses data analytics to predict and 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

Problem: It is a US based company they don’t know what will be the price of house in respective location with respective area. They need to know which features will affect price of the houses the most.

Conceptual Background of the Domain Problem:

In every real estate concept it is common that Price of the house are mostly depend on

1. Location of house (whether it is high income area or not, it is secured or not)
2. Quality of house(based on materials used)
3. Area of house (size)
4. Extra Features available (pool presence ,car garage)
5. Year of built
6. Total rooms presence

Motivation for the Problem Undertaken:

Aim of this project is to predict the Sales Price of house with the help of given features of house. 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.

Analytical Problem Framing

Mathematical/ Analytical Modeling of the Problem:

Outliers replacing and Skewing the data and modeling done in Exploratory Data Analysis (EDA) process.

Values occur due to mistyping and wrong calculation are consider as outliers it can be replace or removed by ZScore or by IQR. Here I use IQR by replace outlier with maximum and minimum data for the respective input variable.

Skewing is by make mean, median, mode are near as possible. It can be done by Log, Square Root, Cube Root, boxcox and power transformation. Here I used Log, Square Root, Cube Root transformation, boxcox1p.

Data Sources and their formats:

Data Sources: Provided by Surprise Housing about features and details of house.

Formats:

```
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
Id               1168 non-null int64
MSSubClass       1168 non-null int64
MSZoning         1168 non-null object
LotFrontage      954 non-null float64
LotArea          1168 non-null int64
Street           1168 non-null object
Alley            77 non-null object
LotShape         1168 non-null object
LandContour      1168 non-null object
Utilities        1168 non-null object
LotConfig        1168 non-null object
LandSlope        1168 non-null object
Neighborhood     1168 non-null object
Condition1       1168 non-null object
Condition2       1168 non-null object
BldgType         1168 non-null object
HouseStyle       1168 non-null object
OverallQual      1168 non-null int64
OverallCond      1168 non-null int64
YearBuilt        1168 non-null int64
YearRemodAdd     1168 non-null int64
RoofStyle        1168 non-null object
RoofMatl         1168 non-null object
Exterior1st      1168 non-null object
Exterior2nd      1168 non-null object
MasVnrType       1161 non-null object
MasVnrArea       1161 non-null float64
ExterQual        1168 non-null object
ExterCond        1168 non-null object
Foundation       1168 non-null object
BsmtQual         1138 non-null object
BsmtCond         1138 non-null object
BsmtExposure     1137 non-null object
BsmtFinType1     1138 non-null object
BsmtFinSF1       1168 non-null int64
BsmtFinType2     1137 non-null object
BsmtFinSF2       1168 non-null int64
BsmtUnfSF        1168 non-null int64
TotalBsmtSF      1168 non-null int64
Heating          1168 non-null object
HeatingQC        1168 non-null object
CentralAir       1168 non-null object
Electrical       1168 non-null object
1stFlrSF         1168 non-null int64
2ndFlrSF         1168 non-null int64
LowQualFinSF     1168 non-null int64
GrLivArea        1168 non-null int64
BsmtFullBath     1168 non-null int64
BsmtHalfBath     1168 non-null int64
FullBath         1168 non-null int64
HalfBath         1168 non-null int64
BedroomAbvGr     1168 non-null int64
KitchenAbvGr     1168 non-null int64
KitchenQual      1168 non-null object
TotRmsAbvGrd     1168 non-null int64
Functional       1168 non-null object
Fireplaces       1168 non-null int64
FireplaceQu      617 non-null object
GarageType       1104 non-null object
GarageYrBlt      1104 non-null float64
GarageFinish     1104 non-null object
GarageCars       1168 non-null int64
GarageArea       1168 non-null int64
GarageQual       1104 non-null object
GarageCond       1104 non-null object
PavedDrive       1168 non-null object
WoodDeckSF       1168 non-null int64
OpenPorchSF      1168 non-null int64
EnclosedPorch    1168 non-null int64
3SsnPorch        1168 non-null int64
ScreenPorch      1168 non-null int64
PoolArea         1168 non-null int64
PoolQC           7 non-null object
Fence            237 non-null object
MiscFeature      44 non-null object
MiscVal          1168 non-null int64
MoSold           1168 non-null int64
YrSold           1168 non-null int64
SaleType         1168 non-null object
SaleCondition    1168 non-null object
SalePrice        1168 non-null int64
dtypes: float64(3), int64(35), object(43)
```

Necessary:

Other than Id all variables are necessary but we can treat that necessary variable as our requirement. If the data have more than 60% is null then we have to drop that, if there replacement value for null value in description then treat by replacement.

Data Description:

```
In [8]: df1.describe()
Out[8]:
```

	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

```
In [9]: df2.describe()
Out[9]:
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	292.000000	292.000000	247.000000	292.000000	292.000000	292.000000	292.000000	292.000000	291.000000	292.000000	292.000000
mean	755.955479	57.414384	66.425101	10645.143836	6.078767	5.493151	1972.616438	1985.294521	109.171821	439.294521	46.157534
std	442.565228	43.780649	21.726343	13330.669795	1.356147	1.063267	30.447016	20.105792	175.030021	429.559675	152.467119
min	6.000000	20.000000	21.000000	1526.000000	3.000000	3.000000	1872.000000	1950.000000	0.000000	0.000000	0.000000
25%	377.750000	20.000000	53.500000	7200.000000	5.000000	5.000000	1954.000000	1968.000000	0.000000	0.000000	0.000000
50%	778.000000	50.000000	65.000000	9200.000000	6.000000	5.000000	1976.000000	1994.000000	0.000000	369.500000	0.000000
75%	1152.250000	70.000000	79.000000	11658.750000	7.000000	6.000000	2001.000000	2003.250000	180.000000	700.500000	0.000000
max	1456.000000	190.000000	150.000000	215245.000000	10.000000	9.000000	2009.000000	2010.000000	1031.000000	1767.000000	1085.000000

