PROJECT REPORT

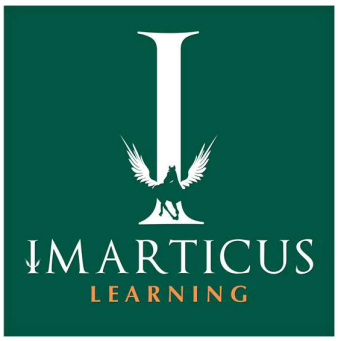
ADVANCED HOME SALES PRICE

Submitted towards the partial fulfillment of the criteria for award of PGA by Imarticus

Submitted By:

Joyson.J(IL028139)

PGA Online 01: October 2021



Abstract

Prices of houses are sophisticatedly linked with the additional luxury added to the house. Despite this, we do not have accurate measures of housing prices based on the vast amount of data available. Housing prices keep changing day in and day out and sometimes are hyped rather than being based on valuation. Predicting housing prices with real factors is the main crux of our research project. Here we aim to make our evaluations based on every basic parameter that is considered while determining the price. We use various regression techniques in this pathway, and our results are not sole determination of one technique rather it is the weighted mean of various techniques to give most accurate results. Therefore, in this project by using machine learning, the selling prices of each house based on many additional factors will be predicted.

Acknowledgements

We are using this opportunity to express my gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have Mr.Thulasidass as our mentor. He has readily shared his immense knowledge in data analytics and guide us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the PGA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date:November-04,2021 Joyson.J

Place: Chennai

Certificate of Completion

I hereby certify that the project titled “Advanced home sales price” was undertaken and completed under supervision by Joyson.J from the batch of PGA (Online-01).

Mentor: Mr.Thulasidass

Date: November 04, 2021

Place – Chennai.

Table of Contents

Chapter 1: INTRODUCTION

1.1 Title & Objective of the study

This dataset titled "Advanced home sales price" is the taken to predict selling price of each house. Here EDA process includes Data analysis, treating missing values, treating outliers, feature selection and building a best model with high accuracy are going to be performed in a perfect way

1.2 Need of the study

Every adult in this world will have dream to buy a house, eventually houses are not easy to buy. It costs lots of money. Sales price of houses are changing with the addition of luxury such as increasing space, more no of rooms, swimming pool, the location of the house and lots and lots of features.so here by having this vast data, it's really tough to decide the price of each houses. Everyone should get a fair price for their house because large amount of money is invested here.

So this study is done to predict the sales price of each house by using a vast amount of data. Many machine learning algorithms such as linear regression, ridge, lasso, random forest, Boosting techniques like Gradient boosting, extreme gradient boosting are used.so lots of

comparisons are performed. Rmse is calculated for every algorithm and finally Gradient boosting is choosed as the good model for this data.

1.3 Data Description

1. SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.
2. MSSubClass: The building class
3. MSZoning: The general zoning classification
4. LotFrontage: Linear feet of street connected to property
5. LotArea: Lot size in square feet
6. Street: Type of road access
7. Alley: Type of alley access
8. LotShape: General shape of property
9. LandContour: Flatness of the property
10. Utilities: Type of utilities available
11. LotConfig: Lot configuration
12. LandSlope: Slope of property
13. Neighborhood: Physical locations within Ames city limits
14. Condition1: Proximity to main road or railroad
15. Condition2: Proximity to main road or railroad (if a second is present)
16. BldgType: Type of dwelling
17. HouseStyle: Style of dwelling
18. OverallQual: Overall material and finish quality
19. OverallCond: Overall condition rating
20. YearBuilt: Original construction date
21. YearRemodAdd: Remodel date
22. RoofStyle: Type of roof
23. RoofMatl: Roof material
24. Exterior1st: Exterior covering on house
25. Exterior2nd: Exterior covering on house (if more than one material)
26. MasVnrType: Masonry veneer type
27. MasVnrArea: Masonry veneer area in square feet
28. ExterQual: Exterior material quality
29. ExterCond: Present condition of the material on the exterior
30. Foundation: Type of foundation
31. BsmtQual: Height of the basement
32. BsmtCond: General condition of the basement
33. BsmtExposure: Walkout or garden level basement walls
34. BsmtFinType1: Quality of basement finished area
35. BsmtFinSF1: Type 1 finished square feet
36. BsmtFinType2: Quality of second finished area (if present)

37. BsmtFinSF2: Type 2 finished square feet
38. BsmtUnfSF: Unfinished square feet of basement area
39. TotalBsmtSF: Total square feet of basement area
40. Heating: Type of heating
41. HeatingQC: Heating quality and condition
42. CentralAir: Central air conditioning
43. Electrical: Electrical system
44. 1stFlrSF: First Floor square feet
45. 2ndFlrSF: Second floor square feet
46. LowQualFinSF: Low quality finished square feet (all floors)
47. GrLivArea: Above grade (ground) living area square feet
48. BsmtFullBath: Basement full bathrooms
49. BsmtHalfBath: Basement half bathrooms
50. FullBath: Full bathrooms above grade
51. HalfBath: Half baths above grade
52. Bedroom: Number of bedrooms above basement level
53. Kitchen: Number of kitchens
54. KitchenQual: Kitchen quality
55. TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
56. Functional: Home functionality rating
57. Fireplaces: Number of fireplaces
58. FireplaceQu: Fireplace quality
59. GarageType: Garage location
60. GarageYrBlt: Year garage was built
61. GarageFinish: Interior finish of the garage
62. GarageCars: Size of garage in car capacity
63. GarageArea: Size of garage in square feet
64. GarageQual: Garage quality
65. GarageCond: Garage condition
66. PavedDrive: Paved driveway
67. WoodDeckSF: Wood deck area in square feet
68. OpenPorchSF: Open porch area in square feet
69. EnclosedPorch: Enclosed porch area in square feet
70. 3SsnPorch: Three season porch area in square feet
71. ScreenPorch: Screen porch area in square feet
72. PoolArea: Pool area in square feet
73. PoolQC: Pool quality
74. Fence: Fence quality
75. MiscFeature: Miscellaneous feature not covered in other categories
76. MiscVal: $Value of miscellaneous feature
77. MoSold: Month Sold
78. YrSold: Year Sold
79. SaleType: Type of sale
80. SaleCondition: Condition of sale

There are totally 81 features present in this data with 23 nominal, 23 ordinal, 14 discrete, and 20 continuous variables. So lots of study and explorations of data is needed to get a perfect model to predict each house price correctly. This data have all problems like missing values, outliers,

multicollinearity. All these problems are treated correctly and a perfect model is created.

1.4 Data Source

- Kaggle

1.5 Tools & Technique

Tools: Jupiter Notebook.

Techniques: Linear Regression, Random Forest, GB & XG Boost.

Chapter 2: Data Exploring & Analysing

2.1 Data view

This process includes importing data with the required libraries and understanding basic properties of data like data shape, data types and data describing.

#Importing the libraries and data.

