HOUSE PRICE PREDICTION PROJECT

Submitted by:

J USHARANI

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to Flip Robo Technologies Bangalore for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I want to thank my SME Ms. Khusboo Garg for providing the Dataset and helping us to solve the problem and addressing out our Query in right time.

I would like to express my gratitude towards my parents & members of Flip Robo for their kind co-operation and encouragement which help me in completion of this project.

I would like to express my special gratitude and thanks to industry persons for giving me such attention and time.

INTRODUCTION

Business Problem Framing

Houses are one of the necessary needs of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy. It is a very large market and there are various companies working in the domain.

Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

We are required to model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

Conceptual Background of the Domain Problem

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

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

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

Motivation for the Problem Undertaken

Our main objective of doing this project is to build a model to predict the house prices with the help of other supporting features. We are going to predict by using Machine Learning algorithms.

The sample data is provided to us from our client database. In order to improve the selection of customers, the client wants some predictions that could help them in further investment and improvement in selection of customers.

House Price Index is commonly used to estimate the changes in housing price. Since housing price is strongly correlated to other factors such as location, area, population, it requires other information apart from HPI to predict individual housing price.

There has been a considerably large number of papers adopting traditional machine learning approaches to predict housing prices accurately, but they rarely concern themselves with the performance of individual models and neglect the less popular yet complex models.

As a result, to explore various impacts of features on prediction methods, this paper will apply both traditional and advanced machine learning approaches to investigate the difference among several advanced models. This paper will also comprehensively validate multiple techniques in model implementation on regression and provide an optimistic result for housing price prediction.

ANALYTICAL PROBLEM FRAMING

Mathematical/ Analytical Modelling of the Problem

We are building a model in Machine Learning to predict the actual value of the prospective properties and decide whether to invest in them or not. So, this model will help us to determine which variables are important to predict the price of variables & also how do these variables describe the price of the house. This will help to determine the price of houses with the available independent variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns.

Regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates', or 'features'). The most common form of regression analysis is linear regression, in which one finds the line (or a more complex linear combination) that most closely fits the data according to a specific mathematical criterion. For specific mathematical reasons this allows the researcher to estimate the

conditional expectation of the dependent variable when the independent variables take on a given set of values.

Regression analysis is also a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables.

The different Mathematical/Analytical models that are used in this project are as below:

1. Linear regression - is a linear model, e.g., a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).

2. Lasso - In statistics and machine learning, lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model.

3. Ridge - regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multi co linearity (correlations between predictor variables).

4. Elastic Net - is a popular type of regularized linear regression that combines two popular penalties, specifically the L1 and L2 penalty functions. Elastic net linear regression uses the penalties from both the lasso and ridge techniques to regularize regression models. The technique combines both the lasso and ridge regression methods by learning from their shortcomings to improve on the regularization of statistical models.

5. K Neighbors Regressor - KNN algorithm can be used for both classification and regression problems. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. This means that the new point is assigned a value based on how closely it resembles the points in the training set.

6. Decision Tree - is one of the most commonly used, practical approaches for supervised learning. It can be used to solve both Regression and Classification tasks with the latter being put more into practical application. It is a tree-structured classifier with three types of nodes.

7. 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. A Random Forest's nonlinear nature can give it a leg up over linear algorithms, making it a great option.

8. AdaBoost Regressor - is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction.

9. Gradient Boosting Regressor - GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

-> First, use the train dataset and do the EDA process, fitting the best model and saving the model.

-> Then, use the test dataset, load the saved model and predict the values over the test data.

Data Sources and their formats

Let's check the data now. Below I have attached the snapshot below to give an overview.

```
#Importing warning library to avoid any warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading the train dataset

```
import pandas as pd
df_train=pd.read_csv('D:/Python file/train.csv')  #Path location of the dataset
df_train.head()  #Checking out the top 5 rows of the dataset
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	M
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	MnPrv	NaN	0	
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

```
df_train.shape    #Checking the dimensions of the dataset
```

```
(1168, 81)
```

```
df_train.columns   #Checking out the columns of the dataset
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
```

Data description

Data contains 1460 entries each having 81 variables. The details of the features are given below:

1. ***MSSubClass**: Identifies the type of dwelling involved in the sale.*
2. ***MSZoning**: Identifies the general zoning classification of the sale.*
3. ***LotFrontage**: Linear feet of street connected to property*
4. ***LotArea**: Lot size in square feet*
5. ***Street**: Type of road access to property*
6. ***Alley**: Type of alley access to property*
7. ***LotShape**: General shape of property*

8. ***LandContour**: Flatness of the property*
9. ***Utilities**: Type of utilities available*
10. ***LotConfig**: Lot configuration*
11. ***LandSlope**: Slope of property*
12. ***Neighborhood**: Physical locations within Ames city limits*
13. ***Condition1**: Proximity to various conditions*
14. ***Condition2**: Proximity to various conditions (if more than one is present)*
15. ***BldgType**: Type of dwelling*
16. ***HouseStyle**: Style of dwelling*
17. ***OverallQual**: Rates the overall material and finish of the house*
18. ***OverallCond**: Rates the overall condition of the house*
19. ***YearBuilt**: Original construction date*
20. ***YearRemodAdd**: Remodel date (same as construction date if no remodeling or additions)*
21. ***RoofStyle**: Type of roof*
22. ***RoofMatl**: Roof material*
23. ***Exterior1st**: Exterior covering on house*
24. ***Exterior2nd**: Exterior covering on house (if more than one material)*
25. ***MasVnrType**: Masonry veneer type*
26. ***MasVnrArea**: Masonry veneer area in square feet*
27. ***ExterQual**: Evaluates the quality of the material on the exterior*
28. ***ExterCond**: Evaluates the present condition of the material on the exterior*
29. ***Foundation**: Type of foundation*
30. ***BsmtQual**: Evaluates the height of the basement*
31. ***BsmtCond**: Evaluates the general condition of the basement*
32. ***BsmtExposure**: Refers to walkout or garden level walls*
33. ***BsmtFinType1**: Rating of basement finished area*
34. ***BsmtFinSF1**: Type 1 finished square feet*
35. ***BsmtFinType2**: Rating of basement finished area (if multiple types)*
36. ***BsmtFinSF2**: Type 2 finished square feet*
37. ***BsmtUnfSF**: Unfinished square feet of basement area*

38. ***TotalBsmtSF****: Total square feet of basement area*
39. ***Heating****: Type of heating*
40. ***HeatingQC****: Heating quality and condition*
41. ***CentralAir****: Central air conditioning*
42. ***Electrical****: Electrical system*
43. ***1stFlrSF****: First Floor square feet*
44. ***2ndFlrSF****: Second floor square feet*
45. ***LowQualFinSF****: Low quality finished square feet (all floors)*
46. ***GrLivArea****: Above grade (ground) living area square feet*
47. ***BsmtFullBath****: Basement full bathrooms*
48. ***BsmtHalfBath****: Basement half bathrooms*
49. ***FullBath****: Full bathrooms above grade*
50. ***HalfBath****: Half baths above grade*
51. ***Bedroom****: Bedrooms above grade (does NOT include basement bedrooms)*
52. ***Kitchen****: Kitchens above grade*
53. ***KitchenQual****: Kitchen quality*
54. ***TotRmsAbvGrd****: Total rooms above grade (does not include bathrooms)*
55. ***Functional****: Home functionality (Assume typical unless deductions are warranted)*
56. ***Fireplaces****: Number of fireplaces*
57. ***FireplaceQu****: Fireplace quality*
58. ***GarageType****: Garage location*
59. ***GarageYrBlt****: Year garage was built*
60. ***GarageFinish****: Interior finish of the garage*
61. ***GarageCars****: Size of garage in car capacity*
62. ***GarageArea****: Size of garage in square feet*
63. ***GarageQual****: Garage quality*
64. ***GarageCond****: Garage condition*
65. ***PavedDrive****: Paved driveway*
66. ***WoodDeckSF****: Wood deck area in square feet*
67. ***OpenPorchSF****: Open porch area in square feet*
68. ***EnclosedPorch****: Enclosed porch area in square feet*
69. ***3SsnPorch****: Three season porch area in square feet*

70. ***ScreenPorch**: Screen porch area in square feet*
71. ***PoolArea**: Pool area in square feet*
72. ***PoolQC**: Pool quality*
73. ***Fence**: Fence quality*
74. ***MiscFeature**: Miscellaneous feature not covered in other categories*
75. ***MiscVal**: $Value of miscellaneous feature*
76. ***MoSold**: Month Sold (MM)*
77. ***YrSold**: Year Sold (YYYY)*
78. ***SaleType**: Type of sale*
79. ***SaleCondition**: Condition of sale*
80. ***Id:** Id of House*
81. ***SalePrice**: Price of House*

Checking the data type & info of dataset

```
df_train.info()    #Checking the info of all the columns present
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1168 non-null   int64
 1   MSSubClass     1168 non-null   int64
 2   MSZoning       1168 non-null   object
 3   LotFrontage    954 non-null    float64
 4   LotArea        1168 non-null   int64
 5   Street         1168 non-null   object
 6   Alley          77 non-null     object
 7   LotShape       1168 non-null   object
 8   LandContour    1168 non-null   object
 9   Utilities      1168 non-null   object
 10  LotConfig      1168 non-null   object
 11  LandSlope      1168 non-null   object
 12  Neighborhood   1168 non-null   object
 13  Condition1     1168 non-null   object
 14  Condition2     1168 non-null   object
 15  BldgType       1168 non-null   object
 16  HouseStyle     1168 non-null   object
 17  OverallQual    1168 non-null   int64
 18  OverallCond    1168 non-null   int64
 19  YearBuilt      1168 non-null   int64
 20  YearRemodAdd   1168 non-null   int64
 21  RoofStyle      1168 non-null   object
 22  RoofMatl       1168 non-null   object
 23  Exterior1st    1168 non-null   object
 24  Exterior2nd    1168 non-null   object
 25  MasVnrType     1161 non-null   object
 26  MasVnrArea     1161 non-null   float64
 27  ExterQual      1168 non-null   object
 28  ExterCond      1168 non-null   object
 29  Foundation     1168 non-null   object
 30  BsmtQual       1138 non-null   object
```

Checking the no. of null values in the dataset

```
df_train.isnull().sum().sort_values(ascending=False).head(30)   #Checking for null values in the dataset for top 30 columns
```

```
PoolQC          1161
MiscFeature     1124
Alley           1091
Fence            931
FireplaceQu      551
LotFrontage      214
GarageYrBlt       64
GarageFinish      64
GarageType        64
GarageQual        64
GarageCond        64
BsmtExposure      31
BsmtFinType2      31
BsmtQual          30
BsmtCond          30
BsmtFinType1      30
MasVnrType         7
MasVnrArea         7
Id                 0
Functional         0
Fireplaces         0
KitchenQual        0
KitchenAbvGr       0
BedroomAbvGr       0
HalfBath           0
FullBath           0
BsmtHalfBath       0
BsmtFullBath       0
TotRmsAbvGrd       0
GarageCars         0
dtype: int64
```

Data Pre-processing

Data pre-processing in Machine Learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for a building and training Machine Learning models. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre- process our data before feeding it into our model.