```
In [8]: df1.describe()
Out[8]:
```

BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	Kitch
1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	116
569.721747	1061.095034	1169.860445	348.826199	6.380137	1525.066781	0.425514	0.055651	1.562500	0.388699	2.884418	
449.375525	442.272249	391.161983	439.696370	50.892844	528.042957	0.521615	0.236699	0.551882	0.504929	0.817229	
0.000000	0.000000	334.000000	0.000000	0.000000	334.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
216.000000	799.000000	892.000000	0.000000	0.000000	1143.250000	0.000000	0.000000	1.000000	0.000000	2.000000	
474.000000	1005.500000	1096.500000	0.000000	0.000000	1468.500000	0.000000	0.000000	2.000000	0.000000	3.000000	
816.000000	1291.500000	1392.000000	729.000000	0.000000	1795.000000	1.000000	0.000000	2.000000	1.000000	3.000000	
2336.000000	6110.000000	4692.000000	2065.000000	572.000000	5642.000000	3.000000	2.000000	3.000000	2.000000	8.000000	

```
In [9]: df2.describe()
Out[9]:
```

BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	Kitchen
292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292
557.315068	1042.767123	1133.691781	339.657534	3.702055	1477.051370	0.424658	0.065068	1.575342	0.359589	2.794521	1
411.043768	424.561153	366.941919	424.278825	38.219527	514.199429	0.508831	0.247070	0.547856	0.494795	0.807336	0
0.000000	0.000000	372.000000	0.000000	0.000000	520.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1
255.000000	771.750000	858.000000	0.000000	0.000000	1061.500000	0.000000	0.000000	1.000000	0.000000	2.000000	1
487.000000	971.000000	1047.500000	0.000000	0.000000	1440.000000	0.000000	0.000000	2.000000	0.000000	3.000000	1
780.000000	1322.000000	1370.500000	717.000000	0.000000	1720.250000	1.000000	0.000000	2.000000	1.000000	3.000000	1
1935.000000	3094.000000	2402.000000	1589.000000	479.000000	3447.000000	2.000000	1.000000	3.000000	2.000000	6.000000	3

```
In [8]: df1.describe()
```

Out[8]:

KitchenAbvGr	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	
1168.000000	1168.000000	1168.000000	1104.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	11
1.045377	6.542808	0.617295	1978.193841	1.776541	476.860445	96.206336	46.559932	23.015411	3.639555	15.051370	
0.216292	1.598484	0.650575	24.890704	0.745554	214.466769	126.158988	66.381023	63.191089	29.088867	55.080816	
0.000000	2.000000	0.000000	1900.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1.000000	5.000000	0.000000	1961.000000	1.000000	338.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1.000000	6.000000	1.000000	1980.000000	2.000000	480.000000	0.000000	24.000000	0.000000	0.000000	0.000000	
1.000000	7.000000	1.000000	2002.000000	2.000000	576.000000	171.000000	70.000000	0.000000	0.000000	0.000000	
3.000000	14.000000	3.000000	2010.000000	4.000000	1418.000000	857.000000	547.000000	552.000000	508.000000	480.000000	7

```
In [9]: df2.describe()
```

Out[9]:

KitchenAbvGr	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	Poo
292.00000	292.000000	292.000000	275.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	
1.05137	6.417808	0.595890	1979.760000	1.729452	457.458904	86.397260	47.061644	17.708904	2.489726	15.099315	
0.23616	1.728105	0.621259	23.868875	0.754430	210.785591	121.898836	65.865449	51.892906	30.247488	58.483473	
1.00000	3.000000	0.000000	1916.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1.00000	5.000000	0.000000	1964.000000	1.000000	300.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1.00000	6.000000	1.000000	1979.000000	2.000000	467.500000	0.000000	28.500000	0.000000	0.000000	0.000000	
1.00000	7.000000	1.000000	2003.000000	2.000000	569.750000	149.250000	66.000000	0.000000	0.000000	0.000000	
3.00000	12.000000	2.000000	2010.000000	4.000000	1052.000000	728.000000	418.000000	330.000000	407.000000	396.000000	

```
In [8]: df1.describe()
```

Out[8]:

Cars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SalePrice
000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
541	476.860445	96.206336	46.559932	23.015411	3.639555	15.051370	3.448630	47.315068	6.344178	2007.804795	181477.005993
554	214.466769	126.158988	66.381023	63.191089	29.088867	55.080816	44.896939	543.264432	2.686352	1.329738	79105.586863
000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2006.000000	34900.000000
000	338.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000	2007.000000	130375.000000
000	480.000000	0.000000	24.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.000000	2008.000000	163995.000000
000	576.000000	171.000000	70.000000	0.000000	0.000000	0.000000	0.000000	0.000000	8.000000	2009.000000	215000.000000
000	1418.000000	857.000000	547.000000	552.000000	508.000000	480.000000	738.000000	15500.000000	12.000000	2010.000000	755000.000000

