Housing Price Predictions

Submitted by:

Purva Sonsare

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my SME (Subject Matter Expert) **Mohd. Kashif** as well as **Flip Robo Technologies** who gave me the opportunity to do this project on **House Price Prediction**, which also helped me in doing lots of research wherein I came to know about so many new things especially the data collection part. Also, I have utilized a few external resources that helped me to complete the project. I ensured that I learn from the samples and modify things according to my project requirement. All the external resources that were used in creating this project are listed below:

1) https://www.google.com/

2) https://www.youtube.com/

3) https://scikit-learn.org/stable/user_guide.html

4) https://github.com/

5) https://www.kaggle.com/

6) https://medium.com/

7) https://towardsdatascience.com/

8) https://www.analyticsvidhya.com/

INTRODUCTION

Thousands of houses are sold every day. There are some questions every buyer asks himself like: What is the actual price that this house deserves? Am I paying a fair price? Also Is it the location? Is it the overall quality of the house? Is it the size? Could it be sold at a good price in future? All these questions come in to our mind when we decide to purchase a house. In this study, a machine learning model is proposed to predict a house price based on data related to the house (its size, the year it was built in, etc.). During the development and evaluation of our model, we will show the code used for each step followed by its output. This will facilitate the reproducibility of our work.

Business Problem Framing

Housing and real estate markets are important contributors to a country's economy. It is a huge market with many firms operating in it. Data Science may assist countries improve their total income, profitability, and marketing strategies by solving challenges in this sector. Machine learning techniques may be utilized to help this housing company achieve its commercial objectives. Our problem is with a housing company based in the United States called Surprise Housing, which wants to enter the Australian market. The company intends to use Data Analytics to buy houses at a discount from their true value and resell them at a profit. The company has compiled a dataset based on house sales in Australia. The firm is looking at potential properties to purchase residences in order to enter the market. We will create a model utilizing Machine Learning to estimate the actual worth of potential properties, which will assist the firm in deciding whether or not to invest in real estate.

Conceptual Background of the Domain Problem

Housing price trends reflect the current economic situation and are a source of concern for both buyers and sellers. House prices are affected by a variety of factors, including the number of bedrooms and bathrooms.

The cost of a house is also affected by its location. A house with easy access to roads, schools, malls, and employment possibilities will be more expensive than a house without such connectivity. Predicting home prices manually is a tough process that is typically not very accurate, which is why various methods for house price prediction have been developed.

Review of Literature

The world is evolving away from manual processes and toward automated ones. The goal of our project is to alleviate the customer's issues. In the current circumstance, the consumer goes to a real estate agent so that he or she may recommend acceptable showplaces for his assets. However, the above strategy is risky because the agent may forecast incorrect rates to the customer, resulting in a loss of the customer's investment. This manual approach, which is currently being utilized in the market, is out of date and dangerous. To overcome the disadvantage, an improved and automated system is required. Machine learning is a type of artificial intelligence that consists of accessible computers with the capability of being learned without being explicitly programmed. Machine learning is focused in the expansions of computer programs that are capable of modifying when exposed to new-fangled data. Machine learning algorithms are divided into three categories: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is a type of learning in which we instructor train the machine using well-labeled data, which implies that part of the data has already been tagged with the correct answer. Following that, the computer is given a fresh collection of samples so that the supervised learning algorithm may analyze the training data and create a proper result from labeled data

Motivation for the Problem Undertaken

The increasing unaffordability of housing has become a serious problem for governments all around the world. To obtain a better grasp of the commercialized housing market we are presently dealing with, we wish to identify the main influencing elements of home prices. Aside from the more apparent driving causes, such as inflation and land scarcity, there are a number other variables to consider. Our objective is to explore the key variables that influence house prices and offer accurate predictions. We utilize 80 explanatory variables that cover nearly every element of

Australian residential dwellings. Methods from both statistics and regression models, as well as machine learning models, are used and evaluated in order to better predict the ultimate price of each dwelling. The algorithm predicts home prices based on similar comparable of people's ideal homes, allowing both buyers and sellers to better negotiate home pricing based on market trends.