Checking the value counts of categorical data

```
#Checking the value counts of categorical data
for column in df_train.columns:
    if df_train[column].dtypes == object:
        print(str(column) + ' : ' + str(df_train[column].unique()))
        print(df_train[column].value_counts())
        print('\n')
```

```
MSZoning : ['RL' 'RM' 'FV' 'RH' 'C (all)']
RL         928
RM         163
FV          52
RH          16
C (all)      9
Name: MSZoning, dtype: int64


Street : ['Pave' 'Grvl']
Pave    1164
Grvl       4
Name: Street, dtype: int64


Alley : [nan 'Grvl' 'Pave']
Grvl    41
Pave    36
Name: Alley, dtype: int64


LotShape : ['IR1' 'Reg' 'IR2' 'IR3']
Reg    740
IR1    390
IR2     32
IR3      6
Name: LotShape, dtype: int64
```

Observations:

1. There is only one unique value present in utilities column, so that we will be dropping it.

2. In categorical columns there are missing values present in columns Alley, MasVnrType, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, FireplaceQu, GarageType, GarageFinish, GarageQual, GarageCond, PoolQC, Fence, MiscFeature.

Checking the percentage of missing data

```
def missing_values_table(df_train):
    mis_val = df_train.isnull().sum()
    mis_val_percent = 100 * df_train.isnull().sum() / len(df_train)
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
    mis_val_table_ren_columns = mis_val_table.rename(
    columns = {0 : 'Missing Values', 1 : '% of Total Values'})
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)
    print ("Your selected dataframe has " + str(df_train.shape[1]) + " columns.\n"
        "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")
    return mis_val_table_ren_columns
missing_values_table(df_train)
```

```
Your selected dataframe has 77 columns.
There are 17 columns that have missing values.
```

	Missing Values	% of Total Values
MiscFeature	1124	96.2
Alley	1091	93.4
Fence	931	79.7
FireplaceQu	551	47.2
LotFrontage	214	18.3
GarageType	64	5.5
GarageYrBlt	64	5.5
GarageFinish	64	5.5

Handling missing data

```
basement=['BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2']
#NA means No_Basement for all i in basement. Let's replace NAs with 'No_Basement'
for i in basement:
    df_train[i].fillna('No_Basement',inplace=True)
    print(df_train[i].value_counts())

#As per given definition, NA means None. Let's replace NAs with 'None'
df_train['MiscFeature'].fillna('None',inplace=True)
print(df_train['MiscFeature'].value_counts())

#As per given definition, NA means No_alley_access. Let's replace missing data with 'No_alley_access'
df_train['Alley'].fillna('No_alley_access',inplace=True)
print(df_train['Alley'].value_counts())

#As per given definition, NA means No_Fence. Let's replace missing data with 'No_Fence'
df_train['Fence'].fillna('No_Fence',inplace=True)
print(df_train['Fence'].value_counts())

#NA means No_Fireplace. Let's replace missing data with 'No_Fireplace'
df_train['FireplaceQu'].fillna('No_Fireplace',inplace=True)
print(df_train['FireplaceQu'].value_counts())

#Let's Impute the missing values and replace it with the median
df_train['LotFrontage'].fillna(df_train['LotFrontage'].median(),inplace=True)

garage=['GarageType','GarageFinish','GarageQual','GarageCond']
#NA means No_Garage for all i in garage
for i in garage:
    df_train[i].fillna('No_Garage',inplace=True)
    print(df_train[i].value_counts())

#As per dataframe "df" we can say that most of the rows of GarageYrBlt has same value as YearBuilt so we replace with that
df_train["GarageYrBlt"]=df_train["GarageYrBlt"].fillna(df_train["YearBuilt"])
print(df_train['GarageYrBlt'].value_counts())
```

```
#As per given values of MasVnrArea,  Let's replace missing data with 0's
df_train['MasVnrArea'].fillna(0,inplace=True)
print(df_train['MasVnrArea'].value_counts())

#Let's fill the missing values in MasVnrType with None
df_train['MasVnrType'] = df_train['MasVnrType'].fillna('None')
```

```
TA             517
Gd             498
Ex              94
No_Basement     30
Fa              29
Name: BsmtQual, dtype: int64
TA             1041
Gd               56
Fa               39
No_Basement      30
Po                2
Name: BsmtCond, dtype: int64
No             756
Av             180
Gd             108
Mn              93
No_Basement     31
Name: BsmtExposure, dtype: int64
Unf            345
GLQ            330
ALQ            174
BLQ            121
Rec            109
LwQ             59
No_Basement     30
Name: BsmtFinType1, dtype: int64
Unf            1002
Rec              43
LwQ              40
No_Basement      31
```

Dropping some unnecessary columns

```
: #Dropping Utilities column
  df_train.drop(['Utilities'],axis=1,inplace=True)
```

```
: #Dropping other unnecessary columns in the dataset
  df_train.drop('Id',axis=1,inplace=True)  # id column not necessary for prediction
  df_train.drop('PoolArea',axis=1,inplace=True) # contains same value in every row
  df_train.drop('PoolQC',axis=1,inplace=True) # contains same value in every row
```

Checking the statistical summary of the dataset

In descriptive statistics, summary statistics are used to summarize a set of observations, in order to communicate the largest amount of information as simply as possible. Summary statistics summarize and provide information about your sample data. It tells something about the values in data set. This includes where the average lies and whether the data is skewed.

The describe() function computes a summary of statistics pertaining to the Data Frame columns. This function gives the mean, count, max, standard deviation and IQR values of the dataset in a simple understandable way.

```
df_train.describe()    #Statistical summary of the dataset
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	GarageAr
count	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	...	1168.0000
mean	56.767979	70.807363	10484.749144	6.104452	5.595890	1970.930651	1984.758562	101.696918	444.726027	46.647260	...	476.8604
std	41.940650	22.440317	8957.442311	1.390153	1.124343	30.145255	20.785185	182.218483	462.664785	163.520016	...	214.4667
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000	...	0.0000
25%	20.000000	60.000000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000	...	338.0000
50%	50.000000	70.000000	9522.500000	6.000000	5.000000	1972.000000	1993.000000	0.000000	385.500000	0.000000	...	480.0000
75%	70.000000	79.250000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000	...	576.0000
max	190.000000	313.000000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000	...	1418.0000

8 rows × 36 columns

Observations:

-> Maximum standard deviation of 8957.44 is observed in LotArea column.

-> Maximum SalePrice of a house observed is 755000 and minimum is 34900.

-> In the columns MSSubclass, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfsF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, HalfBath, TotRmsAbvGrd, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, Miscval, salePrice mean is considerably greater than median so the columns are positively skewed.

-> In the columns FullBath, BedroomAbvGr, Fireplaces, Garagecars, GarageArea, YrSold Median is greater than mean so the columns are negatively skewed.

-> In the columns MSSubClass, LotFrontage, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtHalfBath, BedroomAbvGr, ToRmsAbvGrd, GarageArea , WoodDeckS F, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, MiscVal, SalePrice there is considerable difference between the 75 percentile and maximum so outliers are present.

Correlation Factor

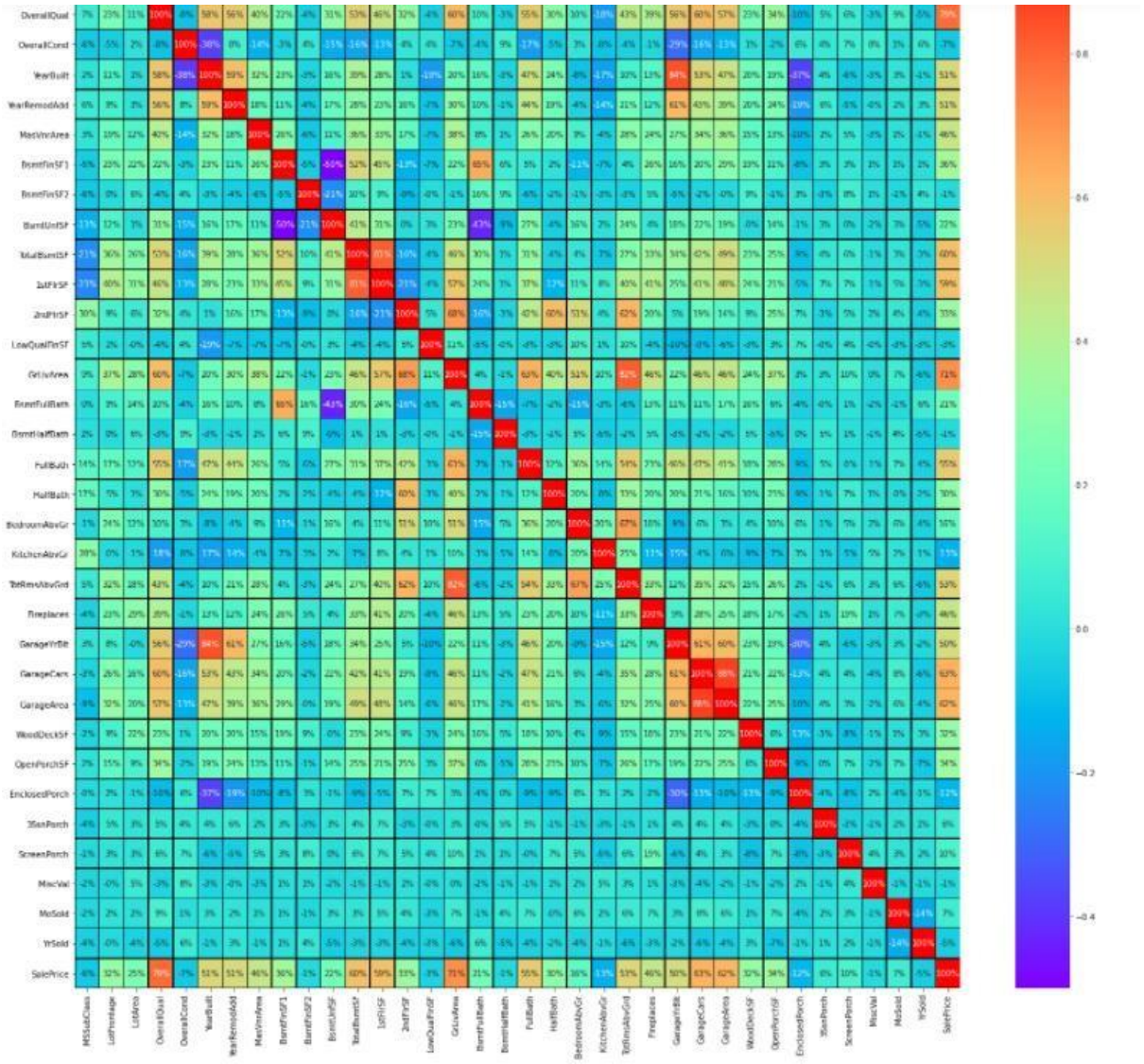
The statistical relationship between two variables is referred to as their correlation. The correlation factor represents the relation between columns in a given dataset. A correlation can be positive, meaning both variables are moving in the same direction or it can be negative, meaning that when one variable's value increasing, the other variable's value is decreasing.