```
In [9]: df2.describe()
```

Out[9]:

rBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	PoolQC	MiscVal	MoSold	YrSold
000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.0	0.0	292.000000	292.000000	292.000000
000	1.729452	457.458904	86.397260	47.061644	17.708904	2.489726	15.099315	0.0	NaN	28.184932	6.232877	2007.859589
875	0.754430	210.785591	121.898836	65.865449	51.892906	30.247488	58.483473	0.0	NaN	224.036218	2.774556	1.322867
000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	NaN	0.000000	1.000000	2006.000000
000	1.000000	300.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	NaN	0.000000	4.000000	2007.000000
000	2.000000	467.500000	0.000000	28.500000	0.000000	0.000000	0.000000	0.0	NaN	0.000000	6.000000	2008.000000
000	2.000000	569.750000	149.250000	66.000000	0.000000	0.000000	0.000000	0.0	NaN	0.000000	8.000000	2009.000000
000	4.000000	1052.000000	728.000000	418.000000	330.000000	407.000000	396.000000	0.0	NaN	3500.000000	12.000000	2010.000000

Data Preprocessing:

There is presence of lot of null values in the given data set.

In [11]:
```
(df1.isnull().sum(),40*"--",df2.isnull().sum())
```

Out[11]:
```
(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
```

In [12]:
```
total=df1.isnull().sum().sort_values(ascending=False)          #percentage of null value in respective columns
percentage=(df1.isnull().sum()/df1.isnull().count()).sort_values(ascending=False)
pd.concat([total,percentage],axis=1,keys=("Total","Percentage"))
```

Out[12]:

	Total	Percentage
PoolQC	1161	0.994007
MiscFeature	1124	0.962329
Alley	1091	0.934075
Fence	931	0.797089
FireplaceQu	551	0.471747
LotFrontage	214	0.183219
GarageType	64	0.054795
GarageCond	64	0.054795
GarageYrBlt	64	0.054795
GarageFinish	64	0.054795
GarageQual	64	0.054795
BsmtExposure	31	0.026541

To treat that null value in some columns there are replacement value for null value mentioned in description then treat it by replacement method.

For remaining columns we have to treat it by apply mean for normal distributed data and median for skewed data and can apply mode for object data.

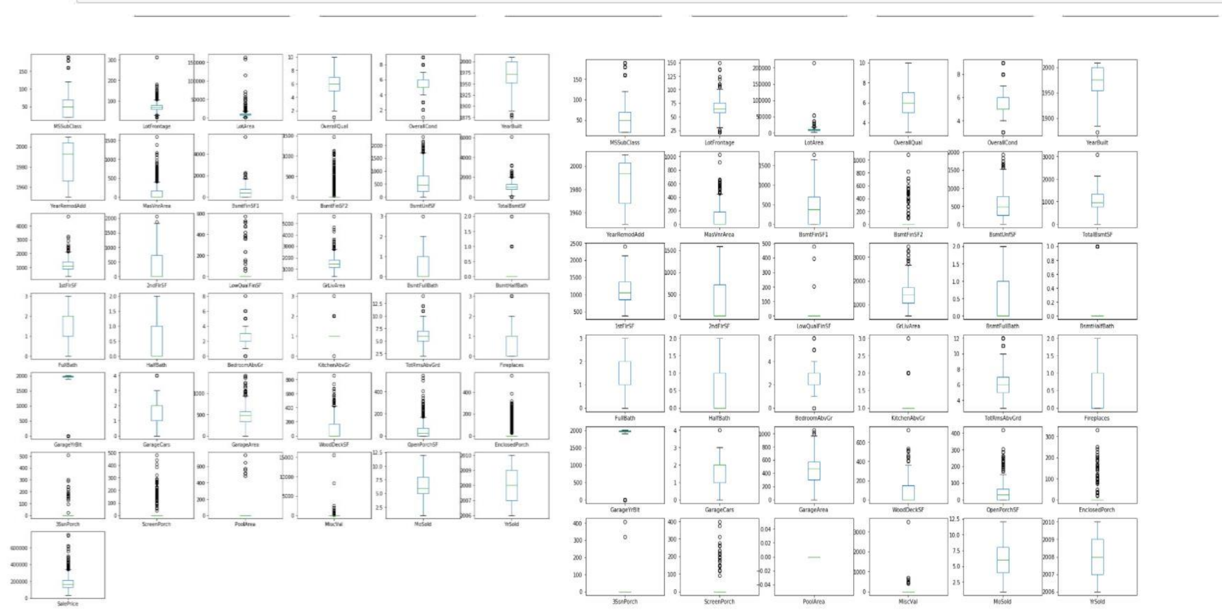
```
In [12]: #Treat by given data description
         for d in data:
             d["Alley"].fillna("No alley access",inplace=True)
             d["BsmtExposure"][d["BsmtExposure"].isnull()&d["BsmtFinType1"].notnull()]="No"
             bsmt=["BsmtExposure","BsmtQual","BsmtCond","BsmtFinType1","BsmtFinType2","FireplaceQu"]
             for b in bsmt:
                 d[b].fillna("NA",inplace=True)
             garage=["GarageType","GarageFinish","GarageQual","GarageCond"]
             for i in garage:
                 d[i].fillna("NA",inplace=True)
             d["GarageYrBlt"].fillna(0,inplace=True)
             d["PoolQC"].fillna("NA",inplace=True)
             d["Fence"].fillna("NA",inplace=True)
             d["MiscFeature"].fillna("NA",inplace=True)
```

```
In [13]: for d in data:
             d["LotFrontage"].fillna(d["LotFrontage"].median(),inplace=True)
```

```
In [14]: df1=df1.dropna()
         df2=df2.dropna()
```

Replace of outliers and skewing of data is done as I mentioned above in Mathematical/ Analytical Modeling of the Problem.