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

```
df=pd.read_csv("Advanced home prices.csv")
```

```
df
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Gilbert	Norm
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NWAmes	Norm
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Crawfor	Norm
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Edwards	Norm

1460 rows × 81 columns

This dataset have 1460 rows and 81 columns totally…

#Data info

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

This data have 35-integer, 3-float, 43-object data types.

2.2 Data Analysis

This process includes analyzing of all types of variables and missing values. Here just these all missing values in all variables are going to be compared with dependent variable and find how much these missing values and non-missing values contribute in increasing or decreasing the Sales price of house. And all types of variables will be compared with dependent variable to find how much they are related. Even outliers are found for all variables and noted.

a) Analyzing Missing values.
b) Analyzing Numerical variables.
c) Analyzing temporal variables.
d) Analyzing Discrete variables.
e) Analyzing Continuous variables.
f) Analyzing each variables outliers.
g) Analyzing Categorical variables.

*/Analyzing Missing values:

#Only features with null values…

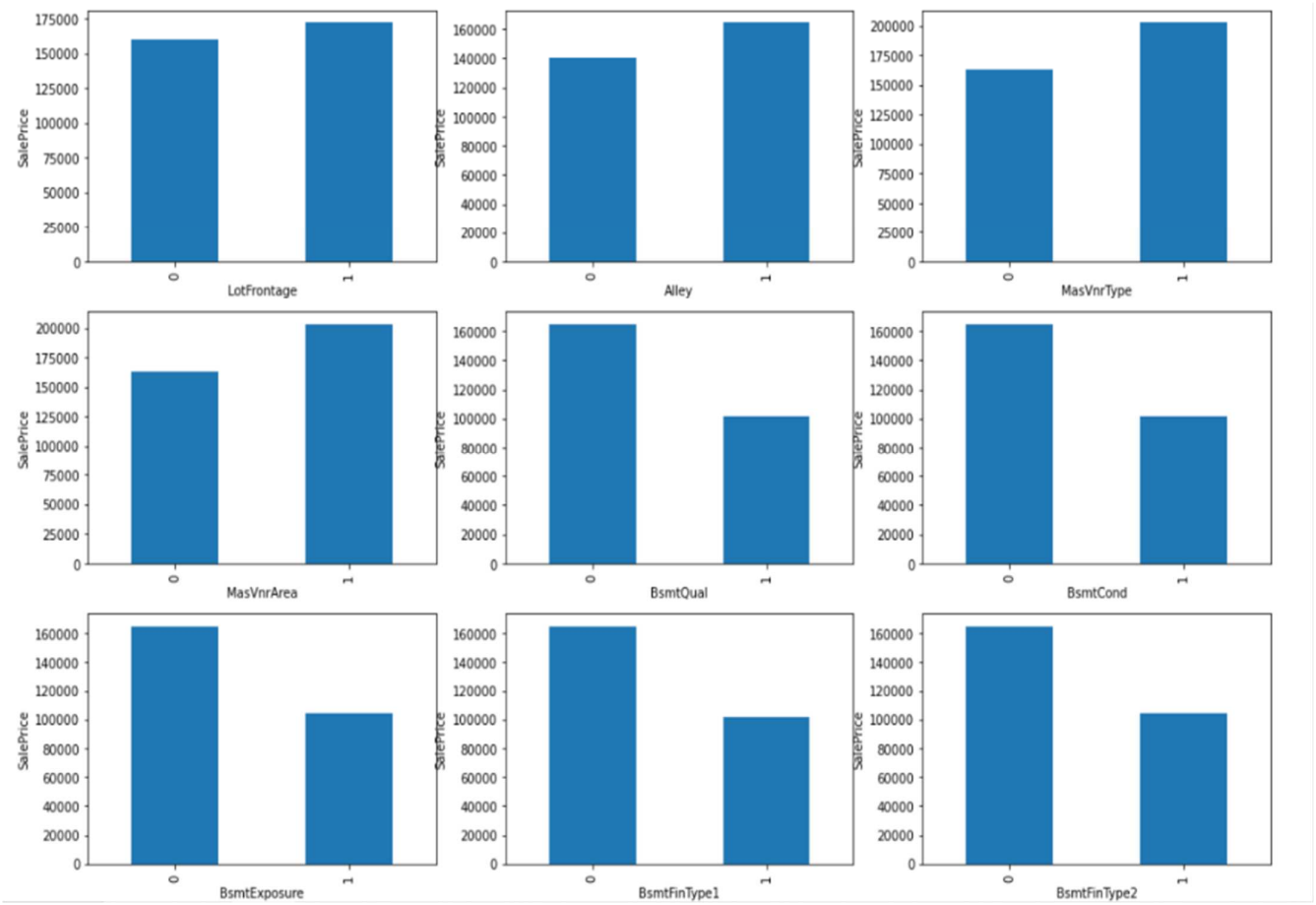
```
features_with_na=[feature for feature in df if df[feature].isnull().sum()]
for feature in features_with_na:
    print(feature,df[feature].isnull().sum())
```

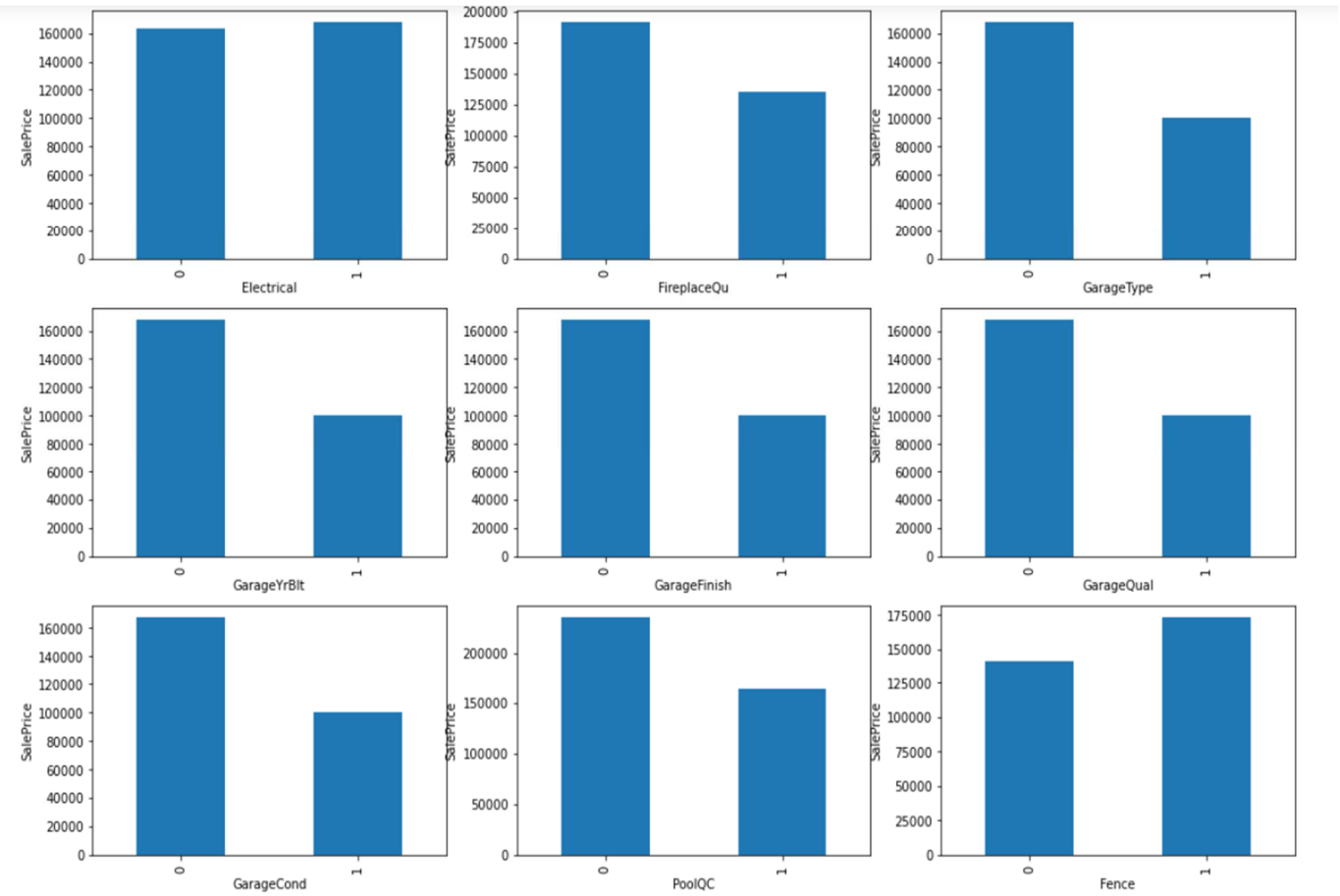
```
LotFrontage 259
Alley 1369
MasVnrType 8
MasVnrArea 8
BsmtQual 37
BsmtCond 37
BsmtExposure 38
BsmtFinType1 37
BsmtFinType2 38
Electrical 1
FireplaceQu 690
GarageType 81
GarageYrBlt 81
GarageFinish 81
GarageQual 81
GarageCond 81
PoolQC 1453
Fence 1179
MiscFeature 1406
```

Here features like Alley, PoolQC, Miscfeture, fence have high missing values above 90%. other features have only low missing values.so from total 81 variables 19 of them have missing values, with the combination of continuous and categorical variables. none of the discrete variables have null values.

#visualizing the comparison of all missing variables with Sales price in barplot using seaborn.

```
plt.figure(figsize=(18,80))
for i,feature in enumerate(features_with_na,1):
    plt.subplot(19,3,i)
    data = df.copy()
    data[feature] = np.where(data[feature].isnull(), 1, 0)
    data.groupby(feature)["SalePrice"].median().plot.bar()
    plt.ylabel("SalePrice")
plt.show()
```





Here the relation between the missing values and the dependent variable is clearly visible. Null values are assigned as 1 and non-null values are assigned as 0.By seeing this graph we could conclude that missing values are important in the features like “alley, Masvnrarea, Masvnrtype, lobfrontage”. So We need to replace these nan values with something meaningful. In remaining features, missing features only contribute less.

*/ Analyzing Numerical features

A numerical variable is a data variable that takes on any value within a finite or infinite interval. numerical variables are splitted into two types, they are Discrete & Continuous variables. But even continuous data with date and time is referred as temporal variables. So First all Temporal variables will be analyzed and then followed by other two.

 # Fetching all numerical variables

```
numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O']
numerical_features
```

```
['Id',
 'MSSubClass',
 'LotFrontage',
 'LotArea',
 'OverallQual',
 'OverallCond',
 'YearBuilt',
 'YearRemodAdd',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
 'KitchenAbvGr',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageYrBlt',
 'GarageCars',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'MiscVal',
 'MoSold',
 'YrSold',
 'SalePrice']
```

This data has totally numerical 38 variables with the combination of continuous, numerical and temporal variables.

*/ Analyzing Temporal variables

Temporal variables are related to time and functions of time. date, day, time, year all features comes under temporal variables.

 # Fetching all temporal variables

```
year_feature = [feature for feature in numerical_features if 'Yr' in feature or 'Year' in feature]
year_feature
```

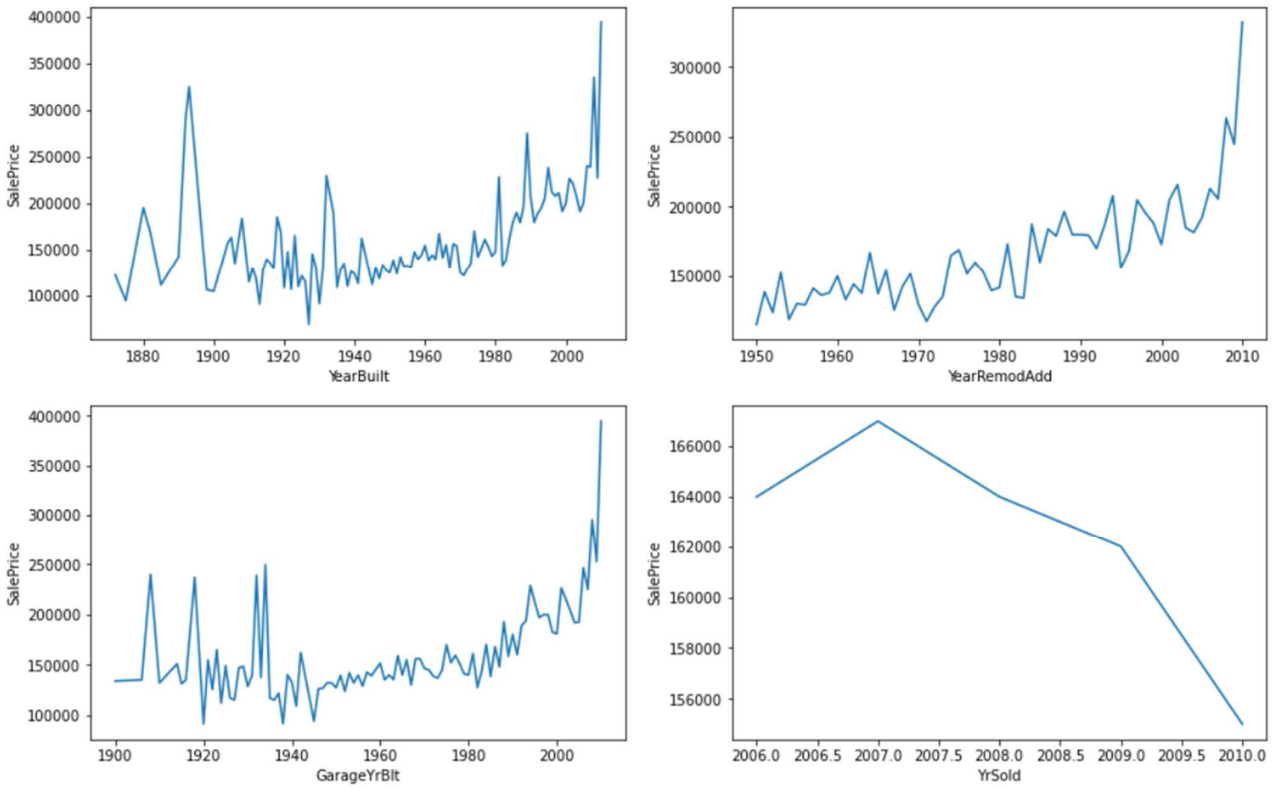
```
['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
```

There are four year features in this dataset. By taking difference b/w year sold and year build, the data will be converted to continuous data which will be good for prediction but this process comes under feature engineering.

Now by checking the relation b/w temporal year features and sales price. we could know, when year increases whether sales price increasing or not.

#visualizing the comparison of temporal features and sales price.