```
#Checking correlation of the dataset
corr=df_train.corr()  #corr() function provides the correlation value of each column
corr
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...	GarageA
MSSubClass	1.000000	-0.336234	-0.124151	0.070462	-0.056978	0.023988	0.056618	0.028215	-0.052236	-0.062403	...	-0.092
LotFrontage	-0.336234	1.000000	0.296790	0.229981	-0.047851	0.112000	0.089513	0.188273	0.227732	0.001253	...	0.322
LotArea	-0.124151	0.296790	1.000000	0.107188	0.017513	0.005506	0.027228	0.120192	0.221851	0.056656	...	0.195
OverallQual	0.070462	0.229981	0.107188	1.000000	-0.083167	0.575800	0.555945	0.403985	0.219643	-0.040893	...	0.566
OverallCond	-0.056978	-0.047851	0.017513	-0.083167	1.000000	-0.377731	0.080669	-0.135133	-0.028810	0.044336	...	-0.126
YearBuilt	0.023988	0.112000	0.005506	0.575800	-0.377731	1.000000	0.592829	0.318562	0.227933	-0.027682	...	0.473
YearRemodAdd	0.056618	0.089513	0.027228	0.555945	0.080669	0.592829	1.000000	0.178583	0.114430	-0.044694	...	0.387
MasVnrArea	0.028215	0.188273	0.120192	0.403985	-0.135133	0.318562	0.178583	1.000000	0.263377	-0.064685	...	0.363
BsmtFinSF1	-0.052236	0.227732	0.221851	0.219643	-0.028810	0.227933	0.114430	0.263377	1.000000	-0.052145	...	0.286
BsmtFinSF2	-0.062403	0.001253	0.056656	-0.040893	0.044336	-0.027682	-0.044694	-0.064685	-0.052145	1.000000	...	-0.002
BsmtUnfSF	-0.134170	0.115628	0.006600	0.308676	-0.146384	0.155559	0.174732	0.108974	-0.499861	-0.213580	...	0.191
TotalBsmtSF	-0.214042	0.356180	0.259733	0.528285	-0.162481	0.386265	0.280720	0.362330	0.518940	0.098167	...	0.492
1stFlrSF	-0.227927	0.402864	0.312843	0.458758	-0.134420	0.279450	0.233384	0.334512	0.445876	0.093442	...	0.475
2ndFlrSF	0.300366	0.089816	0.059803	0.316624	0.036668	0.011834	0.155102	0.172136	-0.127656	-0.092049	...	0.135
LowQualFinSF	0.053737	0.008087	-0.001915	-0.039295	0.041877	-0.189044	-0.072526	-0.070026	-0.070932	-0.000577	...	-0.063
GrLivArea	0.086448	0.374000	0.281360	0.599700	-0.065006	0.198644	0.295048	0.384386	0.217160	-0.007484	...	0.459
BsmtFullBath	0.004556	0.092807	0.142387	0.101732	-0.039680	0.164983	0.104643	0.084498	0.645126	0.163518	...	0.166
BsmtHalfBath	0.008207	0.001375	0.059282	-0.030702	0.091016	-0.028161	-0.011375	0.014974	0.063895	0.093692	...	-0.020
FullBath	0.140807	0.171842	0.123197	0.548824	-0.171931	0.471264	0.444446	0.264357	0.054511	-0.060773	...	0.405

Correlation matrix and its visualization

A correlation matrix is a tabular data representing the 'correlations' between pairs of variables in a given dataset. It is also a very important pre-processing step in Machine Learning pipelines. The Correlation matrix is a data analysis representation that is used to summarize data to understand the relationship between various different variables of the given dataset.



Observations:

-> SalePrice is highly positively correlated with the columns OverallQual, YearBuilt, YearRemodAdd, TotalBsmtSF, 1stFlrSF, GrLivArea, FullBath, TotRmsAbvGrd, GarageCars, GarageArea.

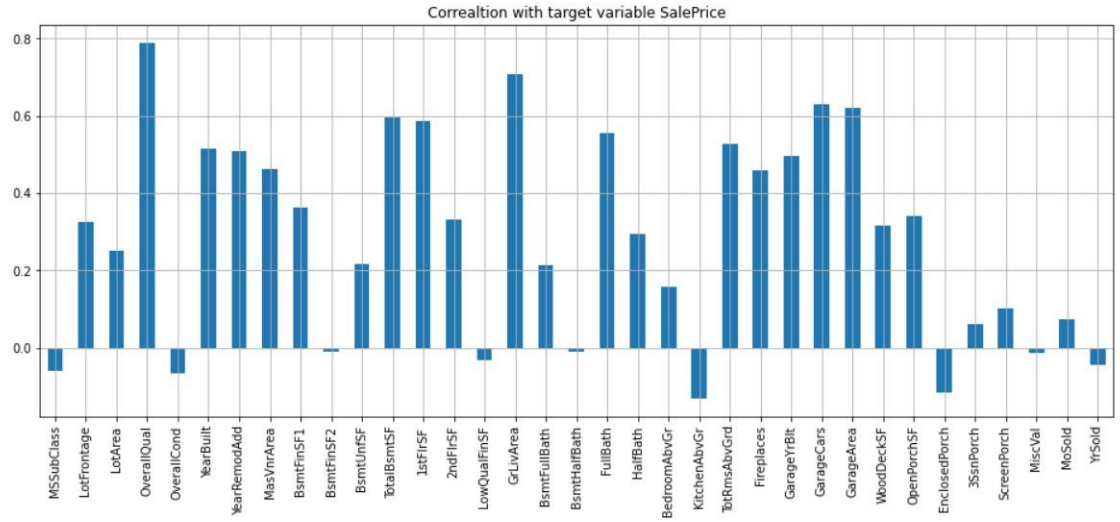
-> SalePrice is negatively correlated with OverallCond, KitchenAbvGr, Encloseporch, YrSold.

-> We observe multicollinearity in between columns, so we will be using Principal Component Analysis (PCA).

Correlation with target variable

```
#Checking the correlation with the target variable SalePrice
plt.figure(figsize=(16,6))
df_train.drop('SalePrice', axis=1).corrwith(df_train['SalePrice']).plot(kind='bar',grid=True)
plt.xticks(rotation=90)
plt.title("Correaltion with target variable SalePrice")
```

```
Text(0.5, 1.0, 'Correaltion with target variable SalePrice')
```



Observations:

>'MSSubClass','OverallCond','OverallCond','LowQualFinSF','BsmtHalf Bath','KitchenAbvGr','YrSold','EnclosedPorch','MiscVal' are negatively correlated with the target column, rest all are positively correlated

> 'OverallQual' & 'GrLivArea' are highly positively correlated with target column

>'MSSubClass','OverallCond','OverallCond','LowQualFinSF','BsmtHalf Bath','YrSold', 'MiscVal', 'MoSold', '3SsnPorch' are least correlated with the target column

```
#Dropping the least correlated columns from the dataset
df_train.drop(['MSSubClass' ,'LowQualFinSF' ,'BsmtHalfBath' ,'BsmtUnfSF' ,'YrSold', 'MiscVal',
               'MoSold', '3SsnPorch'],axis=1,inplace=True)
```

Encoding non-numeric data using Label Encoder

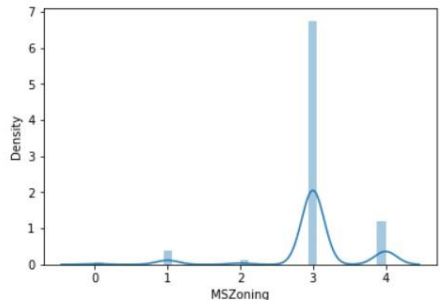
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
list1=['MSZoning','Street','Alley', 'LotShape','LandContour','LotConfig','LandSlope', 'Neighborhood','Condition1','Condition2',
       'BldgType','HouseStyle','RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond',
       'Foundation', 'BsmtQual','BsmtCond', 'BsmtExposure', 'BsmtFinType1','BsmtFinType2',  'Heating','HeatingQC','CentralAir',
       'Electrical', 'KitchenQual','Functional',  'FireplaceQu', 'GarageType','GarageFinish',  'GarageQual', 'GarageCond',
       'PavedDrive', 'Fence', 'MiscFeature',  'SaleType','SaleCondition', ]
for val in list1:
    df_train[val]=le.fit_transform(df_train[val].astype(str))
```

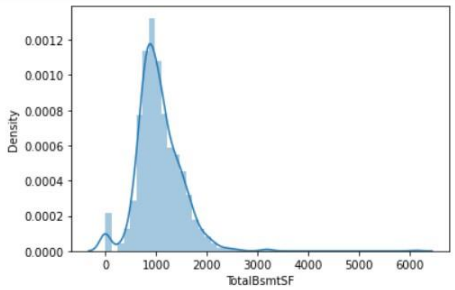
Checking skewness and plotting the distribution plot

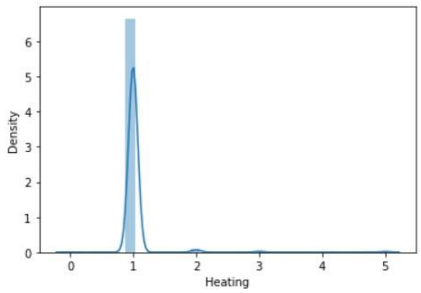
```
df_train.skew()
```

```
MSZoning         -1.796785
LotFrontage       2.733440
LotArea          10.659285
Street          -17.021969
Alley            -0.203241
                    ...
Fence            -1.955758
MiscFeature       4.958391
SaleType         -3.660513
SaleCondition    -2.671829
SalePrice         1.953878
Length: 69, dtype: float64
```

```
#Plotting distplot for checking the distribution of skewness
for col in df_train.describe().columns:
    sns.distplot(df_train[col])
    plt.show()
```







Skewness refers to distortion or asymmetry in a symmetrical bell curve, or normal distribution in a set of data. Besides positive and negative skew, distributions can also be said to have zero or undefined skew. The skewness value can be positive, zero, negative, or undefined.

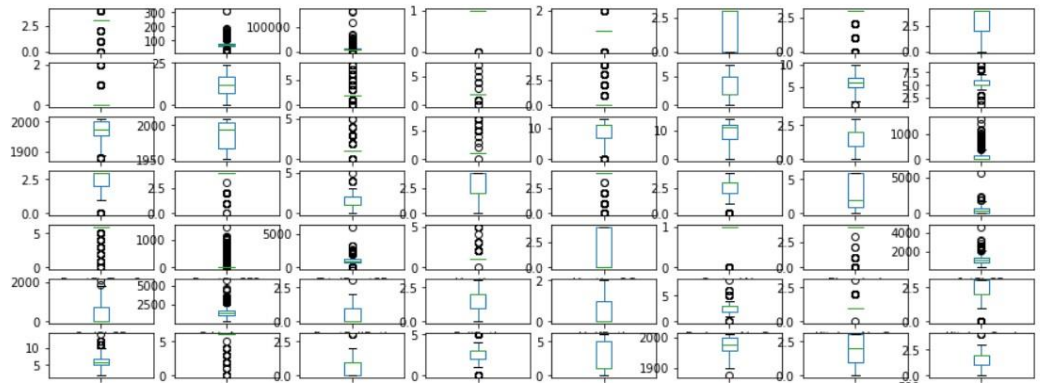
Checking outliers and plotting it

An outlier is a data point in a data set which is distant or far from all other observations available. It is a data point which lies outside the overall distribution which is available in the dataset. In statistics, an outlier is an observation point that is distant from other observations.

A box plot is a method or a process for graphically representing groups of numerical data through their quartiles. Outliers may also be plotted as an individual point. If there is an outlier it will plotted as point in box plot but other numerical data will be grouped together and displayed as boxes in the diagram. In most cases a threshold of 3 or - 3 is used i.e., if the Z-score value is higher than or less than 3 or -3 respectively, that particular data point will be identified as outlier.