```
In [31]: for d in data:
             d.plot(kind="box",subplots=True,layout=(7,6),figsize=(20,20))
             plt.show()
```



Outliers can be replace by maximum and minimum value

```
In [32]: for d in data:
             for i in d.columns:
                 if d[i].dtypes=="object":                                    #outliers in object data have certain meaning
                     pass
                 elif len(d[i].unique())<20:                                  #data with less unique value may have important data
                     pass
                 else:                                                        #treat data with IQR technique
                     IQR=d[i].quantile(.75)-d[i].quantile(.25)
                     d[i]=d[i][(d[i]>=d[i].quantile(.25)-(IQR*1.5))]
                     d[i].fillna(d[i].min(),inplace=True)                     #convert outliers below whiskers as minimum value
                     d[i]=d[i][(d[i]<=d[i].quantile(.75)+(IQR*1.5))]
                     d[i].fillna(d[i].max(),inplace=True)                     #convert ouliers above whiskers as maximum value
```

Finding skewed

```
In [35]: for d in data:
             d.plot(kind="hist",bins=50,subplots=True,layout=(7,8),figsize=(30,20))
             plt.show()
```

Skewing the data

Apply Log, Square Root, Cube Root and boxcox1p transformation.

```
In [23]: def skews(d,features):                       #define function
             if d[features].dtypes=="object":         #object data will not consider as skewed data
                 pass
             elif len(d[features].unique())<=20:      #data with less unique value may have important data
                 pass
             elif d.skew().loc[features]<.55:         #apply different skewness technique with respect to skewed value
                 pass
             elif len(d[features].unique())<=3:
                 d[features]=d[features]**(1/1.5)
             elif d.skew().loc[features]<1.3:
                 d[features]=np.sqrt(d[features])
             elif d.skew().loc[features]>=3:
                 d[features]=d[features]**(1/4)
             else:
                 d[features]=np.log(d[features])
```

```
In [24]: data=[df,df2]
         for d in data:
             for i in d.columns:
                 skews(d,i)                             #calling function
```

```
In [29]: inte=df1.select_dtypes("int64")
         inte.columns

Out[29]: Index(['MSSubClass', 'OverallQual', 'OverallCond', 'YearRemodAdd',
                'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
                'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', '3SsnPorch',
                'PoolArea', 'MoSold', 'YrSold'],
               dtype='object')
```

```
In [30]: skewes=df1[inte.columns].skew().sort_values(ascending=False)
```

```
In [31]: skews=skewes[abs(skewes)>.75]
         from scipy.special import boxcox1p
         rang=0.15
         for d in data:
             for col in skews.index:
                 d[col]=boxcox1p(d[col],rang)
```

Data Inputs and Output Logic Relationships:

Relationships between target variable and inputs can be find by visualization and correlation of data

Correlation:

If near to 1 is strong +ve correlation

If near to 0 is normal correlation

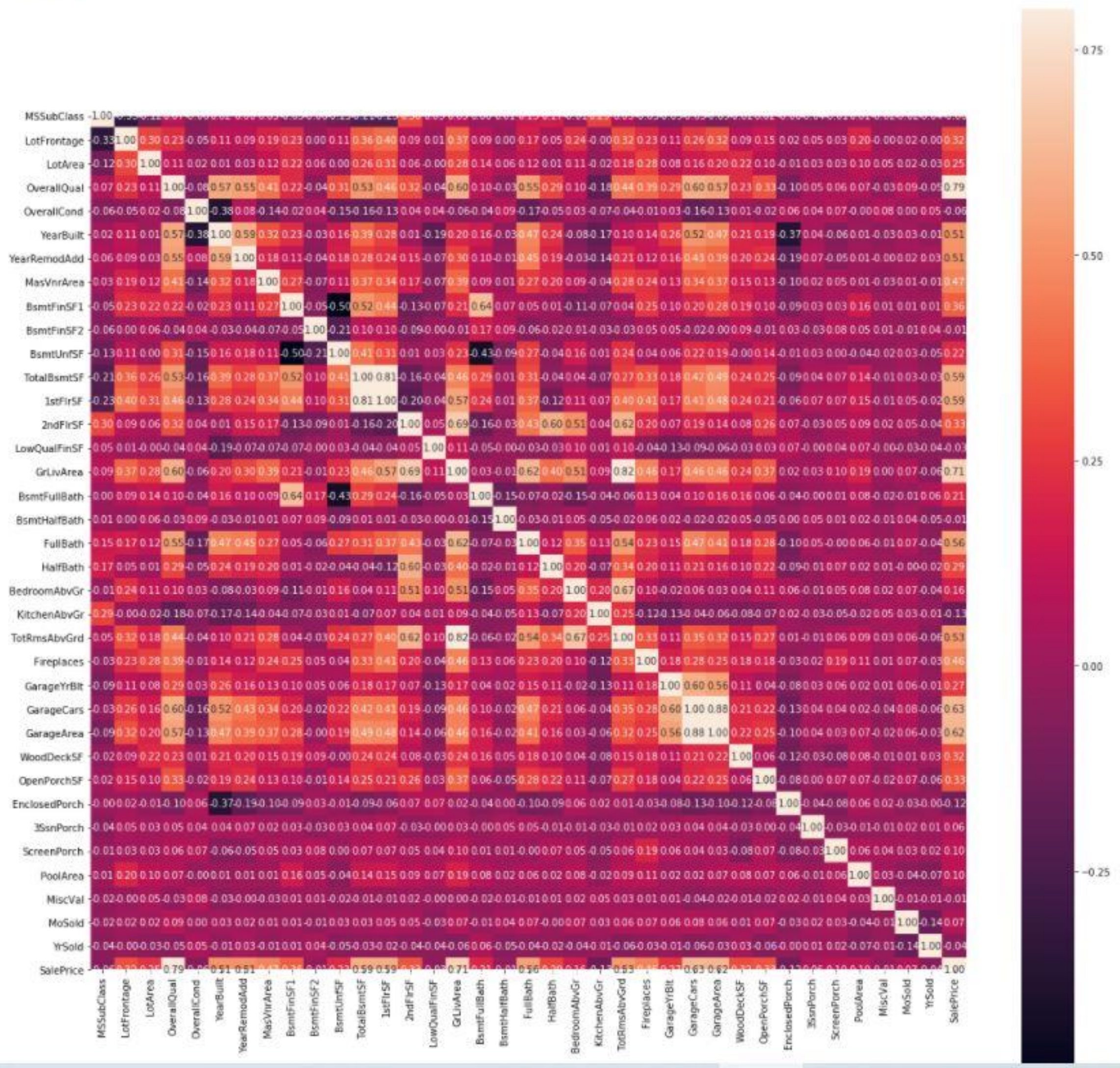
IF near to -1 is strong –ve correlation

Input Variables which is strong correlated with output variable is the necessary variable.

By use of heat map in seaborn library we came to know the correlation of data between each variables.

Also by a code function df.corr()