```
plt.figure(figsize=(15,20))
for i,feature in enumerate(year_feature,1):
    plt.subplot(4,2,i)
    data.groupby(feature)["SalePrice"].median().plot()
    plt.xlabel(feature)
    plt.ylabel("SalePrice")
plt.show()
```



By comparing temporal features with *sales* price, when the house is getting old then sales price of the house is decreasing & vice versa. so when year of sold increasing then sales price is decreasing. But other features like year built increase then sales price increase because when the year built is near to year sold then sales price is high but if there is high difference b/w tear built and sold then sales price is decreasing.

*/ Analyzing Discrete variables

Discrete variables are numeric variables that have a countable number of values between any two values. A discrete variable is always numeric.

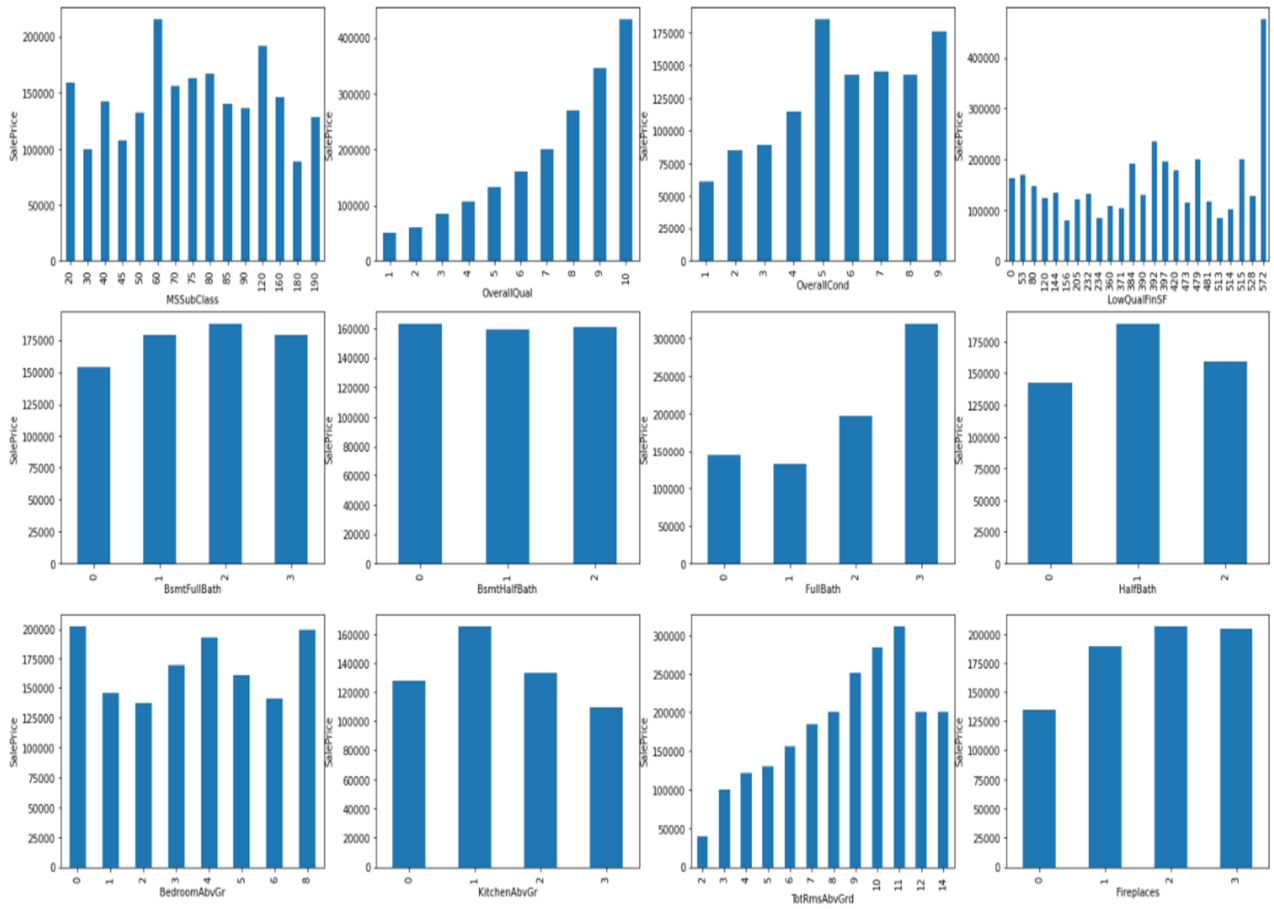
 # Finding only discrete variables from all numerical variables

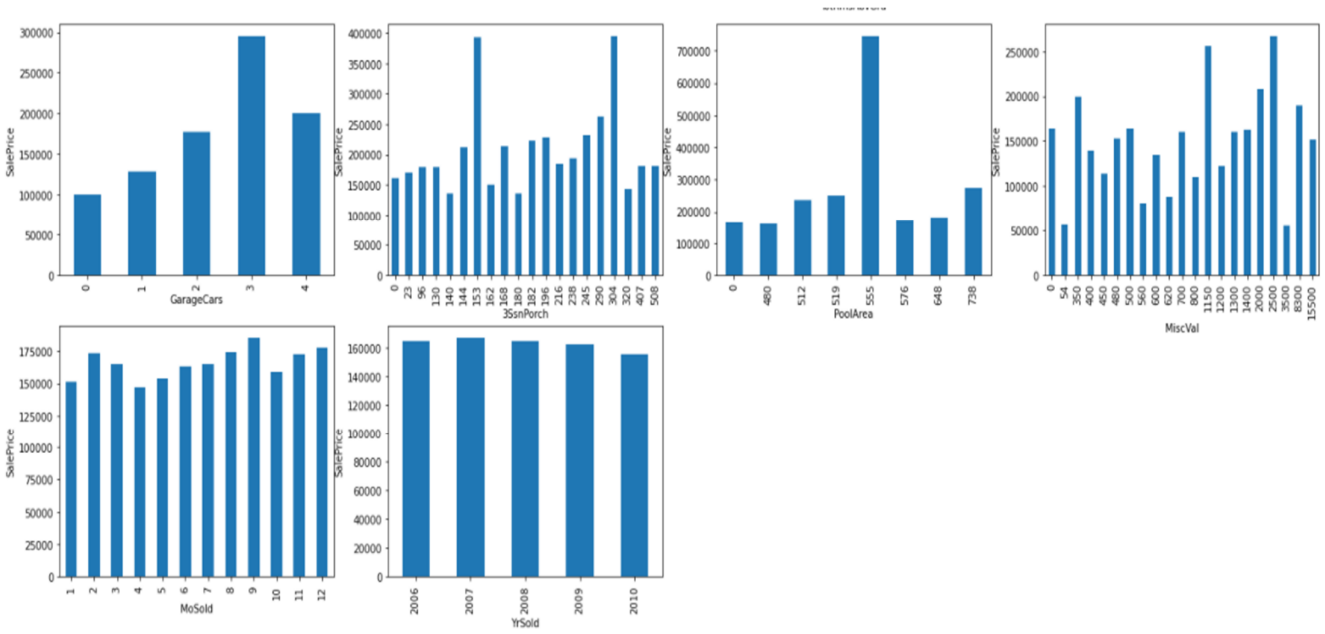
```
discrete_feature=[feature for feature in numerical_features if len(df[feature].unique())<25]
discrete_feature
```

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

These are discrete variables present in all numerical variables. There are totally 18 discrete variables present here.

#visualizing the comparison of discrete variables with sales price in barplot using seaborn.





In this graph we can clearly see the relationship b/w discrete variables and sales price. Many features have good relation with dependent variable but Pool area and LowQualFinSF doesn't seems to have good relation with dependent variable. may be swimming pools are not important for deciding price." OverallQual: Overall material and finish quality" have Exponential rise with sales price and also this feature "TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)" have some exponential curve relation.

*/Analyzing Continuous variable

Continuous variables are numeric variables that have an infinite number of values between any two values. A continuous variable can be numeric or date/time.

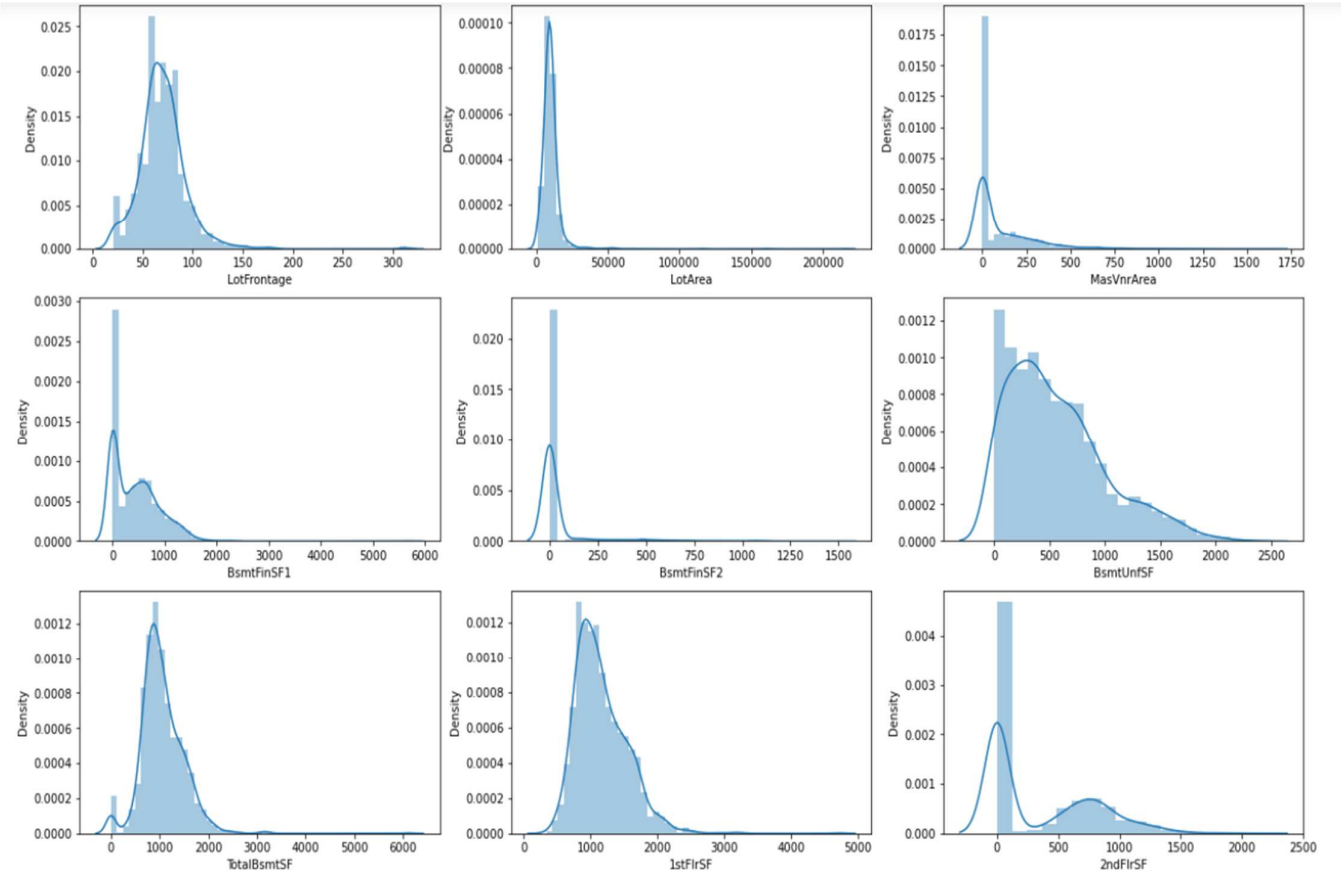
 # fetching continuous variables in total numerical variables.

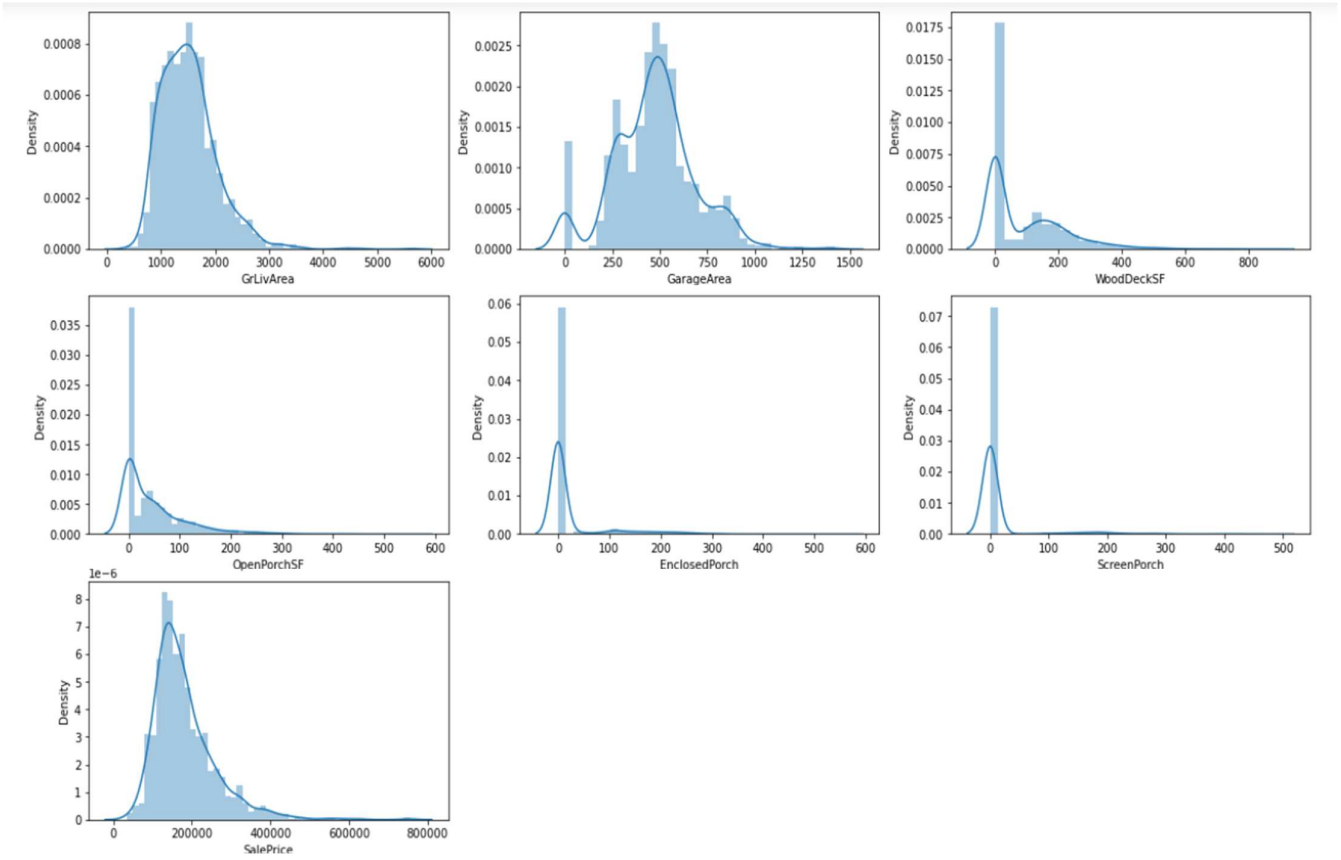
```
continuous_feature=[feature for feature in numerical_features if feature not in discrete_feature+year_feature+['Id']]
continuous_feature
```