Hardware and Software Requirements

The hardware utilized for this project is a laptop with high-end specifications and a steady internet connection. When it came to the software, I utilized anaconda navigator and Jupyter notebook to conduct my Python programming and analysis. Microsoft Excel is required to use an excel file. In Jupyter notebook, I utilized several Python libraries to complete this project, which I have listed below with appropriate substantiation:

1. Pandas - It is a library that is used to read data, visualize it, and analyses it.
2. NumPy- utilized for dealing with arrays and different mathematical methods.
3. Seaborn- a visualization tool for plotting many sorts of plots.
4. Matplotlib- It provides an object-oriented API for embedding plots into applications. Analytical Problem Framing Data sources and their formats We are provided two CSV files comprising train and test datasets of house.

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. **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.

6. **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.
7. **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.
8. **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

A US-based housing company named Surprise Housing has collected the dataset from the sale of houses in Australia and the data is provided by Flip Robo Company and it is in csv format. There are 2 data sets:

1. Train dataset

2. Test dataset

- Train dataset will be used for training the machine learning models. The dataset contains 1168 rows and 81 columns, out of 81 columns, 80 are independent variables and remaining 1 is dependent variable (SalePrice).
- Test dataset contains all the independent variables, but not the target variable. We will apply the trained model to predict the target variable for the test data. The dataset contains 292 rows and 80 columns.
- The dataset contains both numerical and categorical data. Numerical data contains both continuous and discrete variables and categorical data contains both nominal and ordinal variables.

- I can concatenate both train and test data, but this may cause data leakage so I decided to process both the data separately

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

```
]: # To display maximum rows and columns in the dataset
   pd.set_option("display.max_columns",None)
   pd.set_option("display.max_rows",None)

]: # Loading the dataset from the given file
   train_df =pd.read_csv('train.csv')
   train_df.head(15)
```

]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Co
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	
5	1197	60	RL	58.0	14054	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	
6	561	20	RL	NaN	11341	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Sawyer	Norm	
7	1041	20	RL	88.0	13125	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	Sawyer	Norm	
8	503	20	RL	70.0	9170	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	Edwards	Feedr	
9	576	50	RL	80.0	8480	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	
10	449	50	RM	50.0	8600	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl	IDOTRR	Norm	
11	833	60	RL	44.0	9548	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	CollgCr	Norm	
12	277	20	RL	129.0	9196	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Mitchel	Norm	
13	84	20	RL	80.0	8892	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	NAmes	Norm	
14	888	50	RL	59.0	16466	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Edwards	Norm	

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 datato 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 stepin 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 thatwe pre- process our data before feeding it into our model.

- Firstly, I have imported the necessary libraries and imported both train and test datasets which were in csv format. And process both datasets one by one.

- I have done some statistical analysis like checking shape, nunique, column names, data types of the features, info about the features, value counts etc. for both train and test data.

- While looking into the value count function I found some of the columns having more than 85% of zero values so, I dropped those columns from both the datasets as they might create skewness which will impact my model.

- I have dropped the columns “Id” and “Utilities” from both the datasets. Since Id is the unique identifier which contains unique value throughout the data also all the entries in Utilities column were unique. They had no significance impact on the prediction.

Checking Unique values and Value Counts

```
[12]: train_df.nunique().sort_values()
```

```
[12]: Utilities          1
      Street             2
      CentralAir         2
      PavedDrive         3
      HalfBath           3
      LandSlope          3
      BsmtHalfBath       3
      GarageFinish       3
      Fireplaces         4
      FullBath           4
      KitchenAbvGr       4
      KitchenQual        4
      MasVnrType         4
      BsmtFullBath       4
      LandContour        4
      LotShape           4
      BsmtQual           4
      BsmtCond           4
      BsmtExposure       4
      ExterQual          4
      HeatingQC          5
      Electrical         5
      ExterCond          5
      YrSold             5
      FireplaceQu        5
      BldgType           5
      GarageCars         5
      LotConfig          5
      GarageQual         5
      GarageCond         5
      MSZoning           5
```

We can see in above table that Id have all unique values and utilites have all common value so we can drop these column.

```
In [13]: # checking Value counts
         for col in train_df:
             print(col)
             print(train_df[col].value_counts())
             print('\n')
```

```
Id
127     1
1391    1
1389    1
448     1
1179    1
821     1
178     1
1076    1
60      1
733     1
739     1
1192    1
263     1
582     1
392     1
327     1
```

```
]: train_df.drop(['Alley','MiscFeature','Fence','PoolQC'], axis=1, inplace= True)
```

```
]: train_df.shape
```

```
]: (1168, 77)
```