```
In [27]: plt.figure(figsize=(20,20))
         sns.heatmap(df1.corr(),vmax=.8,fmt='.2f',square=True,annot=True)
         plt.show()
```



```
In [29]: Necessary=df1.corr()["SalePrice"].sort_values(ascending=False)[:11]
         necessary=Necessary.index
         necessary

Out[29]: Index(['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea',
                'TotalBsmtSF', '1stFlrSF', 'FullBath', 'TotRmsAbvGrd', 'YearBuilt',
                'YearRemodAdd'],
               dtype='object')
```

Here OverallQual ,GrLivArea ,GarageCars , GarageArea ,TotalBsmtSF , 1stFlrSF, FullBath ,TotRmsAbvGrd, YearBuilt, YearRemodAdd are highly correlated with SalePrice so these are important variables to predict the SalesPrice.

So these are important variables to predict the SalesPrice

Change in these variables leads to changes in target variable

Visualization techniques like bi-variant visualization can also perform to see relationship between input and output variable.

Assumption (Problem Under Consideration):

Two datasets are being provided to you (test.csv, train.csv). You will train on train.csv dataset and predict on test.csv file

To find

Which variables are important to predict the price variable.

Hardware and Software Requirements and Tools Used:

Hardware: i5 processor, 8 GB RAM.

Software: OS(windows),

Tools: Jupiter Notebook or Py charm

Libraries: numpy, pandas, matplotlib, seaborn, sklearn

Packages: Pyplot, metrics, model_selection, and respective model packages.

Models Development and Evaluation

Identification of Possible Problem solving approaches:

Model Selection: In Superwised there are classification and regression models. But here output variable have many unique values, so it will come under linear regression model.

Testing of identification Approaches:

To find the best random state we have to run for loop by iterating some range of numbers in a model Train Test Split. Random state with high accuracy will be considers as best.

After the selection the best random state we have to find best model by getting good accuracy score, through train and test the data.

Run and Evaluate selected models:

Import the required model through respective packages in sklearn library,

```
In [37]: from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
         from sklearn.linear_model import LinearRegression,ElasticNet,Lasso
         from sklearn.tree import DecisionTreeRegressor
         from sklearn.neighbors import KNeighborsRegressor
         from sklearn.metrics import mean_squared_error,r2_score,mean_absolute_error
         from sklearn.ensemble import RandomForestRegressor,GradientBoostingRegressor
```

Split the data as input train and test, output train and test,

```
In [36]: x=df                        #independent variable
         y=df1["SalePrice"]          #dependent variable
         print(x.shape,y.shape)

         (1161, 79) (1161,)
```

```
In [40]: xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=.20,random_state=40)
```

Firstly have to fit the model,

Secondly have to train the input and output data,

Then have to predict certain test input data,

Finally have to check the accuracy score by compare output test data with predicted output data.

I define 2nd,3rd and final steps in model user define function and metrics in matrix user define function

```
In [38]: def metrics(a,b):                                          #define matrices to check score of model
             print("r2_score : ",r2_score(a,b))
             print("mean_squared_error : ",mean_squared_error(a,b))
             print("mean_absolute_error : ",mean_absolute_error(a,b))
```

```
In [39]: def model(mod,xtrain,ytrain,xtest,ytest):              #define model
             mod.fit(xtrain,ytrain)                             #fitting the trained data of both independent and dependent variable
             pred=mod.predict(xtest)                            #Predict test data
             metrics(ytest,pred)
```

Not only by this process can find the best model because there may occur over fitting due to presence of high bias and high variance in it. So we have to do Cross Validation on it for right result.

```
In [41]: list1=[LinearRegression(),DecisionTreeRegressor(),KNeighborsRegressor(),RandomForestRegressor(),GradientBoostingRegressor(),Elas
         for i in list1:
             print("cross_val_score : ",cross_val_score(i,x,y,cv=4))
             model(i,xtrain,ytrain,xtest,ytest)
             print("_____"*20)
```

```
cross_val_score :  [0.90032073 0.83317856 0.83226973 0.86412679]
r2_score :  0.8839878741363736
mean_squared_error :  549012165.2788715
mean_absolute_error :  17744.438463319006
____________________________________________________________________________________________________
cross_val_score :  [0.79693492 0.7686512  0.75986682 0.76568128]
r2_score :  0.7759760125126558
mean_squared_error :  1060164129.6480687
mean_absolute_error :  23123.87982832618
____________________________________________________________________________________________________
cross_val_score :  [0.55141366 0.56104068 0.59200183 0.44978493]
r2_score :  0.5632926372378125
mean_squared_error :  2066660299.8477254
mean_absolute_error :  33157.789699570814
____________________________________________________________________________________________________
cross_val_score :  [0.89797529 0.86491356 0.84133751 0.86247495]
r2_score :  0.8944616683637607
mean_squared_error :  499446308.2216308
mean_absolute_error :  16741.145922746782
____________________________________________________________________________________________________
cross_val_score :  [0.91562484 0.87957538 0.88288949 0.89319914]
r2_score :  0.9148821320066702
mean_squared_error :  402809143.1224324
mean_absolute_error :  14934.910904697423
____________________________________________________________________________________________________
cross_val_score :  [0.89887033 0.84540122 0.83989162 0.87365231]
r2_score :  0.8904034700774555
mean_squared_error :  518651199.19065326
mean_absolute_error :  16607.022736503353
____________________________________________________________________________________________________
cross_val_score :  [0.90036504 0.83328198 0.8323637  0.86453936]
r2_score :  0.884045764800228
mean_squared_error :  548738205.3072261
mean_absolute_error :  17739.030496475654
____________________________________________________________________________________________________
```

So these models in the list have to iterate through cross validation, model and matrix user define function.

Hyper Parameter Tuning: For to improve the accuracy score for selected best model or all model, we have to apply Grid Search or Randomized search, by apply all the required parameters of respective model inside Grid Search we can get better accuracy.