```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch',
 'SalePrice']
```

These are continuous variables present in all numerical variables. There are totally 16 continuous variables present here.

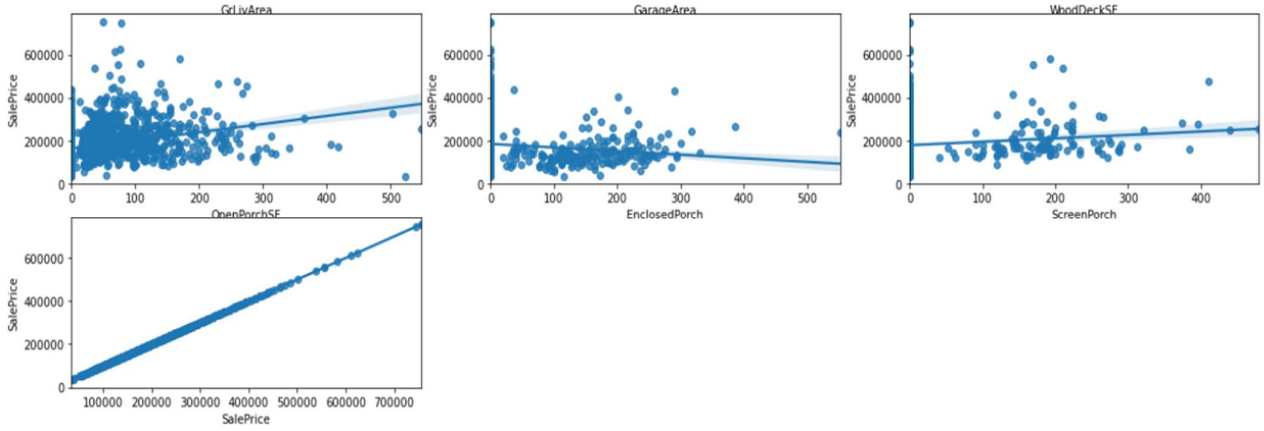
#visualizing the distribution of continuous variables in seaborn distplot.

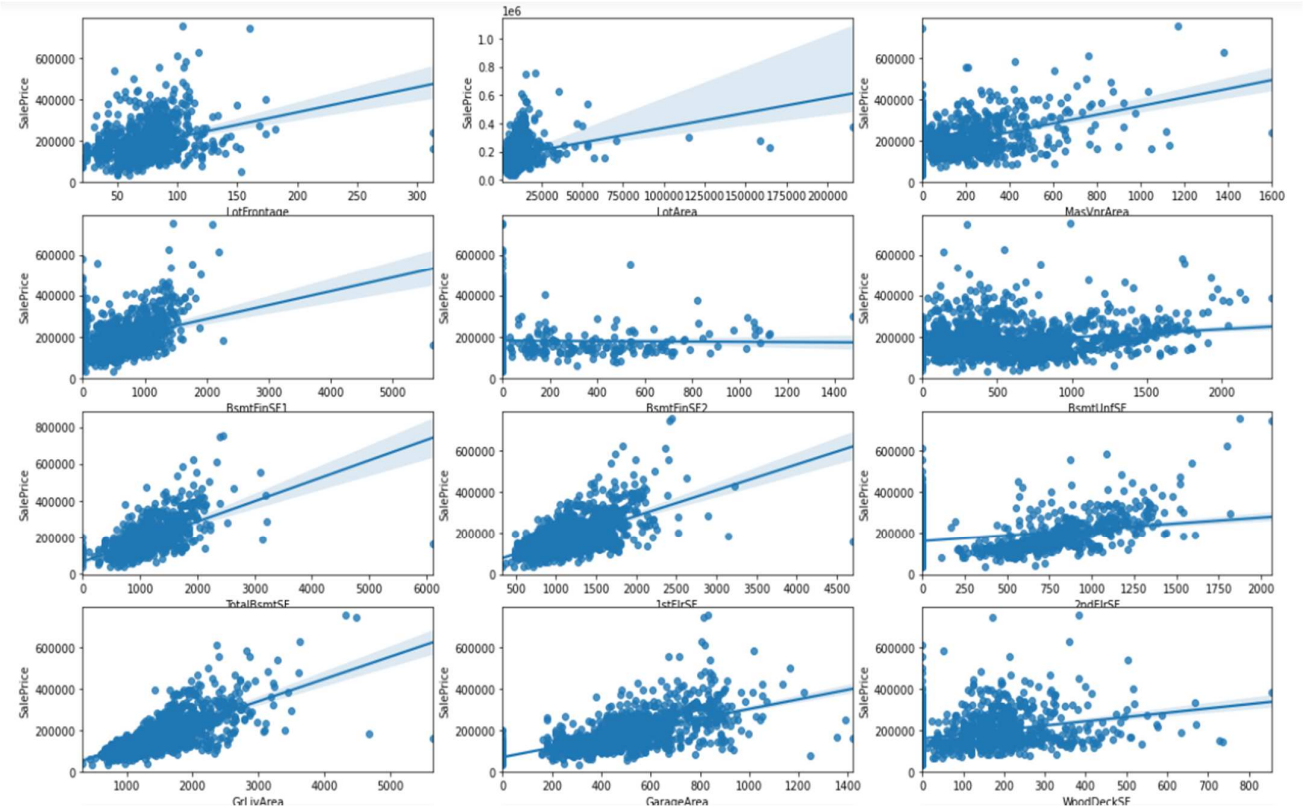




lot of the features are skewed, this might affect the prediction so there is a need of doing feature scaling. Variance of features like Garage area, BsmtUnfSF: Unfinished square feet of basement area are high but features like Screen porch, BsmtUnfS2,lot area, Enclosed porch have low varience.

#visualizing the comparison of continuous variables with sales price in regression plot using seaborn.





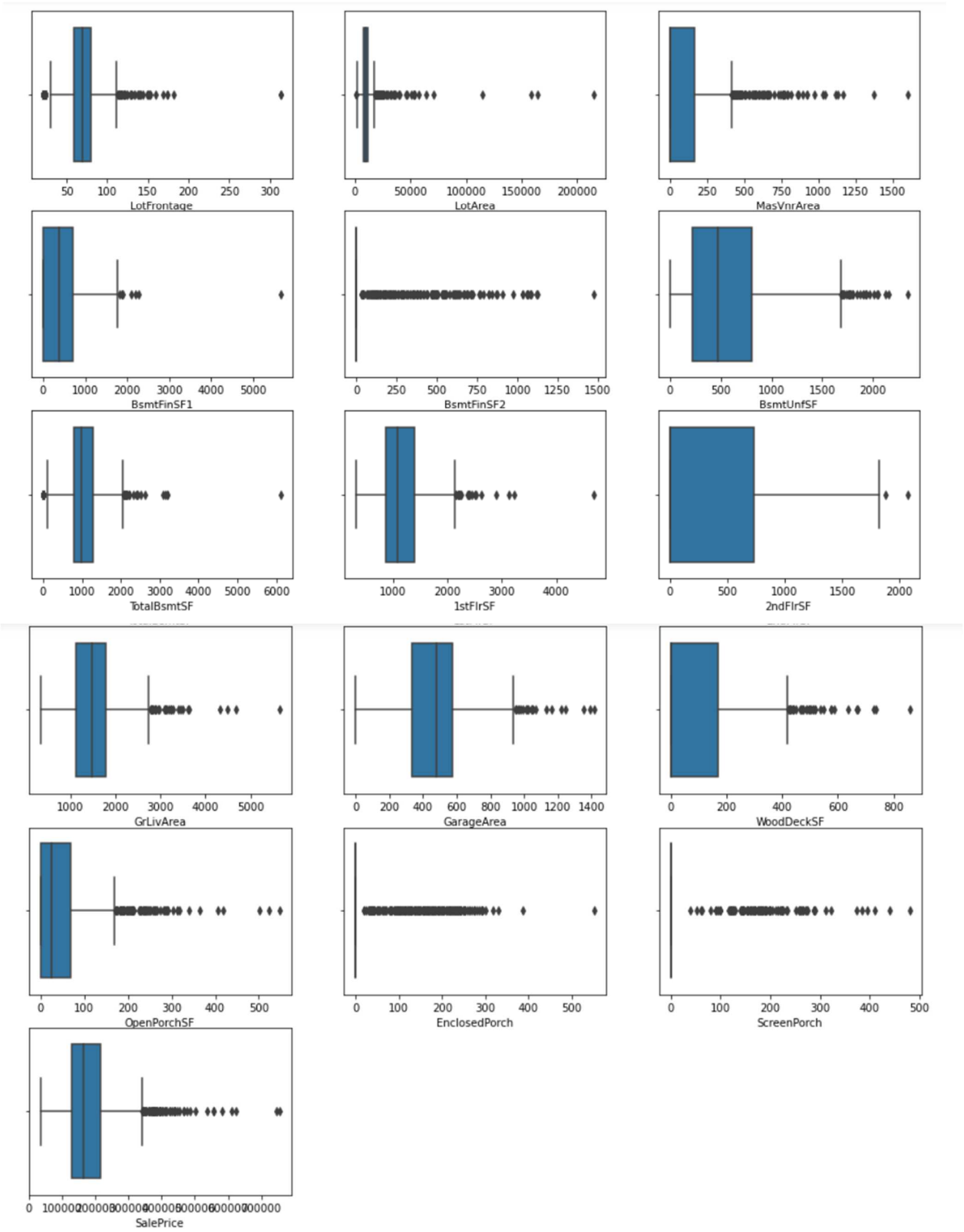
Oh!! It's good to see lots of features have linear relation with Sales price and also some features like BsmtFinSF2: Type 2 finished squarefeet, BsmtUnfSF: Unfinished square feet of basement area, Enclosed porch have low correlation with sales price.

*/ Analyzing Outliers

#Getting all outliers by this code.