```
df_train.plot(kind='box',subplots=True,layout=(12,8),figsize=(15,10))
```

```
MSZoning            AxesSubplot(0.125,0.826831;0.0824468x0.053169)
LotFrontage      AxesSubplot(0.223936,0.826831;0.0824468x0.053169)
LotArea          AxesSubplot(0.322872,0.826831;0.0824468x0.053169)
Street           AxesSubplot(0.421809,0.826831;0.0824468x0.053169)
Alley            AxesSubplot(0.520745,0.826831;0.0824468x0.053169)
                                    ...
Fence               AxesSubplot(0.125,0.316408;0.0824468x0.053169)
MiscFeature      AxesSubplot(0.223936,0.316408;0.0824468x0.053169)
SaleType         AxesSubplot(0.322872,0.316408;0.0824468x0.053169)
SaleCondition    AxesSubplot(0.421809,0.316408;0.0824468x0.053169)
SalePrice        AxesSubplot(0.520745,0.316408;0.0824468x0.053169)
Length: 69, dtype: object
```



Treating skewness

In the Data Science it is just statistics and many algorithms revolve around the assumption that the data is normalized. So, the more the data is close to normal, the better it is for getting good predictions. There are many ways of transforming skewed data such as log transform, square-root transform, box-cox transform, etc.

1. Log Transform

Log transformation is a data transformation method in which it replaces each variable x with a log(x). The log transformation is, arguably, the most popular among the different types of transformations used to transform skewed data to approximately conform to normality

2. Square Root Transform

The square root, x to x^ (1/2) = sqrt(x), is a transformation with a moderate effect on distribution shape: it is weaker than the logarithm and the cube root. It is also used for reducing right skewness, and also has the advantage that it can be applied to zero values. So, applying a square root transform inflates smaller numbers but stabilises bigger ones.

3. Box-Cox Transform

In statistics, a power transform is a family of functions that are applied to create a monotonic transformation of data using power functions. This is a useful data transformation technique used to stabilize variance, make the data more normal distribution-like, improve the validity of measures of association such as the Pearson correlation between variables and for other data stabilization procedures.

```
#We are treating skewness by using square root transform
import numpy as np
for col in df_x.skew().index:
    if col in df_x.describe().columns:
        if df_x[col].skew()>0.55:
            df_x[col]=np.sqrt(df_x[col])
        if df_x[col].skew()<-0.55:
            df_x[col]=np.sqrt(df_x[col])
```

```
df_x.skew()   #Checking skewness after treating it
```

```
MSZoning        -3.545090
LotFrontage     -0.236909
LotArea          0.304655
Street           0.000000
Alley           -0.174215
                   ...
ScreenPorch      3.364416
Fence           -2.816227
MiscFeature      4.459805
SaleType        -4.399310
SaleCondition   -3.286506
Length: 68, dtype: float64
```

Hardware and Software Requirements and Tools Used

For doing this project, the hardware used is a laptop with high end specification and a stable internet connection. While coming to software part, I had used anaconda navigator and in that I have used **Jupyter notebook** to do my python programming and analysis.

For using a csv file, Microsoft excel is needed. In Jupyter notebook, I had used lots of python libraries to carry out this project and I have mentioned below with proper justification:

1. Pandas- a library which is used to read the data, visualisation and analysis of data.
2. NumPy- used for working with array and various mathematical techniques.
3. Seaborn- visualization tool for plotting different types of plot.
4. Matplotlib- It provides an object-oriented API for embedding plots into applications.
5. zscore- technique to remove outliers.
6. skew ()- to treat skewed data using various transformation like sqrt, log, cube, boxcox, etc.
7. PCA- I used this to reduce the data dimensions to 10 columns.
8. standard scaler- I used this to scale my data before sending it to model.

9. train_test_split- to split the test and train data.
10. Then I used different classification algorithms to find out the best model for predictions.
11. joblib- library used to save the model in either pickle or obj file.

MODEL/S DEVELOPMENT AND EVALUATION

Identification of possible problem-solving approaches (methods)

From the given dataset it can be concluded that it is a Regression problem as the output column “SalesPrice” has continuous output. So, for further analysis of the problem, we have to import or call out the Regression related libraries in Python work frame.

The different libraries used for the problem solving are:
sklearn - Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy.

1. sklearn.linear_model
i. Linear Regression - Linear regression - is a linear model, e.g., a model that assumes a linear relationship between the input variables (x) and the single output variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables. The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regressions.

ii. Lasso - In statistics and machine learning, lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model.

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e., models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

iii. Ridge - The regression is a way to create a parsimonious model when the number of predictor variables in a set exceeds the number of observations, or when a data set has multicollinearity (correlations between predictor variables). Ridge regression is particularly useful to mitigate the problem of multicollinearity in linear regression, which commonly occurs in models with large numbers of parameters. In general, the method provides improved efficiency in parameter estimation problems in exchange for a tolerable amount of bias.

iv. Elastic Net - It is a popular type of regularized linear regression that combines two popular penalties, specifically the L1 and L2 penalty functions. Elastic net linear regression uses the penalties from both the lasso and ridge techniques to regularize regression models. The technique combines both the lasso and ridge regression methods by learning from their shortcomings to improve on the regularization of statistical models.

The elastic net method improves on lasso's limitations, i.e., where lasso takes a few samples for high dimensional data, the elastic net procedure provides the inclusion of "n" number of variables until saturation. In a case where the variables are highly correlated groups,

lasso tends to choose one variable from such groups and ignore the rest entirely.

To eliminate the limitations found in lasso, the elastic net includes a quadratic expression (||β||2) in the penalty, which, when used in isolation, becomes ridge regression. The quadratic expression in the penalty elevates the loss function toward being convex. The elastic net draws on best of both i.e., lasso and ridge regression. In the procedure for finding the elastic net method's estimator, there are two stages that involve both the lasso and regression techniques. It first finds the ridge regression coefficients and then conducts the second step by using a lasso sort of shrinkage of the coefficients.

2. sklearn.tree –
Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

There are several advantages of using decision trees for predictive analysis:

- Decision trees can be used to predict both continuous and discrete values i.e., they work well for both regression and classification tasks.
- They require relatively less effort for training the algorithm.
- They can be used to classify non-linearly separable data.
- They're very fast and efficient compared to KNN and other algorithms.

Decision tree learning is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree to go from observations about an item to conclusions about the item's target value.

Decision Tree Regressor - Decision Tree is one of the most commonly used, practical approaches for supervised learning. It can be used to solve both Regression and Classification tasks with the latter being put more into practical application. It is a tree-structured classifier with three types of nodes.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

3. sklearn.ensemble

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator. The sklearn.ensemble module includes two averaging algorithms based on randomized decision trees: the RandomForest algorithm and the Extra-Trees method. Both algorithms are perturb-and-combine techniques specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers.

Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence. Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction.

The different types of ensemble techniques used in the model are:

i. **Random Forest Regressor -** It 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. A Random Forest's nonlinear nature can give it a leg up over linear algorithms, making it a great option. Random forest is a type of supervised learning algorithm that uses ensemble methods (bagging) to solve both regression and classification problems. The algorithm operates by constructing a multitude of decision trees at training time and outputting the mean/mode of prediction of the individual trees.

ii. **AdaBoost Regressor -** It is a meta-estimator that begins by fitting a regressor on the original dataset and then fits additional copies of the regressor on the same dataset but where the weights of instances are adjusted according to the error of the current prediction.

iii. **Gradient Boosting Regressor -** GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

4. sklearn.metrics - The sklearn. metrics module implements several losses, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

Important sklearn.metrics modules used in the project are:

i. **mean_absolute_error -** In statistics, mean absolute error is a measure of errors between paired observations expressing the same phenomenon. Examples of Y versus X include comparisons of predicted versus observed, subsequent time versus initial time, and one technique of measurement versus an alternative technique of measurement.

The MAE measures the average magnitude of the errors in a set of forecasts, without considering their direction. It measures accuracy for continuous variables. Mean Absolute Error (MAE): MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

ii. mean_squared_error - In statistics, the mean squared error or mean squared deviation of an estimator measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. MSE is a risk function, corresponding to the expected value of the squared error loss. Mean Square Error (MSE) is defined as Mean or Average of the square of the difference between actual and estimated values.

iii. r2_score - In statistics, the coefficient of determination, denoted R^2 or r^2 and pronounced "R squared", is the proportion of the variance in the dependent variable that is predictable from the independent variable. R-squared is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determinations for multiple regressions.

5. sklearn.model_selection –

i. GridSearchCV - It is a library function that is a member of sklearn's model_selection package. It helps to loop through predefined hyper parameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters. GridSearchCV combines an estimator with a grid search preamble to tune hyper-parameters. The method picks the optimal parameter from the grid search and uses it with the estimator selected by the user.

ii. cross_val_score - Cross validation helps to find out the over fitting and under fitting of the model. In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces where each part being 20% of full dataset. While running the Cross validation the 1st part (20%) of the 5 parts will be kept out as a holdout set for validation and everything else is used for training data.

This way we will get the first estimate of the model quality of the dataset. In the similar way further iterations are made for the second 20% of the dataset is held as a holdout set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross-validation process to get the remaining estimate of the model quality.

cross_val_score estimates the expected accuracy of the model on out- of-training data (pulled from the same underlying process as the training data). The benefit is that one need not set aside any data to obtain this metric, and we can still train the model on all of the available data.

Testing of Identified Approaches

After completing the required pre-processing techniques for the model building data is separated as input and output columns before passing it to the train_test_split.

```
df_x=df_newtrain.drop(columns=['SalePrice'])
y=df_newtrain['SalePrice']
```

```
#Checking x data
df_x.head()
```

	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	...	GarageCond	PavedDrive	WoodDeckSF
0	3	70.0	4928	1	1	0	3	4	0	13	...	5	2	0
2	3	92.0	9920	1	1	0	3	1	0	15	...	5	2	180
3	3	105.0	11751	1	1	0	3	4	0	14	...	5	2	0
4	3	70.0	16635	1	1	0	3	2	0	14	...	5	2	240
5	3	58.0	14054	1	1	0	3	4	0	8	...	5	2	100

5 rows × 68 columns

```
#Checking y data after splitting
y.head()
```

```
0    128000
2    269790
3    190000
4    215000
5    219210
Name: SalePrice, dtype: int64
```

Scaling the data using Standard Scaler

For each value in a feature, StandardScaler subtracts the minimum value in the feature and then divides by the range. The range is the difference between the original maximum and original minimum. StandardScaler preserves the shape of the original distribution.

Sometimes model can be biased to higher values in dataset, so it is better to scale the dataset so that we can bring all the columns in common range. We can use StandardScaler here.

```
#Scaling the dataset using StandardScaler
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(df_x)
x=pd.DataFrame(x,columns=df_x.columns)
x
```

	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	...	GarageCond	PavedDrive	WoodD
0	0.045740	0.074811	-1.267021	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	0.147523	...	0.268178	0.270750	-0.5
1	0.045740	1.102980	0.160834	0.0	0.014247	-1.435050	0.275363	-0.726254	-0.204609	0.481624	...	0.268178	0.270750	0.5
2	0.045740	1.652949	0.588365	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	0.314574	...	0.268178	0.270750	-0.5
3	0.045740	0.074811	1.587624	0.0	0.014247	-1.435050	0.275363	-0.193229	-0.204609	0.314574	...	0.268178	0.270750	1.2
4	0.045740	-0.555343	1.081204	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	-0.687730	...	0.268178	0.270750	0.4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1090	0.045740	0.074811	0.136146	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	1.149826	...	0.268178	0.270750	-0.5
1091	0.045740	-0.077307	-0.126372	0.0	0.014247	0.718496	0.275363	0.560583	-0.204609	-0.854780	...	0.268178	-3.955457	-0.5
1092	0.045740	-2.835490	-2.357377	0.0	0.014247	0.718496	0.275363	-0.193229	-0.204609	0.147523	...	0.268178	0.270750	0.3
1093	-7.498944	-1.012502	-0.198747	0.0	3.914429	0.718496	0.275363	0.560583	-0.204609	-0.520679	...	-5.104820	-3.955457	-0.5
1094	0.045740	0.074811	-0.370343	0.0	0.014247	-1.435050	0.275363	0.560583	-0.204609	-0.687730	...	0.268178	0.270750	0.4

1095 rows × 68 columns

Using PCA

An important machine learning method for dimensionality reduction is called Principal Component Analysis. It is a method that uses simple matrix operations from linear algebra and statistics to calculate a projection of the original data into the same number or fewer dimensions.