```
In [42]: def bestmodel(x,y):                                         #Declare Function
             model={"lr":{"model":LinearRegression(),        #insert model and its parameter in model dictionary
                         "para":{"copy_X":["True","False"], "fit_intercept":[True,False],"normalize":[True,False]}
                        },
                  "lasso":{"model":Lasso(),
                           "para":{"alpha":[1,2],"selection":["random","cyclic"]}
                          },
                  "dt":{"model":DecisionTreeRegressor(),
                        "para":{"criterion":["mse","friedman_mse"],"splitter":["best","random"]}
                        },
                  "elsicnet":{"model":ElasticNet(),
                              "para":{"alpha":[1.0,0],"fit_intercept":[True,False],"normalize":[True,False]}
                              }
                 }
             scores=[]
             for keys,values in model.items():           #supply model and its parameter in GridSearchCv for HyperParameter to select be
                 gv=GridSearchCV(values["model"],values["para"],cv=5)
                 gv.fit(x,y)
                 scores.append({
                     "model":keys,
                 "best_scores":gv.best_score_,
                 "best_params":gv.best_params_})
             return pd.DataFrame(scores)
         bestmodel(x,y)                                     # calling function
```

Out[42]:

	model	best_scores	best_params
0	lr	0.855016	{'copy_X': 'True', 'fit_intercept': False, 'no...
1	lasso	0.855320	{'alpha': 2, 'selection': 'cyclic'}
2	dt	0.731657	{'criterion': 'mse', 'splitter': 'best'}
3	elsicnet	0.863926	{'alpha': 1.0, 'fit_intercept': True, 'normali...

- ElasticNet gives a better result at 86.3%
- And also came to know the best parameters

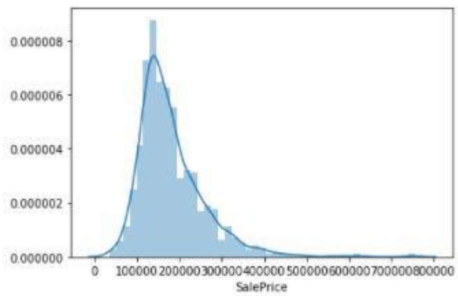
```
In [43]: para={"alpha":[1,2],"selection":["random","cyclic"]}
         gv=GridSearchCV(Lasso(),para,cv=4)
         gv.fit(x,y)                                     #fit the train dataset
         sale=gv.predict(df2)                            #pass test DataSet

In [44]: sale

Out[44]: array([350146.18571679, 252547.93337678, 296034.51425216, 210915.89807329,
                276442.32460026, 119769.34463696, 179099.3419391 , 330101.87068264,
                286683.02109943, 210517.38964645,  99407.27023486, 183998.23953747,
                172667.653587  , 227589.20689719, 330846.54349578, 158710.67423739,
                152882.74983946, 158772.58658622, 224257.37080827, 269708.77194947,
                218190.19006704, 185579.0469997 , 182791.56169194, 110055.41787972,
                138809.11370573, 159024.91625483, 217915.84687151, 187908.37483402,
                211542.79025703, 111299.6028239 , 180258.6979299 , 285216.49105239,
                225832.80444184, 154114.6773902 , 213346.39745976, 241152.67737324,
                155834.52176091, 208593.19466183, 180576.06388131, 124547.44187281,
                327959.20115026, 259519.30621299, 239070.65583324, 180476.46001336,
                179137.21008046, 165790.47380284, 117309.94765142, 251379.09763107,
                353970.7167911 , 184572.44926488, 272289.90006636, 123259.36582269,
                118627.4048069 , 296110.5734023 , 177672.50372698, 194431.02187243,
                234644.04936249, 141331.66078147, 290902.46788507, 139051.56455901,
                233183.12394157, 171102.42521118, 197625.62795083, 256728.92788586,
                125915.12542902, 207336.36772118, 268302.48657029, 185245.82681188,
                210834.48108298, 317992.69744564, 200899.63674211, 218205.45237677,
                205120.35961318, 192676.93159597, 280337.06182281, 339209.2912012 ,
                227916.65566659, 314746.28475156, 183000.39471189, 252875.5764461 ,
                180448.97782997, 190765.89364429, 205874.59341867, 233446.18118831,
                260168.93995694, 127409.87235952, 353100.85098808, 200652.49042565,
                219213.54348032, 292013.30763486, 162205.99098044, 170848.5623859 ,
                152002.4367087 , 241348.47716719, 206284.52206679, 284054.85708831,
                215378.45674925, 355747.68298902, 159925.0952487 , 288147.32186044,
                124555.85224501, 162970.86558482, 237352.6079848 , 183675.30659567,
                313452.62147305, 196610.23043781, 234534.4662244 , 245090.93334338,
                262090.38089428, 219448.21673494, 232094.36712048, 308010.79384417,
                170533.85556786, 139986.54278184, 173193.55807613, 239776.39057112,
                187373.72776174, 132688.13126204, 133031.23380929, 239463.57830282,
                301850.57096718, 169205.16868077, 182497.34946607, 232182.35378739,
                154592.28879404, 235848.21678853, 115418.88301279, 100494.1551439 ,
                193291.33181379, 283254.05238857, 190298.28485976, 198662.19334674,
                229124.12809562, 331376.64740424, 256779.43767442, 168265.61508097,
                315390.11524109, 156251.33265275, 161652.05606914, 391081.53999285,
                 99322.47963242, 371139.88496085, 250263.30436806, 273856.20212706,
                196675.69443403, 176518.76795683, 143710.52732139, 243684.5857751 ,
                209068.429349  , 187315.95466168, 252559.8303928 , 145759.77891242,
                139331.91053273, 223755.60886379, 251616.81075763, 252686.95942175,
                180117.89936093, 219084.34674573, 248628.60308884, 191472.45190125,
                253614.36984432, 132676.57854504, 143084.32956869, 306413.35794568,
                236131.56083519, 247822.1403091 , 167355.45638555, 247996.9037013 ,
                206543.82360504, 157074.48285753, 171418.64508398, 302347.64796987,
                186279.06683232, 384283.12487729, 177831.44613131, 124671.02298318,
                209458.78197226, 229601.62664725, 246164.17482685, 215616.04521275,
                301364.0194998 , 223368.38337388, 408969.09344509, 349690.6767111 4,
```

Visualizations:

Uni variant visualization:

Distribution of SalePrice:

```
In [18]: sns.distplot(df1["SalePrice"])
         plt.show()                                          # distribution of target variable
```

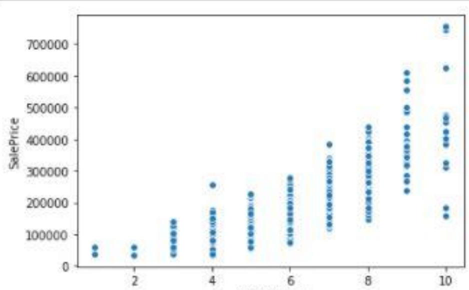


It shows good distribution but slightly skewed.

Bi Variant analysis:

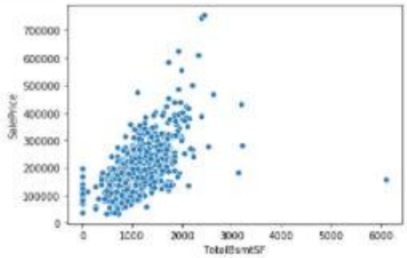
Relation between Output data and quality rate, size of basement, built year

```
In [19]: sns.scatterplot(x="OverallQual",y="SalePrice",data=df1)        #relation between quality of house and sale price
         plt.show()
```



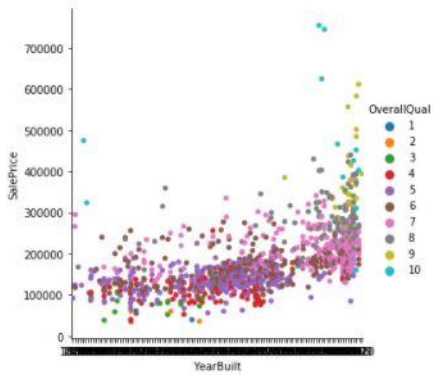
Increase in quality rate of house leads to increase in SalePrice.

```
In [20]: sns.scatterplot(x="TotalBsmtSF",y="SalePrice",data=df1)    #Relation between basement size and SalePrice
         plt.show()
```



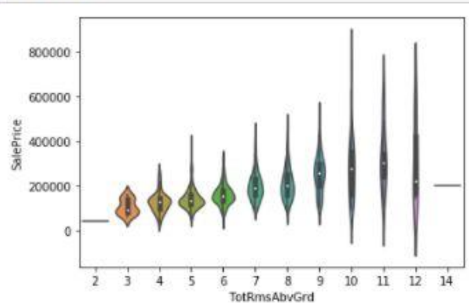
Large size basement leads to high sale price but there is presence of some outliers too.

```
In [21]: sns.catplot(data=df1,x="YearBuilt",y="SalePrice",hue="OverallQual")
         plt.show()
```



Based on built year the price of house slight increases and due to its quality it shows variances

```
In [22]: sns.violinplot(data=df1,x="TotRmsAbvGrd",y="SalePrice")
         plt.show()
```

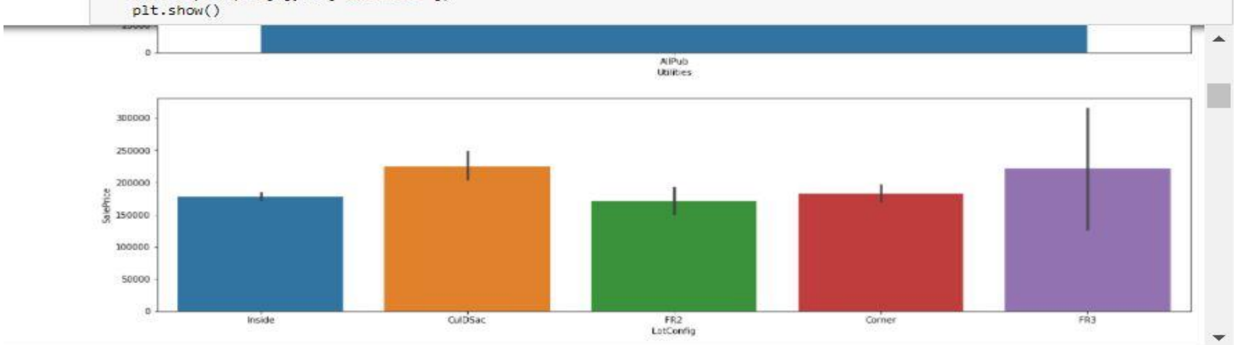


Increase high room number have to pay high may be for following reasons like quality and yearbuilt it get reduced for some house

Bi-Variant Analysis for object data type:

```
In [23]: objects=df1.select_dtypes("object")
         objects.columns

Out[23]: Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
                'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
                'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
                'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
                'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
                'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
                'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
                'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
                'SaleType', 'SaleCondition'],
               dtype='object')
```

```
In [24]: for i in objects.columns:
             plt.figure(figsize=(20,5))
             sns.barplot(df1[i],df1["SalePrice"])
             plt.show()
```



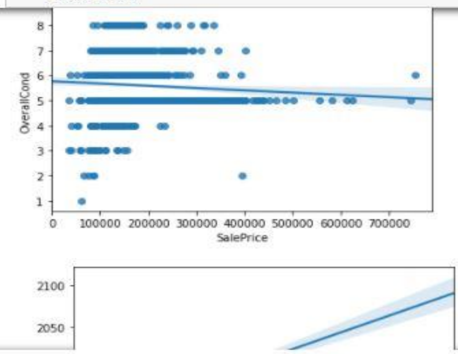
it shows the relation of each object variable with SalePrice.

Here we came to know based on which category in respective features the SalePrice get high or low.

Bi-Variant Analysis for object data type:

```
In [25]: inte=df1.select_dtypes("int64")
         inte.columns

Out[25]: Index(['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
                'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
                '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
                'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
                'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF',
                'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
                'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
               dtype='object')
```

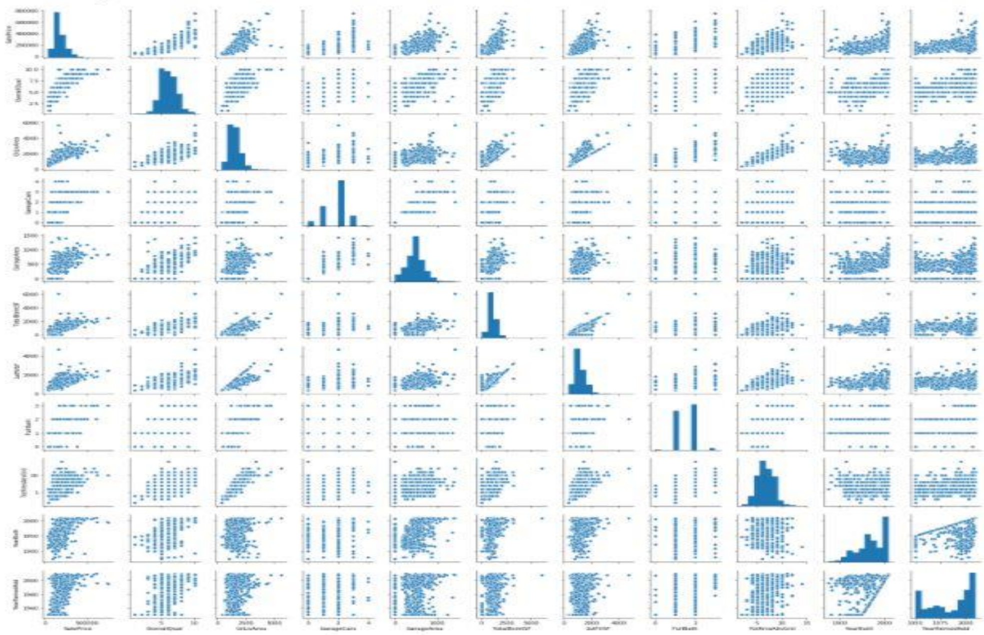
```
In [26]: for i in inte.columns:
             sns.regplot(x="SalePrice",y=i,data=df1)
             plt.show()
```



It shows the relation of each integer variable with SalePrice it shows that there is presence of outliers.

Multi variant analysis on necessary variables:

```
In [30]: sns.pairplot(df1[necessary],size=2)
Out[30]: <seaborn.axisgrid.PairGrid at 0x1f3c0649f88>
```



It shows that the Sale Price is increased based on Raise in Quality, full bath, GarageCars, TotRmsAbvGrd rate increase.

SalePrice is directly Proportional to variables such as GrLivArea,GarageArea, TotalBsmtSF, 1stFlrSF based on it raise and fall in SalePrice.

Based on variables YearBuilt, YearRemodAdd there will be slightly change in SalePrice .

Interpretation of the Results:

Due to presence of negative values and excess of outliers I can’t able to control skewness of data, and also not able to get accurate visualization of data and relationship.

CONCLUSION

Key finding: Prediction of house SalePrice.

Inferences: From the report it concluded that there are some wrong data. By treat it and prediction was lead to get good model.

Observations: Sale Price of House are mostly depend on important variables (OverallQual ,GrLivArea ,GarageCars , GarageArea ,TotalBsmtSF , 1stFlrSF, FullBath ,TotRmsAbvGrd, YearBuilt, YearRemodAdd). Based on it Sales of house get affected.

Learning Outcomes of the study in respect of Data Science

- I learned by visualize also can get important variables and also find how to extract information.
- I used 7 different modes in a loop to get a good predictive model.
- Apply more model on model prediction and did hyper parameter tuning for all the models.

Limitations and Future work:

Limitations: Skew in data can’t control due of large percentage of unique value in some variables.

Steps to follow further:

Here I clean the all the data in one go I have to clean the data separately by consider their variations.

I have to try more important parameter for hyper parameter tuning and also focus on skewing and outliers removal technique.