```
plt.figure(figsize=(15,55))
for i,feature in enumerate(continuous_feature,1):
    plt.subplot(16,3,i)
    sns.boxplot(df[feature],orient="h")
plt.show()
```

.



This looks like the whole data is filled with outliers. 95% of the features have outliers.

*/Analyzing Catogorical variables

#finding all categorical variables.

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

```
['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',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']
```

There are totally 43 categorical variables present out of 81 variables.so categorical variables are high in this data.

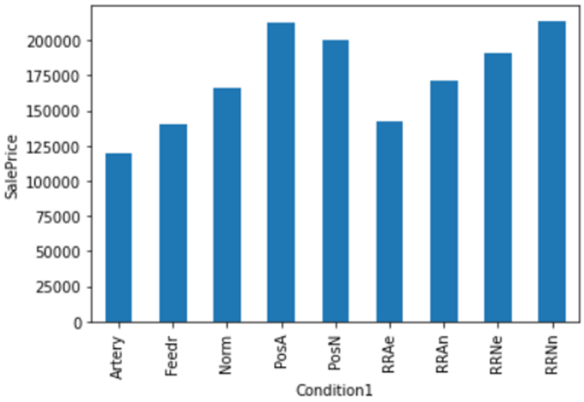
Now we need to see how many categories does each category variable have.To see that below code is executed..

```
for feature in categorical_features:
    print("{} have {} catogories.".format(feature,len(df[feature].unique())))
```

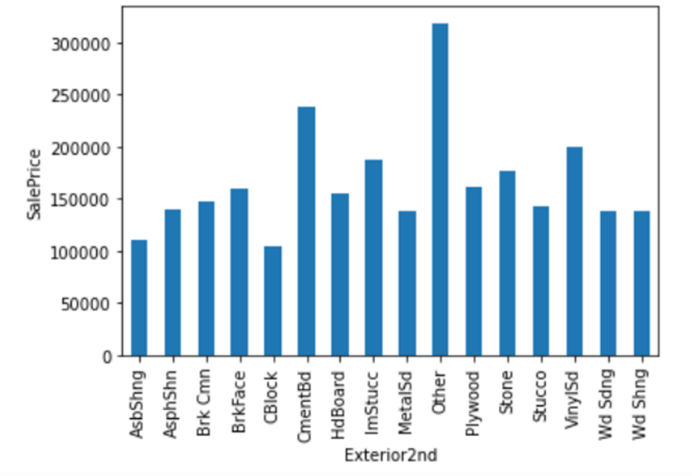
```
MSZoning have 5 catogories.
Street have 2 catogories.
Alley have 3 catogories.
LotShape have 4 catogories.
LandContour have 4 catogories.
Utilities have 2 catogories.
LotConfig have 5 catogories.
LandSlope have 3 catogories.
Neighborhood have 25 catogories.
Condition1 have 9 catogories.
Condition2 have 8 catogories.
BldgType have 5 catogories.
HouseStyle have 8 catogories.
RoofStyle have 6 catogories.
RoofMatl have 8 catogories.
Exterior1st have 15 catogories.
Exterior2nd have 16 catogories.
MasVnrType have 5 catogories.
ExterQual have 4 catogories.
ExterCond have 5 catogories.
Foundation have 6 catogories.
BsmtQual have 5 catogories.
BsmtCond have 5 catogories.
BsmtExposure have 5 catogories.
BsmtFinType1 have 7 catogories.
BsmtFinType2 have 7 catogories.
Heating have 6 catogories.
HeatingQC have 5 catogories.
CentralAir have 2 catogories.
Electrical have 6 catogories.
KitchenQual have 4 catogories.
Functional have 7 catogories.
FireplaceQu have 6 catogories.
GarageType have 7 catogories.
GarageFinish have 4 catogories.
GarageQual have 6 catogories.
GarageCond have 6 catogories.
PavedDrive have 3 catogories.
PoolQC have 4 catogories.
Fence have 5 catogories.
MiscFeature have 5 catogories.
SaleType have 9 catogories.
SaleCondition have 6 catogories.
```

Maximum there are 16 category and minimum 2.Exterior covering of house has more category than the rest of the features.

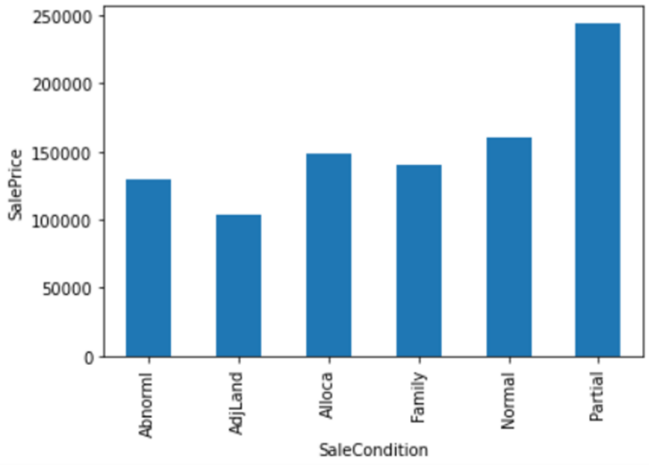
#visualizing the comparison of categorical variables with sales price in bar plot using seaborn.(only some graphs are shown due to large data)



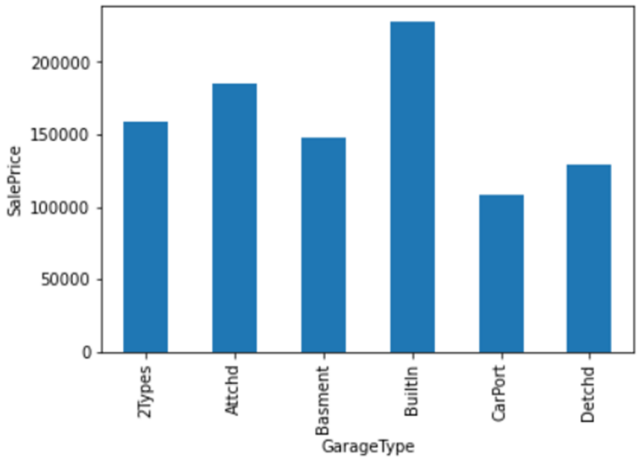
It seems to be condition 1 has some good relation with saleprice. PosA and RRNn have maximum relation.



This is the largest category feature which has only medium relation with sales price. only other category has some relation here. Even by doing all the exterior covering for the house sales price is not increasing.



Even if people are going to stay by family sales price is medium but if it is partial kind of stay then sales price has some sudden increase.



If the garage is in Builtin condition then sales price is increasing but no one is giving much interest if the garage type is Carplot.

2.3 Feature Engineering

The engineering of features is the process of catching what available data is appropriate to use for machine learning algorithms, these engineered features are also responsible for testing the accuracy of models and further improving it.

i. Handling Missing values.
ii. Treating Temporal variables.
iii. Label Encoding.
iv. Feature Scaling.
v. Multi collinearity check.

*/ Handling missing values

 # Now lets check for numerical variables which have missing values and treat it.

```
numerical_nan_values=[feature for feature in numerical_features if df[feature].isnull().sum()>1]
numerical_nan_values
```

```
['LotFrontage', 'MasVnrArea', 'GarageYrBlt']
```

```
df[numerical_nan_values].isnull().sum()
```

```
LotFrontage    259
MasVnrArea       8
GarageYrBlt     81
dtype: int64
```

There are three numerical features which have null values. No discrete features has null values. These null values will be filled with median value of the respective features because this data more outliers so filling with mean or mode will cause some problem .Here outliers are not treated because of large amount of outliers. if we remove that we could lose more data.

```
for feature in numerical_nan_values:
    df[feature]=df[feature].fillna(df[feature].median())
```

```
df[numerical_nan_values].isnull().sum()
```

```
LotFrontage    0
MasVnrArea     0
GarageYrBlt    0
dtype: int64
```

So as mentioned before null values are filled with median value and now no numerical features have null values.

 # Now lets check for categorical variables which have missing values and treat it.

```
categorical_with_nan=[feature for feature in categorical_features if df[feature].isnull().sum()]
categorical_with_nan
```

```
['Alley',
 'MasVnrType',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Electrical',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PoolQC',
 'Fence',
 'MiscFeature']
```

These are the categorical features which have null values. Here all the null values are going to be categorized as a separate category which is denoted by the name “Unknown” and filled. later then it will be converted to integer.