```
# PCA is required for the analysis to reduce curse of Dimensionality & at the same time minimizing information loss
from sklearn.decomposition import PCA
for i in range(20,50):
    pca = PCA(n_components=i)
    x_pca=pca.fit_transform(x)
    print(i," variance :{}".format(np.sum(pca.explained_variance_ratio_)))
```

```
20  variance :0.6452176855013926
21  variance :0.6604797881840554
22  variance :0.6750091109588265
23  variance :0.6897255210832398
24  variance :0.7029699330377
25  variance :0.71638628884884
26  variance :0.7298331803443648
27  variance :0.7424910024837831
28  variance :0.755207680824439
29  variance :0.7674027365491365
30  variance :0.7791032902447095
31  variance :0.7905522618602978
32  variance :0.8016472303726692
33  variance :0.8124653847539705
34  variance :0.8228254085512873
35  variance :0.8327154483773845
36  variance :0.8423107313993045
37  variance :0.8518268908350162
38  variance :0.86143170939541
39  variance :0.8703086327905595
40  variance :0.8791675225715468
41  variance :0.8878607450232532
42  variance :0.8960359604000946
43  variance :0.9038561733454583
44  variance :0.911048606784388
45  variance :0.9178700572377332
46  variance :0.9244452402913237
47  variance :0.9303886690409281
48  variance :0.9369527210081303
49  variance :0.9427841466461516
```

```
: #As 49 has the highest value, we will select that
  pca = PCA(n_components=49)
  x=pca.fit_transform(x)
```

Run and evaluate selected models

We will find the best random state value so that we can create our train_test_split

```
#Importing required metrices and model for the dataset
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

```
#Finding the best random state
max_r_score=0
for r_state in range(42,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=r_state,test_size=0.20)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    y_pred=lr.predict(x_test)
    r2_scr=r2_score(y_test,y_pred)
    if r2_scr>max_r_score:
        max_r_score=r2_scr
        final_r_state=r_state
print("max r2 score corresponding to",final_r_state,"is",max_r_score)
```

```
max r2 score corresponding to 85 is 0.9109123245763177
```

Train Test Split

Scikit-learn is a Python library that offers various features for data processing that can be used for classification, clustering, and model selection. Model_selection is a method for setting a blueprint to analyze data and then using it to measure new data. Selecting a proper model allows you to generate accurate results when making a prediction. If we have one dataset, then it needs to be split by using the Sklearn train_test_split function first. By default, Sklearn train_test_split will make random partitions for the two subsets.

The train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, we don't need to divide the dataset manually. The train_test_split function is for splitting a single dataset for two different purposes: training and testing. The testing subset is for building your model. The testing subset is for using the model on unknown data to evaluate the performance of the model.

#Creating train_test_split using best random_state

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=85,test_size=.20)

Now, we will run a for loop for all regression algorithms and find the best model

```
#Importing the algorithms and other parameters
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
```

```
LR=LinearRegression()
l=Lasso()
en=ElasticNet()
rd=Ridge()
svr=SVR()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
```

```
models= []
models.append(('Linear Regression',LR))
models.append(('Lasso Regression',l))
models.append(('Elastic Net Regression',en))
models.append(('Ridge Regression',rd))
models.append(('Support Vector Regressor',svr))
models.append(('Decision Tree Regressor',dtr))
models.append(('KNeighbors Regressor',knr))
```

```
#Importing required metrics
from sklearn.metrics import mean_absolute_error,mean_squared_error
from sklearn.model_selection import cross_val_score
```

As you can see above, I had called the algorithms, then I called the empty list with the name models [ ], and calling all the model one by one and storing the result in that.

We can observe that I imported the metrics in order to interpret the model's output. Then I also selected the model to find the cross_validation_score value.

Let's check the code below:

```
#Finding the required metrices for all models together using a for loop
Model=[]
score=[]
cvs=[]
sd=[]
mae=[]
mse=[]
rmse=[]
for name,model in models:
    print('******************************',name,'**************************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=r2_score(y_test,pre)
    print('r2_score: ',AS)
    score.append(AS*100)
    print('\n')
    sc=cross_val_score(model,x,y,cv=5,scoring='r2').mean()
    print('cross_val_score: ',sc)
    cvs.append(sc*100)
    print('\n')
    std=cross_val_score(model,x,y,cv=5,scoring='r2').std()
    print('Standard Deviation: ',std)
    sd.append(std)
    print('\n')
    MAE=mean_absolute_error(y_test,pre)
    print('Mean Absolute Error: ',MAE)
    mae.append(MAE)
    print('\n')
    MSE=mean_squared_error(y_test,pre)
    print('Mean Squared Error: ',MSE)
    mse.append(MSE)
    print('\n')
    RMSE=np.sqrt(mean_squared_error(y_test,pre))
    print('Root Mean Squared Error: ',RMSE)
```

rmse.append(RMSE) print('\n\n') #Last 2 lines

As you can observe above, I made a for loop and called all the algorithms one by one and appending their result to models. The same I had done to store MSE, RMSE, MAE, SD and cross validation score. Let me show the output so that we can glance the result in more appropriate way.

The following are the outputs of the different algorithms I had used, along with the metrics score obtained and after finalizing the outputs in a data frame, it will be as follows:

```
#Finalizing the result
result=pd.DataFrame({'Model':Model, 'r2_score': score, 'Cross_val_score':cvs, 'Standard_deviation':sd,
                     'Mean_absolute_error':mae, 'Mean_squared_error':mse, 'Root_Mean_Squared_error':rmse})
result
```

	Model	r2_score	Cross_val_score	Standard_deviation	Mean_absolute_error	Mean_squared_error	Root_Mean_Squared_error
0	Linear Regression	91.091232	85.255524	0.029512	16265.055654	4.945806e+08	22239.168315
1	Lasso Regression	91.091758	85.300482	0.028811	16264.720263	4.945514e+08	22238.511733
2	Elastic Net Regression	90.769765	86.597983	0.021663	15908.455120	5.124272e+08	22636.855492
3	Ridge Regression	91.093274	85.435592	0.026772	16262.386277	4.944673e+08	22236.620268
4	Support Vector Regressor	-7.442555	-6.226721	0.058896	53839.411287	5.964799e+09	77232.108572
5	Decision Tree Regressor	76.877825	71.088232	0.018561	26519.442922	1.283654e+09	35828.124342
6	KNeighbors Regressor	87.183705	80.255817	0.027341	19643.309589	7.115116e+08	26674.174232

We can see that Ridge and Lasso Regression algorithms are performing well, as compared to other algorithms. Now we will try Hyperparameter Tuning to find out the best parameters and try to increase their scores.

Key Metrics for success in solving problem under consideration

The key metrics used here were r2_score, cross_val_score, sd, MAE, MSE and RMSE. We tried to find out the best parameters and also to increase our scores by using Hyperparameter Tuning and we will be using GridSearchCV method.

1. Cross Validation:

Cross-validation helps to find out the over fitting and under fitting of the model. In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces

where each part being 20% of full dataset. While running the Cross- validation the 1st part (20%) of the 5 parts will be kept out as a holdout set for validation and everything else is used for training data. This way we will get the first estimate of the model quality of the dataset.

In the similar way further iterations are made for the second 20% of the dataset is held as a holdout set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross-validation process to get the remaining estimate of the model quality.

2. R2 Score:

It is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.

3. Mean Squared Error (MSE):

MSE of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors — that is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss. RMSE is the Root Mean Squared Error.

4. Mean Absolute Error (MAE):

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

5. Hyperparameter Tuning:

There is a list of different machine learning models. They all are different in some way or the other, but what makes them different is nothing but input parameters for the model. These input parameters are named as **Hyperparameters.** These hyperparameters will define the architecture of the model, and the best part about these is that you get a choice to select these for your model. You must select from a specific list of hyperparameters for a given model as it varies from model to model.

We are not aware of optimal values for hyperparameters which would generate the best model output. So, what we tell the model is to explore and select the optimal model architecture automatically. This selection procedure for hyperparameter is known as **Hyperparameter Tuning.** We can do tuning by using **GridSearchCV.**

GridSearchCV is a function that comes in Scikit-learn (or SK-learn) model selection package. An important point here to note is that we need to have Scikit-learn library installed on the computer. This function helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, we can select the best parameters from the listed hyperparameters.

Lasso

```
from sklearn.model_selection import GridSearchCV
#Creating parameter list to pass in GridSearchCV
parameters={'alpha' :[0.001, 0.01, 0.1, 1], 'random_state':range(42, 100), 'selection':['cyclic','random']}
```

```
#Using GridSearchCV to run the parameters and checking final r2_score
l=Lasso()
grid=GridSearchCV(l,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)  #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'alpha': 1, 'random_state': 65, 'selection': 'random'}
0.8196648980841614
```

```
#Using the best parameters obtained
l=Lasso(alpha=1, random_state=65, selection='random')
l.fit(x_train,y_train)
pred=l.predict(x_test)
print('Final r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(l,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(l,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
Final r2_score after tuning is:  91.09179250030249
Cross validation score:  85.3011816759398
Standard deviation:  0.028799845722883945


Mean absolute error:  16264.631342747698
Mean squared error:  494549515.9424953
Root Mean squared error:  22238.469280561898
```

Ridge

```
#Creating parameter list to pass in GridSearchCV
parameters={'alpha' :[0.001, 0.01, 0.1, 1], 'random_state':range(42, 100), 'solver':['auto','lsqr','svd']}
```

```
#Using GridSearchCV to run the parameters and checking final r2_score
rd=Ridge()
grid=GridSearchCV(rd,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)  #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'alpha': 1, 'random_state': 42, 'solver': 'lsqr'}
0.8262610991004248
```

```
#Using the best parameters obtained
rd=Ridge(alpha=1, random_state=42, solver='lsqr')
rd.fit(x_train,y_train)
pred=rd.predict(x_test)
print('Final r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(rd,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(rd,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
Final r2_score after tuning is:  91.08700898348798
Cross validation score:  85.44016361657218
Standard deviation:  0.026632007008747523


Mean absolute error:  16275.224176881651
Mean squared error:  494815078.45046294
Root Mean squared error:  22244.43927030895
```

After Tuning the best algorithms, we can see that Lasso Regression has been improved slightly, i.e., r2_score from 91.09175 to 91.09179 and the errors have been decreased. Now, we will try Ensemble techniques like RandomForestRegressor, AdaBoostRegressor and GradientBoostingRegressor to boost up our scores.