Now we have 1168 rows and 77 columns

```
: train_df.drop(['Id','Utilities','PoolArea'], axis=1, inplace= True)

: train_df.shape

: (1168, 74)
```

Handling missing data

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

garage=['GarageType','GarageFinish','GarageQual','GarageCond']
#Blank/missing values means No_Garage for all i in garage
for i in garage:
    train_df[i].fillna('No_Garage',inplace=True)
    print(train_df[i].value_counts())

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


#Blank/missing values means No_Fireplace. Let's replace missing data with 'No_Fireplace'
train_df['FireplaceQu'].fillna('No_Fireplace',inplace=True)
print(train_df['FireplaceQu'].value_counts())

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

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

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

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 sampledata. 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 themean, count, max, standard deviation and IQR values of the dataset in a simple understandable way.

```
train_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
MSSubClass	1168.0	56.767979	41.940650	20.0	20.00	50.0	70.00	190.0
LotFrontage	1168.0	70.807363	22.440317	21.0	60.00	70.0	79.25	313.0
LotArea	1168.0	10484.749144	8957.442311	1300.0	7621.50	9522.5	11515.50	164660.0
OverallQual	1168.0	6.104452	1.390153	1.0	5.00	6.0	7.00	10.0
OverallCond	1168.0	5.595890	1.124343	1.0	5.00	5.0	6.00	9.0
YearBuilt	1168.0	1970.930651	30.145255	1875.0	1954.00	1972.0	2000.00	2010.0
YearRemodAdd	1168.0	1984.758562	20.785185	1950.0	1966.00	1993.0	2004.00	2010.0
MasVnrArea	1168.0	101.696918	182.218483	0.0	0.00	0.0	160.00	1600.0
BsmtFinSF1	1168.0	444.726027	462.664785	0.0	0.00	385.5	714.50	5644.0
BsmtFinSF2	1168.0	46.647260	163.520016	0.0	0.00	0.0	0.00	1474.0