```
for feature in categorical_with_nan:
    df[feature]=df[feature].fillna("Unknown")
```

```
df.head(10)
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Co
0	1	60	RL	65.0	8450	Pave	Unknown	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
1	2	20	RL	80.0	9600	Pave	Unknown	Reg	Lvl	AllPub	FR2	Gtl	Veenker	Feedr	
2	3	60	RL	68.0	11250	Pave	Unknown	IR1	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
3	4	70	RL	60.0	9550	Pave	Unknown	IR1	Lvl	AllPub	Corner	Gtl	Crawfor	Norm	
4	5	60	RL	84.0	14260	Pave	Unknown	IR1	Lvl	AllPub	FR2	Gtl	NoRidge	Norm	
5	6	50	RL	85.0	14115	Pave	Unknown	IR1	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	

*/ Treating temporal variables

In temporal variables we noticed as year of house increase then the price of house is getting reduced. So by taking difference b/w year of sold and year of build we could get a solid data which says how much years the house is away from year sold. we have also year features like the year house was remodified ,year of garage build.so also by taking difference of these with year sold we could get a perfect data.

```
for feature in year_feature:
    if feature!="YrSold":
        df[feature]=df['YrSold']-df[feature]
```

```
df.head(10)
```

OverallCond	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	MasVnrArea
5	5	5	Gable	CompShg	VinylSd	VinylSd	BrkFace	196.0
8	31	31	Gable	CompShg	MetalSd	MetalSd	None	0.0
5	7	6	Gable	CompShg	VinylSd	VinylSd	BrkFace	162.0
5	91	36	Gable	CompShg	Wd Sdng	Wd Shng	None	0.0
5	8	8	Gable	CompShg	VinylSd	VinylSd	BrkFace	350.0
5	16	14	Gable	CompShg	VinylSd	VinylSd	None	0.0
5	3	2	Gable	CompShg	VinylSd	VinylSd	Stone	186.0
6	36	36	Gable	CompShg	HdBoard	HdBoard	Stone	240.0
5	77	58	Gable	CompShg	BrkFace	Wd Shng	None	0.0
6	69	58	Gable	CompShg	MetalSd	MetalSd	None	0.0

#Droping some features

Droping Alley , MiscFeature, PoolQC columns which has 1369 , 1406 & 1453 null values, those are above 90%.

Also dropping id column because it is a unique feature, which dosn't contribute much in prediction.

```
df.drop(["Alley","MiscFeature","PoolQC","Id"],axis=1,inplace=True)
```

final check of null values.

```
for feature in features_with_na:
    if feature!="Alley" and feature!="MiscFeature" and feature!="PoolQC":
        print(feature,df[feature].isnull().sum())
```

```
LotFrontage 0
MasVnrType 0
MasVnrArea 0
BsmtQual 0
BsmtCond 0
BsmtExposure 0
BsmtFinType1 0
BsmtFinType2 0
Electrical 0
FireplaceQu 0
GarageType 0
GarageYrBlt 0
GarageFinish 0
GarageQual 0
GarageCond 0
Fence 0
```

So now there is no null values in the whole data!!!.

*/ LABEL ENCODER

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

 # converting string to integer

Missing values which are filled as Unknown is going to be converted to separate categories in integer format using label encoder.

```
from sklearn.preprocessing import LabelEncoder
LE=LabelEncoder()
```

```
for feature in df.select_dtypes(include='object').columns.values:
    df[feature]=LE.fit_transform(df[feature])
```

```
df.head(10)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2
0	60	3	65.0	8450	1	3	3	0	4	0	5	2	2
1	20	3	80.0	9600	1	3	3	0	2	0	24	1	2
2	60	3	68.0	11250	1	0	3	0	4	0	5	2	2
3	70	3	60.0	9550	1	0	3	0	0	0	6	2	2
4	60	3	84.0	14260	1	0	3	0	2	0	15	2	2
5	50	3	85.0	14115	1	0	3	0	4	0	11	2	2
6	20	3	75.0	10084	1	3	3	0	4	0	21	2	2
7	60	3	69.0	10382	1	0	3	0	0	0	14	4	2
8	50	4	51.0	6120	1	3	3	0	4	0	17	0	2
9	190	3	50.0	7420	1	3	3	0	0	0	3	0	0

*/ Feature Scaling

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.

doing min max scalar

Here we do normalization which converts the infinite range values to (0 to 1).By doing this every individual units for every features will be converted to one unit and machine will able to learn fast.

```
feature_scaling=[feature for feature in df.columns if feature not in ["SalePrice"]]

from sklearn.preprocessing import MinMaxScaler
norm=MinMaxScaler()
df[feature_scaling]=norm.fit_transform(df[feature_scaling])
```

```
df.head(15)
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2
0	0.235294	0.75	0.150685	0.033420	1.0	1.000000	1.0	0.0	1.0	0.0	0.208333	0.250	0.285714
1	0.000000	0.75	0.202055	0.038795	1.0	1.000000	1.0	0.0	0.5	0.0	1.000000	0.125	0.285714
2	0.235294	0.75	0.160959	0.046507	1.0	0.000000	1.0	0.0	1.0	0.0	0.208333	0.250	0.285714
3	0.294118	0.75	0.133562	0.038561	1.0	0.000000	1.0	0.0	0.0	0.0	0.250000	0.250	0.285714
4	0.235294	0.75	0.215753	0.060576	1.0	0.000000	1.0	0.0	0.5	0.0	0.625000	0.250	0.285714
5	0.176471	0.75	0.219178	0.059899	1.0	0.000000	1.0	0.0	1.0	0.0	0.458333	0.250	0.285714

We could see on the above table all the values are converted b/w 0 and 1.

*/ Multicollinearity check

Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model.

If multicollinearity is high for a feature then that feature have some strong relation with some other independent features. we could measure it by variance inflation factor.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor as vifm
def calculate_vif(x):
    vif=pd.DataFrame()
    vif['features']=x.columns
    vif['VIF_Values']=[vifm(x.values,i) for i in range(x.shape[1])]
    return(vif)
```

```
calculate_vif(df).head(50)
```

	features	VIF_Values
0	MSSubClass	9.495413
1	MSZoning	31.566866
2	LotFrontage	10.377407
3	LotArea	3.065363
4	Street	212.562294
5	LotShape	3.641956
6	LandContour	21.194949
7	Utilities	1.099409

Etc...

Here street have some multicollinearity but other features have only low multicollinearity, probably multicollinearity could be fixed in lasso regression.

CHAPTER 3: FITTING MODELS TO DATA

#seperating the data and checking R-sq value.

```
x_ind=df.drop("SalePrice",axis=1) #DROPING THE DEPENDENT VARIABLE
y_dep=df.SalePrice
```

3.1 LINEAR REGRESSION

Linear Regression is a machine learning algorithm based on supervised learning. ... Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x).

Formula: y=mx+c

#checking R-sq value

```
import statsmodels.api as sm
model=sm.OLS(y_dep,x_ind)
my_fit=model.fit()
my_fit.summary()
```

OLS Regression Results

Dep. Variable:	SalePrice	**R-squared (uncentered):**	0.976
Model:	OLS	**Adj. R-squared (uncentered):**	0.974
Method:	Least Squares	**F-statistic:**	754.5
Date:	Fri, 05 Nov 2021	**Prob (F-statistic):**	0.00
Time:	12:37:23	**Log-Likelihood:**	-17159.
No. Observations:	1460	**AIC:**	3.447e+04
Df Residuals:	1386	**BIC:**	3.486e+04
Df Model:	74		
Covariance Type:	nonrobust		

R-sq value is 0.98 which is a good fit. we can go further…

```
#importing train_test split and K-Fold techniques.
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
#assigning 5 folds
kfold_validation=KFold(5)
#importing cross_val_score to perform K-Fold.
from sklearn.model_selection import cross_val_score
```

```
#splitting data for training and testing.
x_train,x_test,y_train,y_test=train_test_split(x_ind,y_dep,train_size=0.8,random_state=1)
```

```
#importing and creating a linear regression model.
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
model=LinearRegression()
```

```
#model fit
model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
#model prediction using test data
y_pred=model.predict(x_test)
```

```
#accuracy of test data
np.round(model.score(x_test,y_test),2)
```

```
0.82
```

```
#accuracy of train data
model.score(x_train,y_train)
```

```
0.84919281579386
```

There is no overfit or underfit in the model.

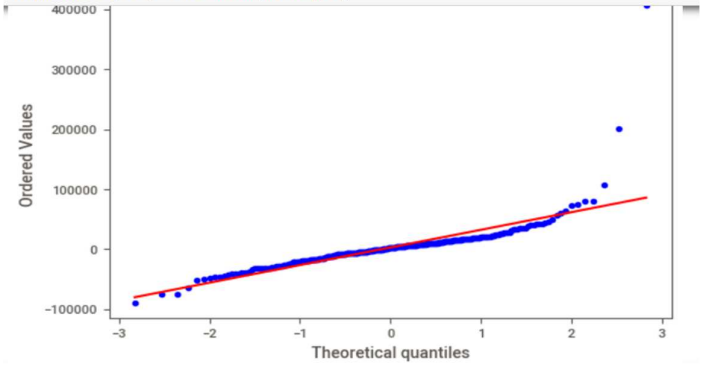
```
#importing libraries to check RMSE
from sklearn.metrics import mean_squared_error,r2_score
```

```
mean_sqr_lin=mean_squared_error(y_test,y_pred)
root_mean_lin=np.sqrt(mean_sqr_lin)
root_mean_lin
```

```
36202.14759282619
```

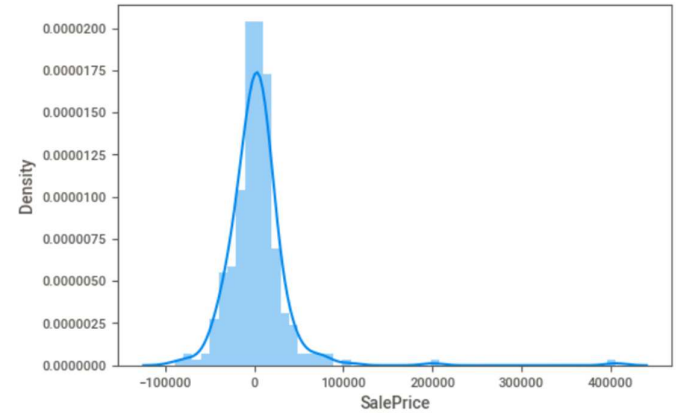
```
#Error
res=y_test-y_pred
```