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rfr=RandomForestRegressor(random_state=85)   #Using the best random state we obtained
parameters={'n_estimators':[10,50,100,500]}
grid=GridSearchCV(rfr,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)  #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'n_estimators': 500}
0.8443066387328602
```

```
#Using the best parameters obtained
RF=RandomForestRegressor(random_state=85, n_estimators=500)
RF.fit(x_train,y_train)
pred=RF.predict(x_test)
print('r2_score: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(RF,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(RF,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  92.29067233468501
Cross validation score:  85.20336522972778
Standard deviation:  0.017578642379780116


Mean absolute error:  15246.049872146119
Mean squared error:  427992305.4388967
Root Mean squared error:  20687.97489941673
```

AdaBoost Regressor

```
from sklearn.ensemble import AdaBoostRegressor
adr=AdaBoostRegressor(random_state=85)   #Using the best random state we obtained
parameters={'n_estimators':[10,50,100,500,1000],'learning_rate':[0.001,0.01,0.1,1],'loss':['linear','square']}
grid=GridSearchCV(adr,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)  #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'learning_rate': 1, 'loss': 'square', 'n_estimators': 1000}
0.8097546758624358
```

```
#Using the best parameters obtained
adr=AdaBoostRegressor(random_state=85, n_estimators=1000, learning_rate=1, loss='square')
adr.fit(x_train,y_train)
pred=adr.predict(x_test)
print("r2_score: ",r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(adr,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(adr,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  87.30376932705036
Cross validation score:  82.3356751775639
Standard deviation:  0.028502865675224746


Mean absolute error:  20705.41160820132
Mean squared error:  704846034.8296449
Root Mean squared error:  26548.936604497834
```

Gradient Boosting Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
gbr=GradientBoostingRegressor(random_state=85)   #Using the best random state we obtained
parameters={'n_estimators':[10,50,100,500,1000]}
grid=GridSearchCV(gbr,parameters,cv=5,scoring='r2')
grid.fit(x_train,y_train)
print(grid.best_params_)  #Printing the best parameters obtained
print(grid.best_score_) #Mean cross-validated score of best_estimator
```

```
{'n_estimators': 500}
0.8661209613328291
```

```
#Using the best parameters obtained
gbr=GradientBoostingRegressor(random_state=85, n_estimators=500)
gbr.fit(x_train,y_train)
pred=gbr.predict(x_test)
print("r2_score: ",r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(gbr,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(gbr,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  92.69990574609956
Cross validation score:  86.97701960036764
Standard deviation:  0.013412715018098998


Mean absolute error:  14797.68750518452
Mean squared error:  405273235.91459244
Root Mean squared error:  20131.399253767544
```

After applying Ensemble Techniques, we can see that GradientBoostingRegressor is the best performing algorithm among all other algorithms as it is giving a r2_score of 92.69 and cross validation score of 86.97. It has also the less amount of error values obtained. Lesser the RMSE score, the better the model. Now we will finalize the model.

Final the model

```
gbr_prediction=gbr.predict(x)
print('Predictions of GradientBoosting Regressor: ',gbr_prediction)   #Printing the predicted values
```

```
Predictions of GradientBoosting Regressor:  [128679.13374394 242192.25510016 191592.41147026 ... 150228.5566519
  41089.75453909 186590.90195356]
```

```
#Saving the model
import pickle
filename='HousingPrice_Project.pkl'   #Specifying the filename
pickle.dump(gbr,open(filename,'wb'))
```

Saving the predicted values

```
train_results=pd.DataFrame(gbr_prediction)
train_results.to_csv('HousingPrice_Project_TrainDataResults.csv')
```

Now, we have saved the model which has the predictions of the training data we had used. Next, we will use the test data and find the predictions using the best model we obtained

Using the test dataset and doing pre-processing

```
df_test=pd.read_csv('D:/Python file/test.csv')  #Path location of the dataset
df_test.head()  #Checking out the top 5 rows of the dataset
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeatu
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	...	0	0	NaN	NaN	N
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	N
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	...	0	0	NaN	NaN	N
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	...	0	0	NaN	NaN	N
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	N

5 rows × 80 columns

```
df_test.shape    #Checking the dimensions of the dataset
```

```
(292, 80)
```

```
#Removing all the columns we did in train dataset
df_test.drop(['Utilities'],axis=1,inplace=True)
df_test.drop('Id',axis=1,inplace=True)  # id column not necessary for prediction
df_test.drop('PoolArea',axis=1,inplace=True) # contains same value in every row
df_test.drop('PoolQC',axis=1,inplace=True) # contains same value in every row
```

```
df_test.isnull().sum().sort_values(ascending=False).head(30)   #Checking for null values in the dataset for top 30 columns
```

```
MiscFeature     282
Alley           278
Fence           248
FireplaceQu     139
LotFrontage      45
GarageYrBlt      17
GarageType       17
```

Here, we will be doing the same steps as we did for training dataset like handling missing data, dropping unnecessary columns, encoding non-categorical data, treating skewness, etc. Then, we will scale the data and do PCA analysis according to the best model requirements.

Predicting over the test data

Loading the saved model

```
fitted_model=pickle.load(open('HousingPrice_Project.pkl','rb'))
```

```
fitted_model   #Checking the model we have saved
```

```
GradientBoostingRegressor(n_estimators=500, random_state=85)
```

We can see that GradientBoostingRegressor algorithm, which was finalized and saved after we found that it was the best model performing, is loaded and it is also showing the best parameters we obtained while doing Hyperparameter Tuning.

Predictions over test data

```
test_predictions=fitted_model.predict(x)   #Predicting the values
```

```
#Making a dataframe for the predictions
test_predictions=pd.DataFrame(test_predictions,columns=['SalePrice'])
test_predictions
```

	SalePrice
0	328183.624429
1	192654.690274
2	245621.546189
3	132365.044128
4	265439.582053
...	...
287	246263.877783
288	149651.851471
289	139491.205645
290	150628.344534
291	150128.419916

292 rows × 1 columns

We have predicted the values over the test data and now we will save the predicted values separately

Saving the predicted values

```
test_results=pd.DataFrame(test_predictions)
test_results.to_csv('HousingPrice_Project_TestDataResults.csv')
```

Visualizations

Now, we will see the different plots done with this dataset in order to know the insight of the data present. Below are the codes given for the plots and the output obtained:

Importing required libraries and plotting graphs for categorical data

Exploratory Data Analysis

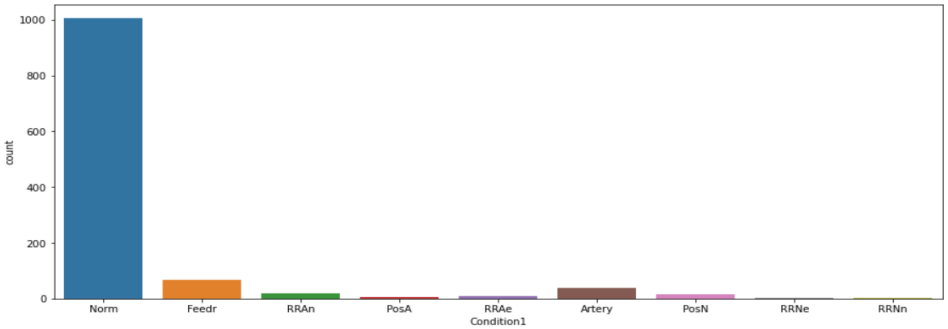
```
#Importing Matplotlib and Seaborn
import seaborn as sns
import matplotlib.pyplot as plt
```

Univariate Analysis

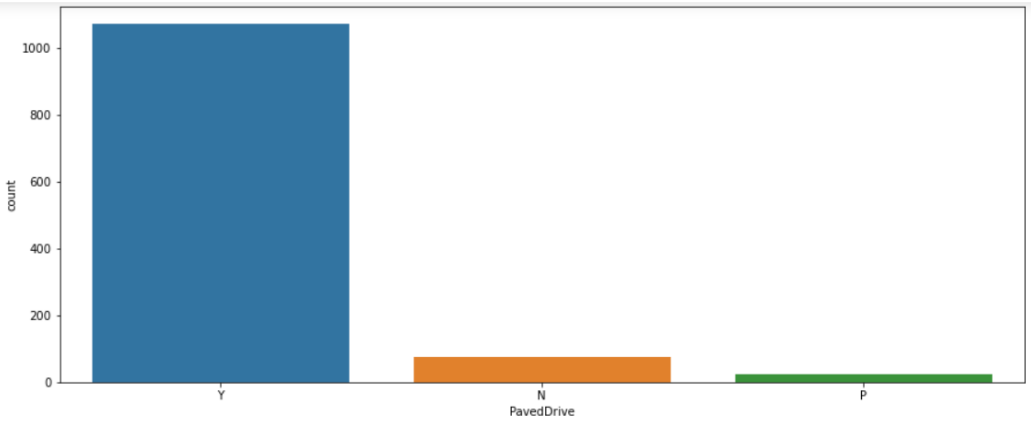
```
#Taking the categorical data alone and analysing it
categorical=[x for x in df_train.columns if df_train[x].dtype==object]
```

```
#Plotting countplot for each categorical data
for i in categorical:
    plt.figure(figsize=(15,6))
    sns.countplot(df_train[i])
    plt.show()

#Checking value counts and percentage of data classification in each string attribute
    print("Value counts:\n")
    print(df_train[i].value_counts())
    print('\n')
    print("Percentage of data: \n")
    print(round(df_train[i].value_counts()/1168*100),2)  #1168 is the entire data
```

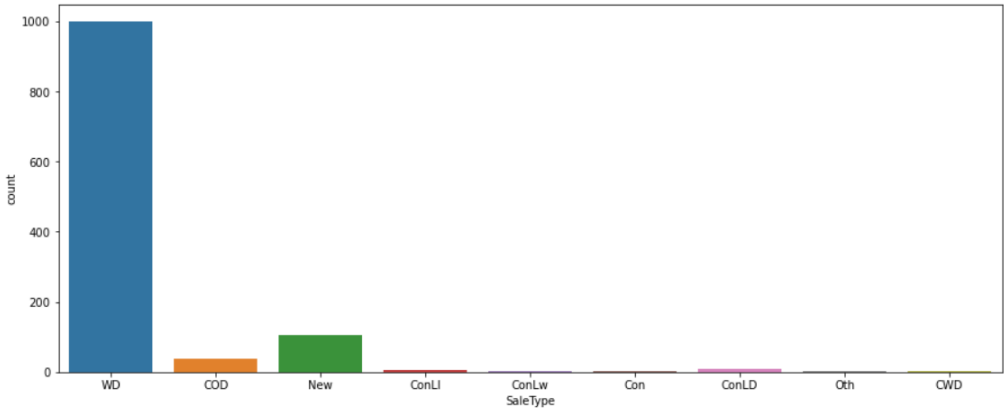
Below are some of the graphs we obtain after running this code:




```
Value counts:

Norm      1005
Feedr       67
Artery      38
RRAn        20
PosN        17
RRAe         9
PosA         6
RRNn         4
RRNe         2
Name: Condition1, dtype: int64
```




```
Value counts:

Y    1071
N      74
P      23
Name: PavedDrive, dtype: int64


Percentage of data:

Y    92.0
N     6.0
P     2.0
Name: PavedDrive, dtype: float64 2
```




```
Value counts:

WD       999
New      106
COD       38
ConLD      8
ConLI      5
ConLw      4
Oth        3
CWD        3
Con        2
Name: SaleType, dtype: int64
```

Observations:

-> By looking at count plot of MSZoning, which Identifies the general zoning classification of the sale, we find that 79 % of houses were sold in Low density residential areas.

-> For street, which states: Type of road access to property, we observe that almost 100% of house which were sold had access to paved roads so we can consider that no houses were purchased which had gravel road access.

-> For Alley,93% of the purchased house do not have access to alley. Only 4% have gravel & 3% have paved alley.

-> LotShape: 63% of the sold property was of Regular shape followed by slightly irregular type (33%). It means Australian gives priority to regular shaped houses.

-> LandContour :90% of sold houses were nearly flat level.

-> LotConfig: 72% of purchased houses had Inside lot of the property.

-> LandSlope: Around 95% of the sold property had gentle slope

-> Neighborhood: Physical locations within Ames city limits-: highest 16% of purchased houses has neighbourhood of NWAmes(Northwest Ames) followed by CollgCr(College Creek) and least houses were purchased in neighbourhood of Bluestem

-> Condition1: Proximity to various conditions-:86% of purchased houses had normal proximity to various conditions1 and least 0.00 had RRne, RRNn proximity

-> Condition2: Proximity to various conditions (if more than one is present)-:99% of purchased houses had normal proximity to various conditions2

-> BldgType: Type of dwelling-:84% purchased houses were single family detached, followed by 8% 2FmCon (Two-family Conversion)

-> HouseStyle: Style of dwelling-:49% houses had 1story followed by 2story style (31%)

-> RoofStyle: Type of roof-:78% of houses have Gable roof style and 19% have Hip roof style

-> RoofMatl: Roof material-:98% houses have CompShg (Standard (Composite) Shingle) roof material

-> Exterior1st: Exterior covering on house-:34% houses have Vinylsiding covering on exteriors 15% have hard board and metal siding

-> Exterior2nd: Exterior covering on house (if more than one material)-
:33% houses have VinylSd(Vinyl Siding) 15% have hard board and metal siding

-> MasVnrType: Masonry veneer type-:60% of houses have no masonry veneer type followed by BrkFace (Brick Face) (30%)

-> ExterQual: Evaluates the quality of the material on the exterior-:61% of the sold hoUse have TA(Average/Typical) quality material on exterior followed by Gd (Good) 34%

-> ExterCond: Evaluates the present condition of the material on the exterior-:88% houses are currently in TA (average) condition of exterior material

-> Foundation: Type of foundation-:44% houses have foundation CBlock (Cinder Block) & 44% have PConc(Poured Contrete)

-> BsmtQual: Evaluates the height of the basement-:44% of houses have TA (typical) (80-89 inches) basement height followed by Gd (Good) (90-99 inches)

-> BsmtCond: Evaluates the general condition of the basement-:89% of houses have TA (Typical - slight dampness allowed) basment

-> BsmtExposure: Refers to walkout or garden level walls-:64% of houses have No (No Exposure) followed by Av (Average Exposure) 15%

-> BsmtFinType1: Rating of basement finished area-:(30%) have Unf(Unfinshed) basement area and 28% comes under GLQ(good living quarters)

-> Heating: Type of heating-:98% houses have GasA (Gas forced warm air furnace) heating type

-> HeatingQC: Heating quality and condition-:30% houses have average quality heating

-> CentralAir: Central air conditioning-:93% houses are central air

-> Electrical: Electrical system-:92% houses have SbrKr (Standard Circuit Breakers & Romex) type of electrical systems

-> KitchenQual: Kitchen quality-:49% houses have average (TA) kitchen quality

-> Functional: Home functionality (Assume typical unless deductions are warranted)-:92% houses have typical (TA) home functionality

-> FireplaceQu: Fireplace quality-:47% of the houses donot have fireplace,25% houses have Gd (Good - Masonry Fireplace in main level) FireplaceQuality

-> GarageType: Garage location-:57% houses have Attached garage type, while 29% have Detchd (Detached from home)

-> GarageFinish: Interior finish of the garage:42% of houses have unfinished garage while 29% have RFn (Rough Finished)

-> GarageQual: Garage quality-:90% of houses have average garage quality

-> GarageCond: Garage condition-:91% of houses have TA (average garage condition)

-> PavedDrive: Paved driveway-:92% of houses have Y (paved drive) way

-> Fence: Fence quality-:89% houses have NA (no fence).

-> MiscFeature: Miscellaneous feature-:96% houses have no miscellaneous features

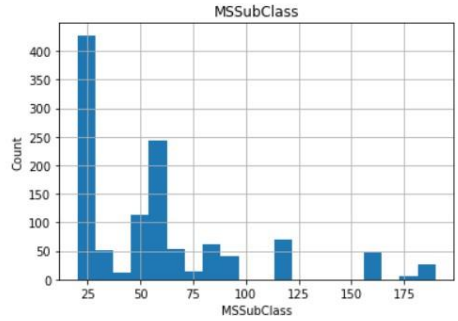
-> SaleType: Type of sale-:85% houses have sale type WD (warranty deed -conventional)

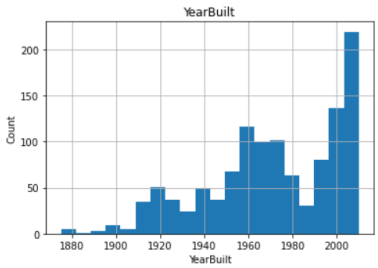
-> SaleCondition: 81% of houses are in normal sale condition

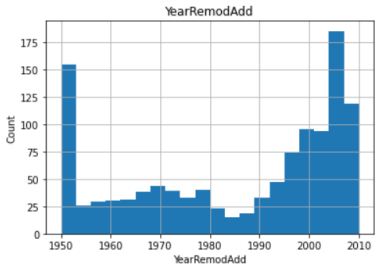
Taking all continuous data and plotting histogram

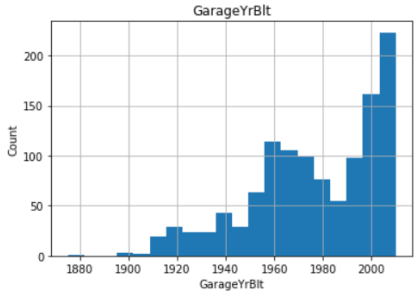
```
#Taking all continuous data and analyzing it
cont=[x for x in df_train.columns if x not in categorical]
```

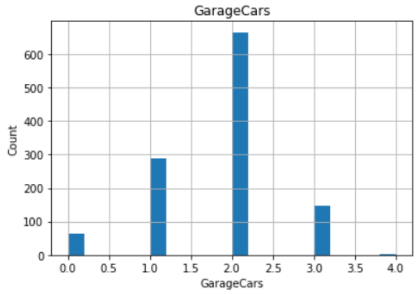
```
#visualizing each continuous features using histogram
for i in cont:
    data=df_train.copy()
    data[i].hist(bins=20)
    plt.xlabel(i)
    plt.ylabel("Count")
    plt.title(i)
    plt.show()
```











Observations:

-> lotFrontage:Almost all houses have LotFrontage between 20 to 150

-> lotArea:Around 580 house have lot Area between (0-10000)sqft.Very few houses have lot area aound 120000sqft & around 160000sqft

-> OverallQual: Rates the overall material and finish of the house-: Around 300 houses sold were in average condition. Only 10-15 houses were in excellent condition.

-> YearBuilt: Original construction date-: More number of people have brought the houses build after 1990

-> MasVnrArea: Masonry veneer area in square feet-:50% of houses have Masonry veneer area as '0-50' and out of rest 50% houses most houses have Masonry veneer area 50-1200

-> BsmtFinSF1: Type 1 finished square feet-: Most houses have Type 1 finished square feet area of basement between 0 and 1500

-> BsmtFinSF2: Type 2 finished square feet-: Around 1000 houses have Type 2 finished square feet area of 0

-> BsmtUnfSF: Unfinished square feet of basement area-: Around 130 houses have unfinished basement of area around 100-500 sqft

-> 1stFlrSF: First Floor square feet-: Around 280 houses have 1st floor square feet area between 800-1200sqft

-> GrLivArea: Above grade (ground) living area square feet-: Most houses have above ground living sq ft area in between 800 to 3000

-> BsmtFullBath: Basement full bathrooms-:50% houses have no full bathrooms in basement and in remaining houses most have 1 full bathroom in basement and very few has 2 full bathrooms

-> FullBath: Full bathrooms above grade-:25% houses have 1 full bathrooms above ground and 50% have 2 full bathrooms located above ground and very less have 3

-> HalfBath: Half baths above grade-: around 700 houses have no half bathrooms very few has 1 half bathroom

-> Bedroom: Bedrooms above grade (does NOT include basement bedrooms)-: Most houses have 3 bedrooms above ground followed by 2 and 4

-> Kitchen: Kitchens above grade-: Maximum houses have 1 Kitchen. very few have 2

-> TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)-: Around 300 houses have 6 rooms, around 200 have 5, &250 have 7. Very few have 12 & 14 rooms

-> Fireplaces: Number of fireplaces-: Most houses have 0 fireplaces followed by 1

-> GarageCars: Size of garage in car capacity-: Most houses have garage with 2 car capcity

-> GarageArea: Size of garage in square feet-: Most houses have Garage area in between 200 to 800

-> woodDeckSF: Wood deck area in square feet-: More than 50% of houses have 0 Wood Deck sqft area and rest have in between 0 to 400

-> OpenPorchSF: Open porch area in square feet-:25% of houses have 0 open porch sqft area and rest have in between 0 to 300

-> EnclosedPorch: Enclosed porch area in square feet-: Almost all houses have 0 enclosed porch sqft area

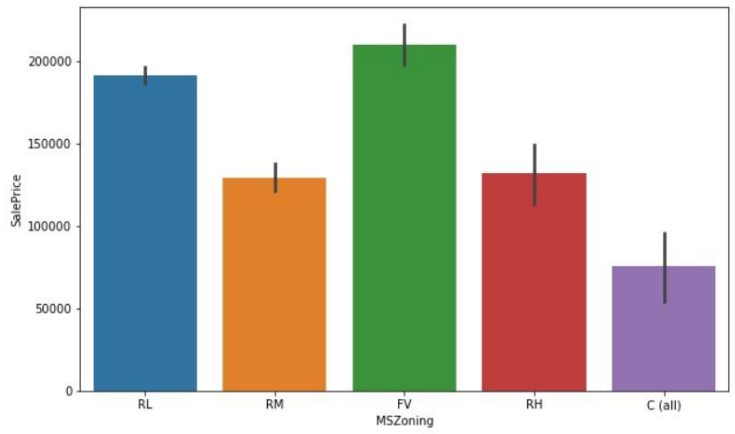
-> ScreenPorch: Screen porch area in square feet-: Almost all houses have 0 screen porch area sqft

-> Sale Price-: Around 500 houses have sale price in between 100000 to 200000.Very few houses have sale price of 600000 & 700000

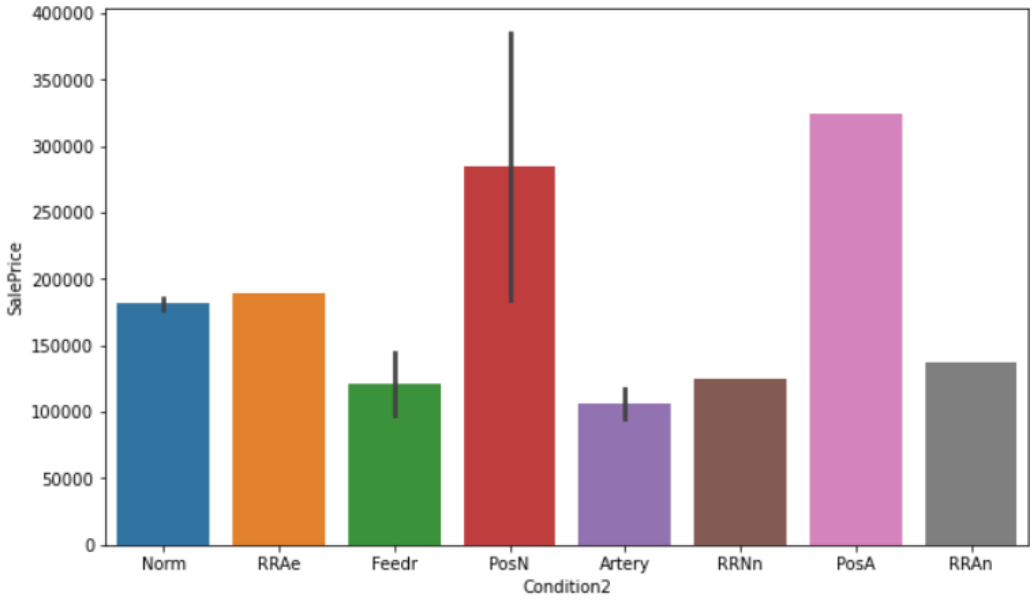
Bivariate Analysis with SalesPrice for categorical data

Bivariate Analysis

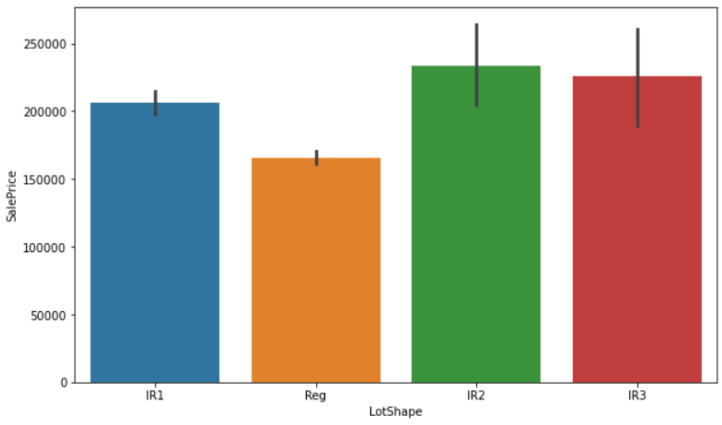
```
#Visualizing each categorical column wrt sale price
for i in categorical:
    plt.figure(figsize=(10,6))
    sns.barplot(x=df_train[i],y=df_train['SalePrice'])
    plt.show()
    print(df_train.groupby(df_train[i])['SalePrice'].mean())  #Calculating mean value of SalePrice
```