BsmtUnfSF	1168.0	569.721747	449.375525	0.0	216.00	474.0	816.00	2336.0
TotalBsmtSF	1168.0	1061.095034	442.272249	0.0	799.00	1005.5	1291.50	6110.0
1stFlrSF	1168.0	1169.860445	391.161983	334.0	892.00	1096.5	1392.00	4692.0
2ndFlrSF	1168.0	348.826199	439.696370	0.0	0.00	0.0	729.00	2065.0
LowQualFinSF	1168.0	6.380137	50.892844	0.0	0.00	0.0	0.00	572.0
GrLivArea	1168.0	1525.066781	528.042957	334.0	1143.25	1468.5	1795.00	5642.0
BsmtFullBath	1168.0	0.425514	0.521615	0.0	0.00	0.0	1.00	3.0
BsmtHalfBath	1168.0	0.055651	0.236699	0.0	0.00	0.0	0.00	2.0
FullBath	1168.0	1.562500	0.551882	0.0	1.00	2.0	2.00	3.0
HalfBath	1168.0	0.388699	0.504929	0.0	0.00	0.0	1.00	2.0
BedroomAbvGr	1168.0	2.884418	0.817229	0.0	2.00	3.0	3.00	8.0
KitchenAbvGr	1168.0	1.045377	0.216292	0.0	1.00	1.0	1.00	3.0
TotRmsAbvGrd	1168.0	6.542808	1.598484	2.0	5.00	6.0	7.00	14.0
Fireplaces	1168.0	0.617295	0.650575	0.0	0.00	1.0	1.00	3.0
GarageYrBlt	1168.0	1976.287671	26.376864	1875.0	1959.00	1978.0	2001.00	2010.0
GarageCars	1168.0	1.776541	0.745554	0.0	1.00	2.0	2.00	4.0
GarageArea	1168.0	476.860445	214.466769	0.0	338.00	480.0	576.00	1418.0
WoodDeckSF	1168.0	96.206336	126.158988	0.0	0.00	0.0	171.00	857.0
OpenPorchSF	1168.0	46.559932	66.381023	0.0	0.00	24.0	70.00	547.0
EnclosedPorch	1168.0	23.015411	63.191089	0.0	0.00	0.0	0.00	552.0
3SsnPorch	1168.0	3.639555	29.088867	0.0	0.00	0.0	0.00	508.0
ScreenPorch	1168.0	15.051370	55.080816	0.0	0.00	0.0	0.00	480.0
MiscVal	1168.0	47.315068	543.264432	0.0	0.00	0.0	0.00	15500.0
MoSold	1168.0	6.344178	2.686352	1.0	5.00	6.0	8.00	12.0
YrSold	1168.0	2007.804795	1.329738	2006.0	2007.00	2008.0	2009.00	2010.0
SalePrice	1168.0	181477.005993	79105.586863	34900.0	130375.00	163995.0	215000.00	755000.0

Observations

- Count column shows equal value stating no missing values in our data.
- Right skewed columns where mean>median are MSSubClass, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF,1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea etc