```
#normality check of error.
import scipy.stats as st
st.probplot(res,dist="norm",plot=plt)
```



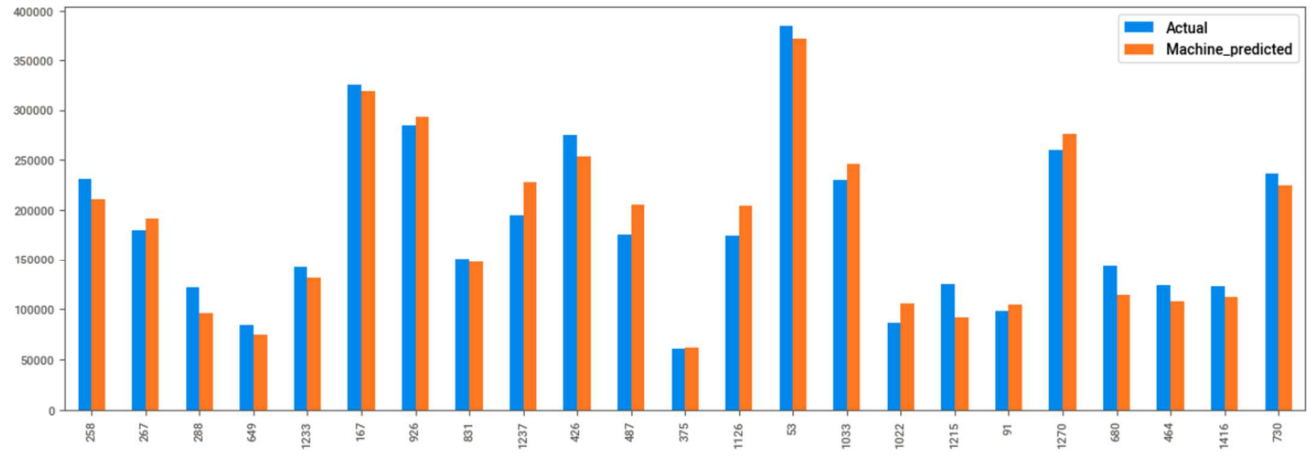

```
#Error distribution.
sns.distplot(res)
```

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



```
#relation b/w actual and predicted in bar graph
f_com=pd.DataFrame({"Actual":y_test,"Machine_predicted":y_pred})
com_g=f_com.head(23)
```

```
com_g.plot(kind="bar",figsize=(15,5));
```



Accuracy: 82% Rmse: 36202

3.2 RIDGE REGRESSION

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization.

```
#importing lasso regression and assigning alpha value as 85 after multiple check with various alpha values.
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha =85, max_iter = 100)
lasso_model.fit(x_train, y_train)
lasso_predict=lasso_model.predict(x_test)
#accuracy of test data
np.round(lasso_model.score(x_test,y_test),2)
```

```
0.84
```

```
#accuracy of train data
lasso_model.score(x_train,y_train)
```

```
0.8421209271658985
```

```
#there is no overfit or underfit in this model
```

```
#performing K-Fold cross validation.
lasso_model_kfold=Lasso(alpha =85, max_iter = 100)
results_lasso=cross_val_score(lasso_model_kfold,x_ind,y_dep,cv=kfold_validation)
lasso_model_kfold.fit(x_train,y_train)
lasso_predict_kfold=lasso_model_kfold.predict(x_test)
print(results_lasso)
print(np.mean(results_lasso))
```

```
[0.88962505 0.81130932 0.84013379 0.84290232 0.70076656]
0.8169474080284308
```

```
#Rmse using train test split model.
mean_sqr_lasso=mean_squared_error(y_test,lasso_predict)
root_mean_lasso=np.sqrt(mean_sqr_lasso)
root_mean_lasso
```

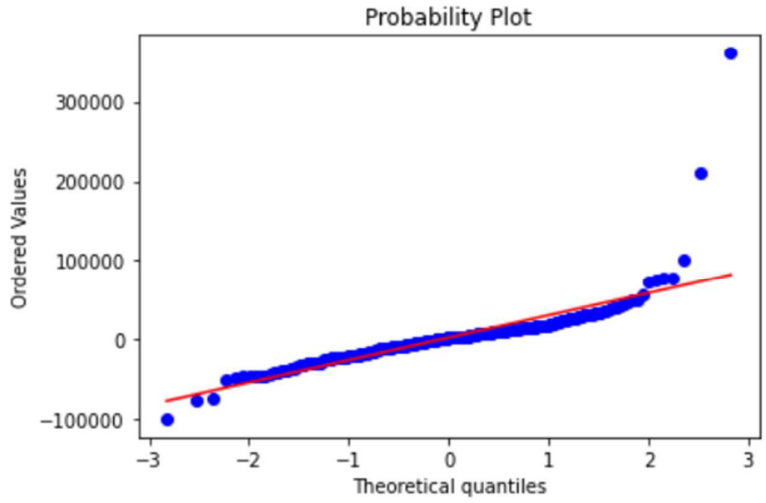
```
34108.95863807599
```

```
#Rmse using k-fold model.
mean_sqr_lasso_kfold=mean_squared_error(y_test,lasso_predict_kfold)
root_mean_lasso_kfold=np.sqrt(mean_sqr_lasso_kfold)
root_mean_lasso_kfold
```

```
34108.95863807599
```

```
#normality check
st.probplot(y_test-lasso_predict,dist="norm",plot=plt)
```

```
          7.87237334e+04,  1.01389028e+05,  2.09708547e+05,  3.61486692e+05])),
(28121.38748553545, 2483.3788614083996, 0.819971821450967))
```



Accuracy: Min acc: 69% , Max acc:81% , Mean acc:75% , RMSE: 42284

3.3 LASSO REGRESSION

Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean.

```
#importing lasso regression and assigning alpha value as 85 after multiple check with various alpha values.
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha =85, max_iter = 100)
lasso_model.fit(x_train, y_train)
lasso_predict=lasso_model.predict(x_test)
#accuracy of test data
np.round(lasso_model.score(x_test,y_test),2)
```

```
0.84
```

```
#accuracy of train data
lasso_model.score(x_train,y_train)
```

```
0.8421209271658985
```

```
#there is no overfit or underfit in this model
```

```
#performing K-Fold cross validation.
lasso_model_kfold=Lasso(alpha =85, max_iter = 100)
results_lasso=cross_val_score(lasso_model_kfold,x_ind,y_dep,cv=kfold_validation)
lasso_model_kfold.fit(x_train,y_train)
lasso_predict_kfold=lasso_model_kfold.predict(x_test)
print(results_lasso)
print(np.mean(results_lasso))
```

```
[0.88962505 0.81130932 0.84013379 0.84290232 0.70076656]
0.8169474080284308
```

```
#Rmse using train test split model.
mean_sqr_lasso=mean_squared_error(y_test,lasso_predict)
root_mean_lasso=np.sqrt(mean_sqr_lasso)
root_mean_lasso
```

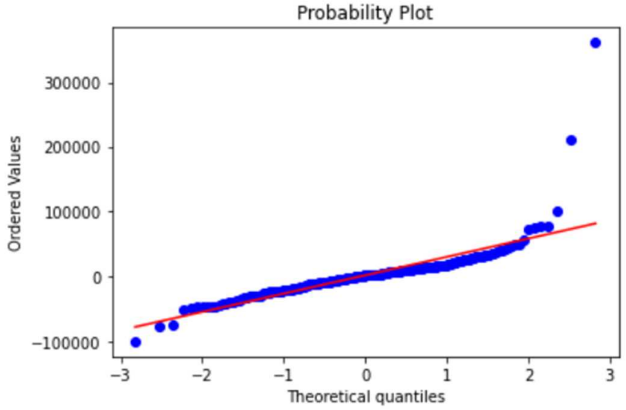
```
34108.95863807599
```

```
#Rmse using k-fold model.
mean_sqr_lasso_kfold=mean_squared_error(y_test,lasso_predict_kfold)
root_mean_lasso_kfold=np.sqrt(mean_sqr_lasso_kfold)
root_mean_lasso_kfold
```

```
34108.95863807599
```

```
#normality check
st.probplot(y_test-lasso_predict,dist="norm",plot=plt)
```

```
          7.87237334e+04,  1.01389028e+05,  2.09708547e+05,  3.61486692e+05])),
 (28121.38748553545, 2483.3788614083996, 0.819971821450967))
```



Accuracy: Min acc: 70% , Max acc:88% , Mean acc:82% , RMSE: 34108

3.4 SGD REGRESSION

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions.

```
#performing SGD regression
from sklearn.linear_model import SGDRegressor
model_sgd=SGDRegressor()
model_sgd.fit(x_train,y_train)
sgd_predict=model_sgd.predict(x_test)
#accuracy of test data
np.round(model_sgd.score(x_test,y_test),2)
```

```
0.83
```

```
#accuracy of train data
model_sgd.score(x_train,y_train)
```

```
0.8335221168433687
```

```
#there is no overfit or underfit in this model
```

```
#k-fold on SGD model
model_sgd_kfold=SGDRegressor()
results_sgd=cross_val_score(model_sgd_kfold,x_ind,y_dep,cv=kfold_validation)
model_sgd_kfold.fit(x_train,y_train)
sgd_predict_kfold=model_sgd_kfold.predict(x_test)
print(results_sgd)
print(np.mean(results_sgd))
```

```
[0.88270118 0.81429689 0.83228672 0.84057597 0.72753109]
0.8194783713063927
```

```
#Rmse using train test split model.
mean_sqr_sgd=mean_squared_error(y_test,sgd_predict)
root_mean_sgd=np.sqrt(mean_sqr_sgd)
root_mean_sgd
```

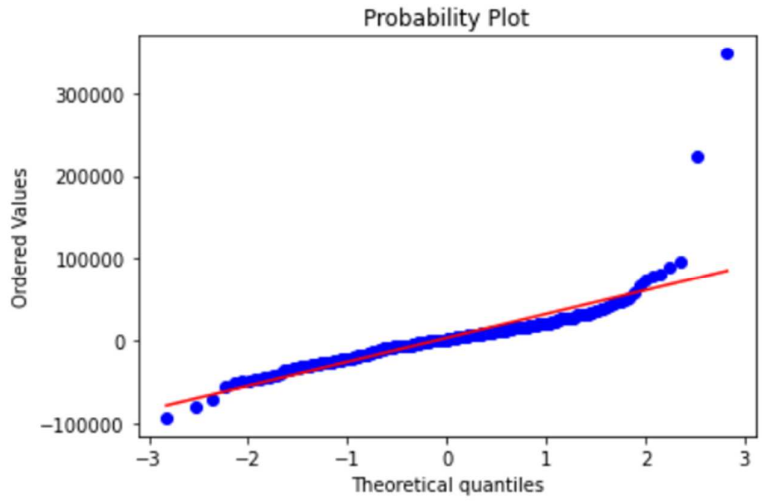
```
34496.60400589246
```

```
#Rmse using k-fold model.
mean_sqr_sgd_kfold=mean_squared_error(y_test,sgd_predict_kfold)
root_mean_sgd_kfold=np.sqrt(mean_sqr_sgd_kfold)
root_mean_sgd_kfold
```