```
MSZoning
C (all)     75208.888889
FV         209478.461538
RH         131558.375000
RL         191004.181034
```




```
Condition2
Artery    106500.000000
Feedr     121166.666667
Norm      181697.129983
PosA      325000.000000
PosN      284875.000000
RRAe      190000.000000
RRAn      136905.000000
RRNn      125000.000000
Name: SalePrice, dtype: float64
```




```
LotShape
IR1    206038.464103
IR2    233827.750000
IR3    226120.833333
Reg    165906.660811
Name: SalePrice, dtype: float64
```

Above are some of the plots obtained and its value counts for bivariate analysis of all columns with the target variable SalesPrice.

Observations:

-> MSZoning: The avg sale price of the house is maximum in FV (Floating Village Residential) foloowed by RL (Residential Low Density) zone

-> Street: The property that have access to paved road have much higher average sale price as compared to that with gravel street

-> Alley: houses that do not have access to alley have higher sale price as compared to those with paved or gravel alley

-> LotShape: sale price is not much affected by lotshape, however IR2(Moderately Irregular) have a bit higher price compared to other while Reg (Regular) have lowest avg sale price

-> LandContour: Flatness of the property-: HLS (Hillside - Significant slope from side to side) have maximum average sale price & Bnk (Banked - Quick and significant rise from street grade to building) have minimum average sale price

-> LandSlope: It doesn't affect the average sale price of house

-> Neighborhood: The houses that has a neighbourhood of NoRidge (Northridge) has the maximum sale price followed by that with a neighbourhood of NridgHt (Northridge Heights)

-> Condition1: house that is RRAn (Adjacent to North-South Railroad) has hightest avg sale price followed by PosA (Adjacent to positive off- site feature) while houses that is Artery (Adjacent to arterial street) has a minimum average sale price.

->BldgType: Type of dwelling-: TwnhsE(Townhouse End Unit) & 1Fam(Single-family Detached) type house have highest selling price.

-> HouseStyle: Style of dwelling-: The average sale price of 2.5Fin (Two and one-half story) is maximum followed by 2Story (Two story). 1.5Unf (One and one-half story: 2nd level unfinished) have lowest avg selling price

-> RoofMatl: Roof material-: House with roof material WdShngl (Wood Shingles) have a very high average selling price, followed by that with roof of WdShake (Wood Shakes), while house with roof material Roll (Roll) have lowest sale price

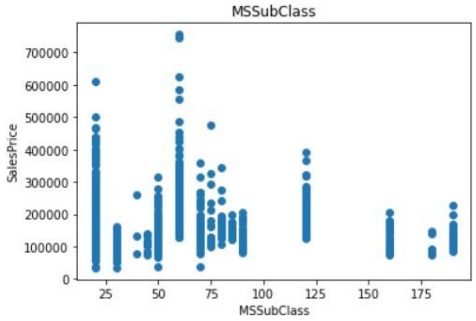
-> Exterior1st: Exterior covering on house-: House with exterior covering of ImStucc (Imitation Stucco) have maximum selling price while that with exterior coverng of BrkComm (Brick Common) have minimum average selling price

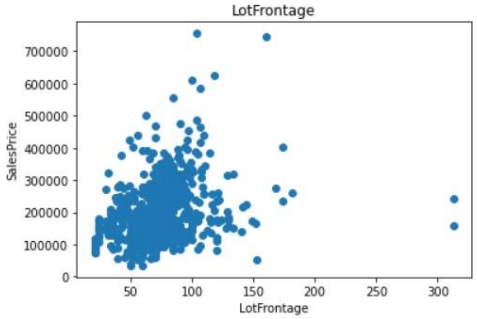
-> ExterQual: Evaluates the quality of the material on the exterior-: Houses with exterior material of excellent quality have highest selling price followed by that of gd(good) quality

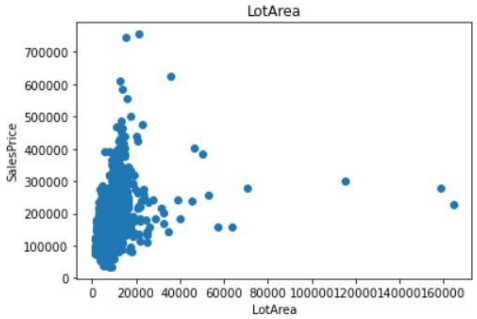
-> KitchenQual: Kitchen quality-: Houses with Ex (Excellent) kitchen quality have higher sale price while that with Fa (Fair) kitchen quality of lower selling price

Plotting for continuous data

```
#Visualising sale price wrt to each continuous feature
for i in cont:
    plt.scatter(df_train[i],df_train['SalePrice'])
    plt.xlabel(i)
    plt.ylabel('SalesPrice')
    plt.title(i)
    plt.show()
```







Observations:

-> LotFrontage: Linear feet of street connected to property-: Lot frontage does not impact much on sale price since houses with different sale price are having same Lot frontage area

-> LotArea: Lot size in square feet-: LotArea doesn't affect sale price of the houses much, as can be seen different sale price are available within the Lot area range of 0 to 20000.In fact some houses where Lot Area is very large have moderate sale price

-> OverallQual: Rates the overall material and finish of the house-: Overall quality is directly proportional to the sale price of houses

-> YearBuilt: & YearRemodAdd: Houses which are build latest have high sale price in comparison to those build in early years. Similar is the case with remodelling date

-> BsmtFinSF1: Type 1 finished square feet-: Total sqft of basement area is directly proportional to sale price Houses with higher number of full bathrooms seems having high sale price

-> Kitchen: Kitchens above grade-: houses with 1 kitchen above ground have high sale price in comparison to those having 2 kitchens

-> Fireplaces: Number of fireplaces-: Houses with 1 and 2 fireplaces have higher prices in comparison to houses having 0 or 3 fireplaces

-> Wood deck, Enclosed porch, three season porch, screen porch, pool area, Miscval do not have impact on sale price

CONCLUSION

Key Findings and Conclusions of the Study

-> After getting an insight of this dataset, we were able to understand that the Housing prices are done on basis of different features.

-> First, we loaded the train dataset and did the EDA process and other pre-processing techniques like skewness check and removal, handling the outliers present, filling the missing data, visualizing the distribution of data, etc.

-> Then we did the model training, building the model and finding out the best model on the basis of different metrices scores we got like Mean Absolute Error, Mean squared Error, Root Mean Squared Error, etc.

-> We got Lasso Regressor as the best algorithm among all as it gave more r2_score and cross_val_score. Then for finding out the best parameter and improving the scores, we performed Hyperparameter Tuning.

-> As the scores were not increased, we also tried using Ensemble Techniques like RandomForestRegressor, AdaBoostRegressor and GradientBoostingRegressor algorithms for boosting up our scores. Finally, we concluded that GradientBoostingRegressor was the best performing algorithm, although there were more errors in it and it had less RMSE compared to other algorithms. It gave an r2_score of 92.69 and cross_val_score of 86.97 which is the highest scores among all.

-> We saved the model in a pickle with a filename in order to use whenever we require.

-> We predicted the values obtained and saved it separately in a csv file.

-> Then we used the test dataset and performed all the pre-processing pipeline methods to it.

-> After treating skewness, we loaded the saved model that we obtained and did the predictions over the test data and then saving the predictions separately in a csv file.

-> From this project, we learnt that how to handle train and test data separately and how to predict the values from them. This will be useful while we are working in a real-time case study as we can get any new data from the client we work on and we can proceed our analysis by loading the best model we obtained and start working on the analysis of the new data we have.

-> The final result will be the predictions we get from the new data and saving it separately.

-> Overall, we can say that this dataset is good for predicting the Housing prices using regression analysis and GradientBoostingRegressor is the best working algorithm model we obtained.

-> We can improve the data by adding more features that are positively correlated with the target variable, having less outliers, normally distributed values, etc.

Learning Outcomes of the Study in respect of Data Science

1. Price Prediction modeling – This allows predicting the prices of houses & how they are varying in nature considering the different factors affecting the prices in the real time scenarios.

2. Prediction of Sale Price – This helps to predict the future revenues based on inputs from the past and different types of factors related to real estate & property related cases. This is best done using predictive data analytics to calculate the future values of houses. This helps in segregating houses, identifying the ones with high future value, and investing more resources on them.

3. Deployment of ML models – The Machine learning models can also predict the houses depending upon the needs of the buyers and recommend them, so customers can make final decisions as per the needs.

4. We see how to deal with outliers when all the rows have at least one value Z>3.

5. To do a visualisation when data has high standard deviation and no Classification

6. Ways to select features and to do hyperparameter tuning efficiently

7. Ways of removing skewness and what are the best methods still not versatile when it comes to data with 0 value

8. How to make a model using a pipeline.

Limitations of this work and Scope forFuture Work

1. The biggest limitation I observed was that not all categories of a particular feature were available in the training data. So, if there were new category in the test data the model would not be able to identify that.

Example: MSZoning has 8
categories A Agriculture
C Commercial
FV Floating Village
Residential I Industrial
RH Residential High
Density RL Residential
Low Density
RP Residential Low-Density
Park RM Residential Medium
Density

2. However in the Training dataset only 5 categories are present, what happen if other 3 categories will present in test data in future. It would be difficult for machine to identify and predict.

3. The high skewness of data reduces the effectivity

4. Many features have NaN values more than 50%, and imputation of them can decrease the effectiveness. And dropping them had the loss of data.

5. we can increase the efficiency of a model by selecting a better method to remove outliers and skewness also how to make the search of perfect model in a way that if we want to change some parameters in model then we don't have to run all the model again