- Left skewed columns where mean<median are YearBuilt, YearRemodAdd, FullBath, BedroomAbvGr etc
- Some columns show presence of outliers ie. the difference

between 75% and max value is very large example TotalBsmtSF, 2ndFlrSF, MSSubClass, LotFrontage, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtHalfBath, BedroomAbvGr, ToRmsAbvGrd, GarageArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, MiscVal, SalePrice.

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 isdecreasing.

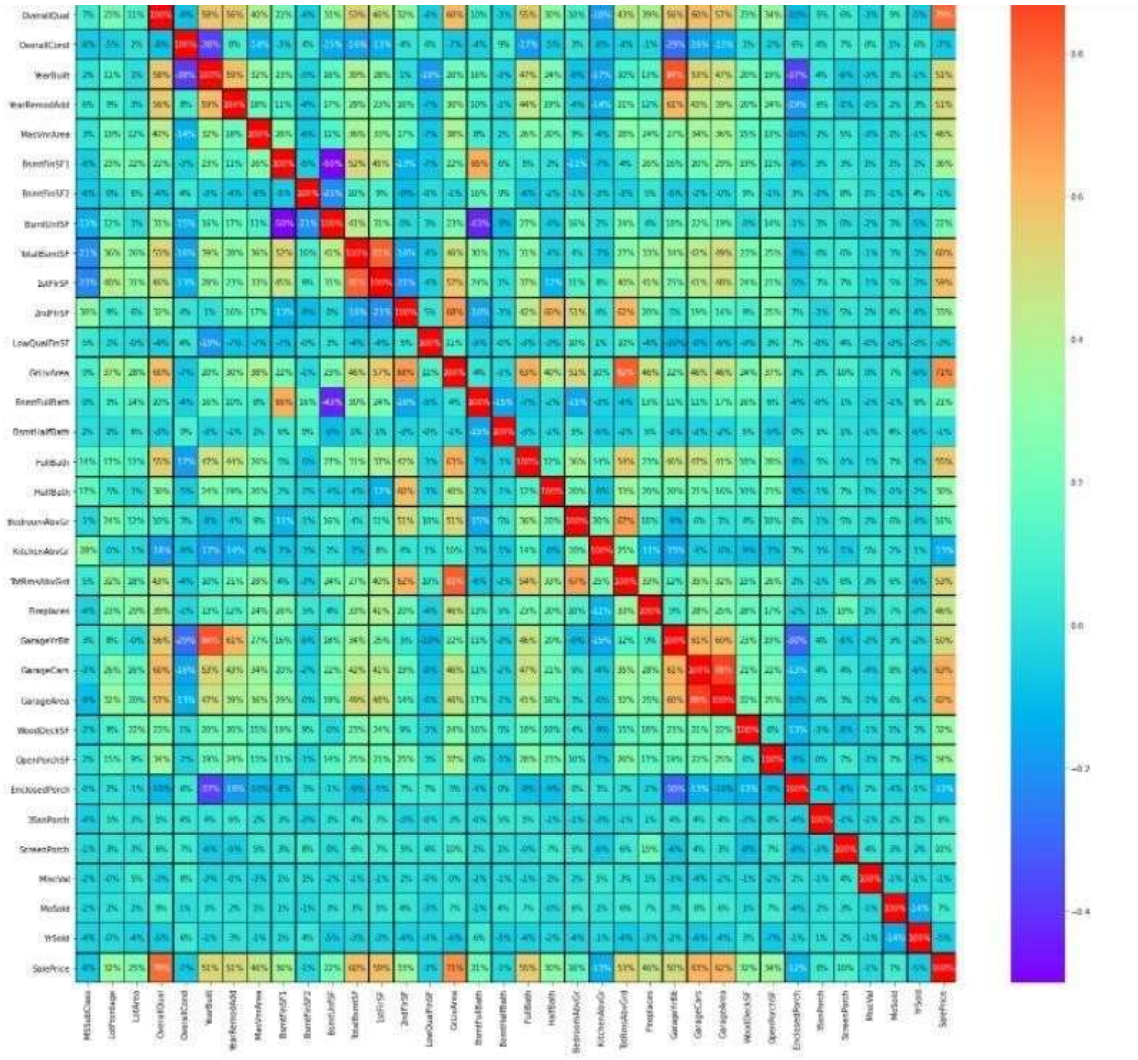
```
#Checking correlation of the dataset
corr=df_train.corr()  #corr() function provides the correlation value of each column
corr
```

	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	...	0.19
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 givendataset.

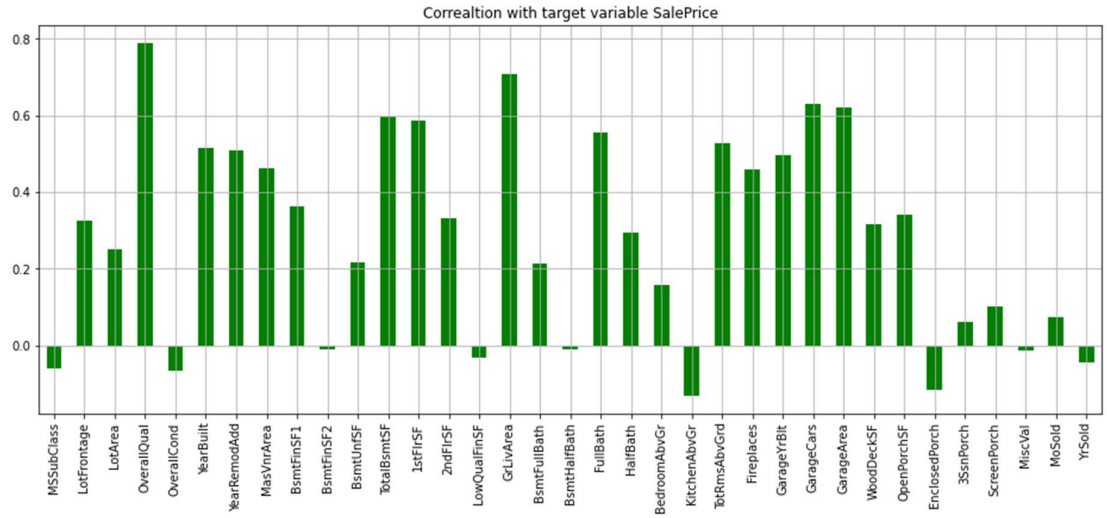


- Observations:
  Highly correlated columns are 1stflrSF, YearBuilt, YearRemodAdd, TotalBsmtSF, GrLivArea, GarageCars, GarageArea, TotRmsAbvGrd.
- Very less correlated are MssubClass, BsmtFinSF2, OverallCond, BsmtHalfBath, MiscVal, MoSold, YrSold.
- We observe multicollinearity in between columns, so 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))
train_df.drop('SalePrice', axis=1).corrwith(train_df['SalePrice']).plot(kind='bar',grid=True, color= 'g')
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 correlatedwith target column
- 'MSSubClass','OverallCond','OverallCond','LowQualFinSF','BsmtHalf Bath','YrSold', 'MiscVal', 'MoSold', '3SsnPorch' are least correlated with the target column

Encoding the Categorical

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
list1=['MSZoning','Street','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','SaleType','SaleCondition',]
for val in list1:
    train_df[val]=le.fit_transform(train_df[val].astype(str))
```