```
34506.305690827125
```

```
#normality check.
st.probplot(y_test-sgd_predict,dist="norm",plot=plt)
```

```
          8.87430531e+04,  9.67599139e+04,  2.23112587e+05,  3.47774529e+05])),
 (28906.7597763056, 3566.321857938138, 0.8356663136892897))
```



Accuracy: Min acc: 72% , Max acc:88% , Mean acc:81% , RMSE: 34270

3.5 RANDOM FOREST REGRESSOR

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

```
#importing random forest regressor
from sklearn.ensemble import RandomForestRegressor
model_rf=RandomForestRegressor(random_state=2)
model_rf=model_rf.fit(x_train,y_train)
y_pred_rf=model_rf.predict(x_test)
```

```
#accuracy of test data
model_rf.score(x_test,y_test)
```

```
0.8865962473311346
```

```
#accuracy of train data
model_rf.score(x_train,y_train)
```

```
0.9751125403531459
```

```
#looks like some overfitting is occuring in random forest regressor
```

```
#Rmse before hyperparameter.
mean_sqr_rf=mean_squared_error(y_test,y_pred_rf)
root_mean_rf=np.sqrt(mean_sqr_rf)
root_mean_rf
```

```
28439.218391506158
```

```
#Hyper parameter tuning
from sklearn.model_selection import RandomizedSearchCV
parameters={'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

```
RF=RandomizedSearchCV(RandomForestRegressor(),param_distributions=parameters,cv=5)
```

```
RF.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(),
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]})
```

```
RF.best_estimator_
```

```
RandomForestRegressor(bootstrap=False, max_depth=70, max_features='sqrt',
                      n_estimators=800)
```

```
model_af_rfht=RandomForestRegressor(bootstrap=False, max_depth=30, max_features='sqrt',min_samples_leaf=2, min_samples_split=5,n_
y_pred_afht=model_af_rfht.fit(x_train,y_train).predict(x_test)
```

```
y_pred_hyp=RF.predict(x_test)
```

```
np.round(model_af_rfht.score(x_test,y_test),2)
```

```
0.89
```

```
#Rmse after hyperparameter.
mean_sqr_rfhp=mean_squared_error(y_test,y_pred_hyp)
root_mean_rfhp=np.sqrt(mean_sqr_rfhp)
root_mean_rfhp
```

```
27110.727170375067
```

Accuracy:88% ， RMSE: 28439

3.6 GRADIENT BOOSTING

Gradient Boosting algorithm is used to generate an ensemble model by combining the weak learners or weak predictive models. Gradient boosting algorithm can be used to train models for both regression and classification

problem. Gradient Boosting Regression algorithm is used to fit the model which predicts the continuous value.

```
#performing gradient boosting technique.
from sklearn.ensemble import GradientBoostingRegressor
model_GB=GradientBoostingRegressor()
model_GB.fit(x_train,y_train)
y_pred_GB=model_GB.predict(x_test)
```

```
#accuracy of test data
np.round(model_GB.score(x_test,y_test),2)
```

```
0.93
```

```
#accuracy of train data
model_GB.score(x_train,y_train)
```

```
0.9654578604850673
```

```
#there is no overfit or underfit here.
```

```
#performing k-fold
model_GB_kfold=GradientBoostingRegressor()
results_GB=cross_val_score(model_GB_kfold,x_ind,y_dep,cv=kfold_validation)
model_GB_kfold.fit(x_train,y_train)
y_pred_GB_kfold=model_GB_kfold.predict(x_test)
print(results_GB)
print(np.mean(results_GB))
```

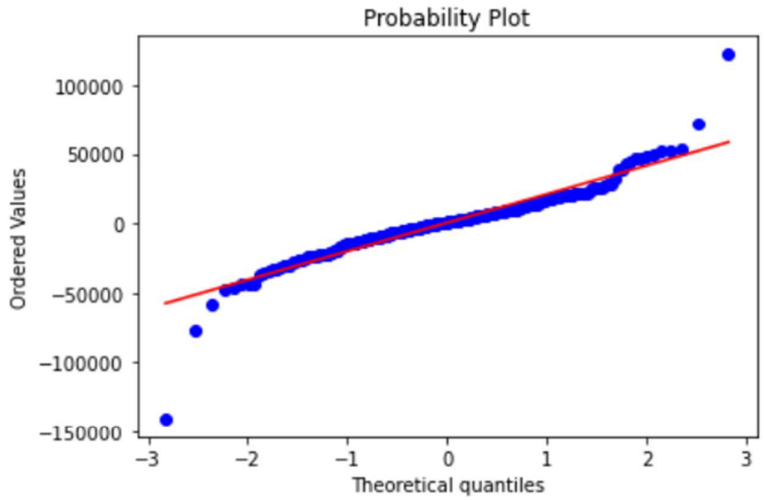
```
[0.89849066 0.87121662 0.8977116  0.89141326 0.89030974]
0.8898283759337922
```

```
#RMSE by using train test split technique.
mean_sqr_GB=mean_squared_error(y_test,y_pred_GB)
root_mean_GB=np.sqrt(mean_sqr_GB)
root_mean_GB
```

```
23041.873970358196
```

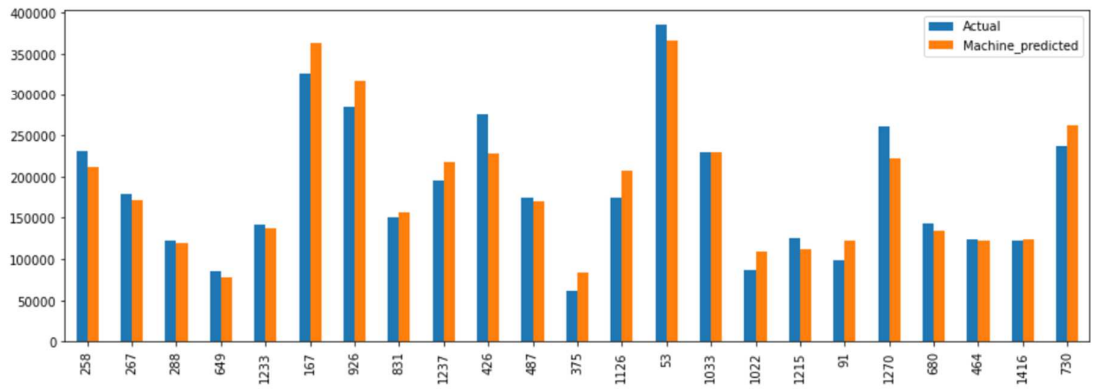
```
#RMSE by using K-fold technique.
mean_sqr_GB1=mean_squared_error(y_test,y_pred_GB_kfold)
root_mean_GB1=np.sqrt(mean_sqr_GB1)
root_mean_GB1
```

```
#normality check
st.probplot(y_test-y_pred_GB_kfold,dist="norm",plot=plt)
          5.21737170e+04,  5.31842660e+04,  7.23069243e+04,  1.22240044e+05])),
(20525.48392859303, 313.04190747769434, 0.9405180614977604))
```




```
#relation b/w actual and predicted in bar graph
f_com_GB=pd.DataFrame({"Actual":y_test,"Machine_predicted":y_pred_GB_kfold})
com_g_GB=f_com_GB.head(23)
```

```
com_g_GB.plot(kind="bar",figsize=(15,5));
```



Accuracy: Min acc: 86% , Max acc:90% , Mean acc:88% , RMSE: 23538

3.6 XGB Boosting

Extreme Gradient Boosting (XGBoost) is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm.

```
#performing XG boost regression.
from xgboost import XGBRegressor
model_XG=XGBRegressor()
model_XG.fit(x_train,y_train)
y_pred_XG=model_XG.predict(x_test)
```

```
#accuracy of test data
np.round(model_XG.score(x_test,y_test),2)
```

```
0.86
```

```
#accuracy of train data
model_XG.score(x_train,y_train)
```

```
0.9998026535805973
```

```
#looks like modelis overfitting here.
```

```
#performing k-fold in XGB model.
model_XG_kfold=XGBRegressor()
results_XG=cross_val_score(model_XG_kfold,x_ind,y_dep,cv=kfold_validation)
model_XG_kfold.fit(x_train,y_train)
y_pred_XG_kfold=model_XG_kfold.predict(x_test)
print(results_XG)
print(np.mean(results_XG))
```

```
[0.88273995 0.84375113 0.86599685 0.91323605 0.86252835]
0.873650467140795
```

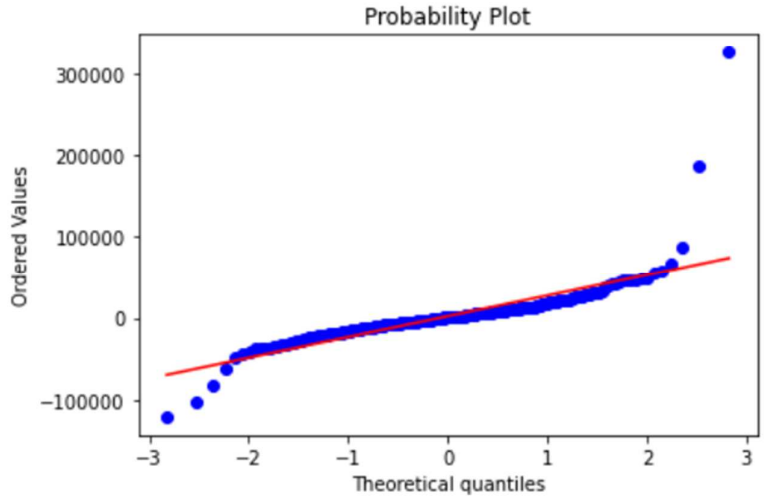
```
#Rmse for train test split model
mean_sqr_XGB=mean_squared_error(y_test,y_pred_XG)
root_mean_XGB=np.sqrt(mean_sqr_XGB)
root_mean_XGB
```

```
31152.96819305763
```

```
#Rmse for k-fold model.
mean_sqr_XGB_kfold=mean_squared_error(y_test,y_pred_XG_kfold)
root_mean_XGB_kfold=np.sqrt(mean_sqr_XGB_kfold)
root_mean_XGB_kfold
```

```
31152.96819305763
```

```
#normality check
st.probplot(y_test-y_pred_XG_kfold,dist="norm",plot=plt)
          6.71416562e+04,  8.80877188e+04,  1.87464766e+05,  3.27133031e+05])),
 (25446.001540360478, 2045.4129655393847, 0.8119601935974728))
```



Accuracy: Min acc: 86% , Max acc:91% , Mean acc:87% , RMSE: 31152

3.7 Overall results

Algorithm	Min Accuracy	Max Accuracy	Mean accuracy	Rmse
Linear Regression	nan	nan	82%	36202 er
Ridge Regression	69%	81%	75%	42284 er
Lasso Regression	70%	88%	82%	34108 er
SGD Regression	72%	88%	81%	34270 er
Random Forest	nan	nan	88%	28439 er
Gradient Boosting	86%	90%	88%	23538 er
XGB Boosting	86%	91%	87%	31152 er

CONCLUSION

In this study we performed several ML supervised algorithms.our goal is to build a model that predict saleprice of houses.Here after doing all this study Gradient Boosting Regressor gives (86 – 90%) accuracy with low RMSE .