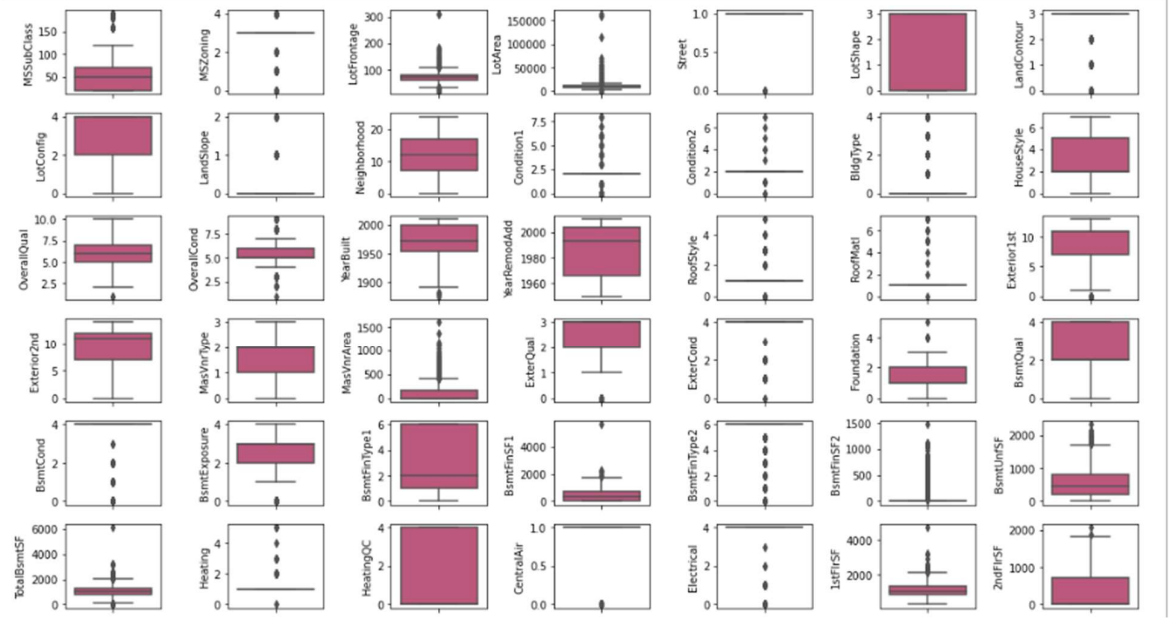
```
train_df.head()
```

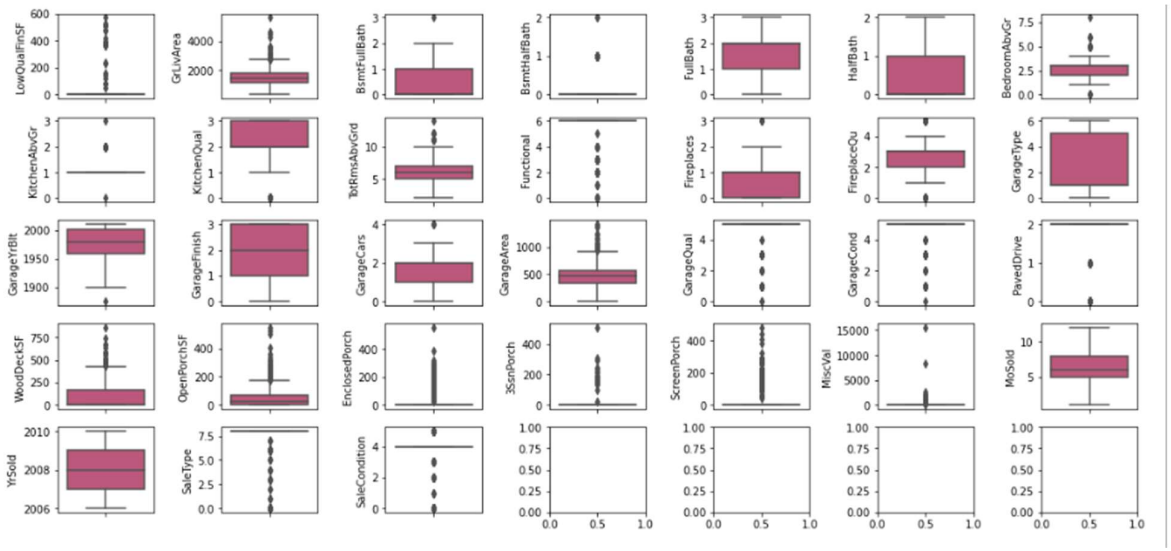
	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	Hou
0	120	3	70.0	4928	1	0	3	4	0	13	2	2	4	
1	20	3	95.0	15865	1	0	3	4	1	12	2	2	0	
2	60	3	92.0	9920	1	0	3	1	0	15	2	2	0	
3	20	3	105.0	11751	1	0	3	4	0	14	2	2	0	
4	20	3	70.0	16635	1	0	3	2	0	14	2	2	0	

Hence encoding of categorical column is done as we can see above.

Checking and Removing Outliers

```
fig, ax = plt.subplots(ncols=7, nrows=11, figsize=(15,15))
index = 0
features = train_df.drop("SalePrice", axis=1)
ax = ax.flatten()
for col, value in features.items():
    sns.boxplot(y=col, data=train_df, ax=ax[index], palette='plasma')
    index += 1
plt.tight_layout(pad=0.1, w_pad=0.5, h_pad=1.0)
plt.show()
```





Removing Outliers

```
z = np.array(abs(zscore(train_df)))
threshold = 6
print(np.where(z>6))
df1_new=train_df[(z<6).all(axis=1)]
df1_new.head(10)
```

```
(array([   1,   20,   22,   32,   32,   33,   51,   51,   63,   66,   95,
          98,  103,  103,  103,  113,  113,  119,  119,  141,  142,  206,
         211,  211,  226,  228,  253,  255,  272,  279,  310,  320,  333,
         356,  361,  363,  370,  396,  418,  429,  441,  441,  443,  463,
         478,  481,  488,  493,  500,  500,  507,  510,  532,  534,  544,
         553,  562,  563,  572,  591,  592,  592,  592,  592,  592,  614,
         614,  637,  639,  652,  686,  689,  689,  691,  691,  699,  713,
         716,  721,  722,  759,  769,  794,  821,  824,  833,  833,  834,
         839,  846,  888,  897,  899,  911,  935,  944,  952,  952,  956,
         961, 1008, 1023, 1038, 1038, 1046, 1047, 1053, 1053, 1080, 1082,
        1094, 1098, 1108, 1120, 1120, 1123, 1123, 1123, 1139, 1142],
       dtype=int64), array([19, 66, 33, 11, 68, 42, 64, 67, 66, 42, 33, 66, 19, 42, 67,  3,  8,
       19, 63, 73, 19, 66,  8, 19, 66, 42, 66, 36, 49, 10, 19, 42, 42, 19,
       11, 11, 36, 36, 10, 36,  8, 19, 19, 10, 66, 19, 66, 10,  8, 19, 33,
       36, 19, 42, 45, 11, 33, 68, 67, 33,  2, 31, 35, 40, 43, 42, 48, 66,
       19, 11, 33,  3, 19, 19, 73, 36, 64,  8, 42, 66,  4, 42, 33, 19, 66,
        8, 33, 36, 23,  8,  8, 11, 36, 33, 36, 11, 19, 45, 64,  8, 67, 42,
        3,  8, 36, 23,  2, 19, 42,  8,  4, 19, 42, 42, 65,  3,  4,  8,  4,
       19], dtype=int64))
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	Hc
0	120	3	70.0	4928	1	0	3	4	0	13	2	2	4	
2	60	3	92.0	9920	1	0	3	1	0	15	2	2	0	
3	20	3	105.0	11751	1	0	3	4	0	14	2	2	0	
4	20	3	70.0	16635	1	0	3	2	0	14	2	2	0	
5	60	3	58.0	14054	1	0	3	4	0	8	2	2	0	
6	20	3	70.0	11341	1	0	3	4	0	19	2	2	0	
7	20	3	88.0	13125	1	3	3	0	0	19	2	2	0	

```
print("Shape of old data",train_df.shape)
print("Shape of new data",df1_new.shape)

dataloss= ((1168-1071)/1168)*100
print("PErcentage data loss is", dataloss)
```

```
Shape of old data (1168, 74)
Shape of new data (1071, 74)
PErcentage data loss is 8.304794520547945
```

We can use threshold value as 3 but data loss is then 58% due to wide spread of data.So chnaged the threshold value to 6. and 8% dataloss is affordable.

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 thedataset. In statistics, an outlier is an observation point that is distant from other

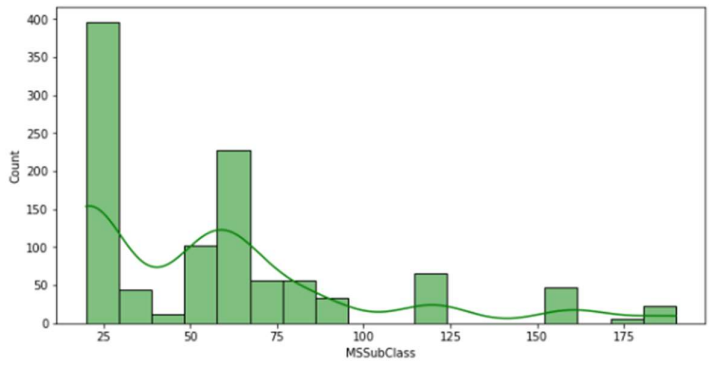
observations.

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.

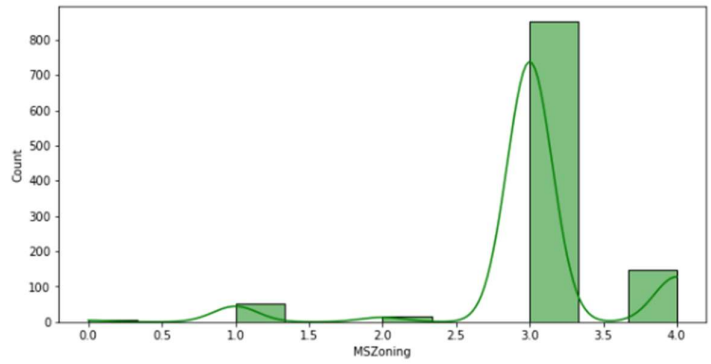
Checking skewness and plotting the distribution plot

```
from scipy.stats import skew
for i in train_df:
    print(i, "=", skew(train_df[i]))
    plt.figure(figsize=(10,5))
    sns.histplot(train_df[i], kde = True, color= 'green')
    plt.show()
```

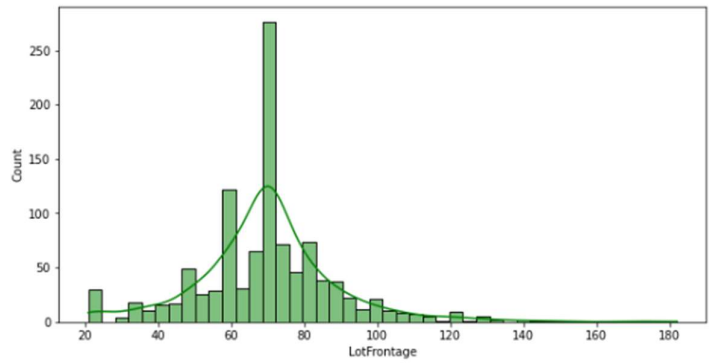
```
MSSubClass = 1.3972981139025331
```



MSZoning = -1.6781665662758536



LotFrontage = 0.7037262364989109




```
train_df.skew().sort_values()
```

```
Condition2      -16.294085
Functional       -4.101835
GarageCond       -3.797630
CentralAir       -3.791556
SaleType         -3.630603
GarageQual       -3.508460
PavedDrive       -3.423921
LandContour      -3.364845
BsmtFinType2     -3.324850
Electrical       -3.324222
BsmtCond         -2.908419
SaleCondition    -2.811667
ExterCond        -2.618932
ExterQual        -1.841365
MSZoning         -1.680521
KitchenQual      -1.441584
LotConfig        -1.214140
BsmtExposure     -1.144384
GarageYrBlt      -0.686663
LotShape         -0.665338
GarageFinish     -0.658067
YearBuilt        -0.613278
Exterior1st      -0.603668
Exterior2nd      -0.587603
YearRemodAdd     -0.521310
BsmtQual         -0.469873
GarageCars       -0.293708
MasVnrType       -0.075736
Foundation       -0.008875
Street            0.000000
Neighborhood      0.044540
FullBath          0.077004
BedroomAbvGr      0.089559
BsmtFinType1      0.096876
YrSold            0.111969
GarageArea        0.154427
OverallQual       0.176144
MoSold            0.222311
HouseStyle        0.263848
FireplaceQu       0.278026
```

Skewness refers to distortion or asymmetry in a symmetricalbell 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.

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.

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

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.

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.

Removing Skewness

```
from sklearn.preprocessing import PowerTransformer
power_t = PowerTransformer(method = 'yeo-johnson')
x = pd.DataFrame(power_t.fit_transform(x), columns=x.columns)
x.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	H
0	1.424826	-0.149122	0.065438	-1.357353	0.0	-1.383877	0.311987	0.595827	-0.202031	0.161960	0.085741	0.061228	2.311918	
1	0.435262	-0.149122	1.113237	0.206682	0.0	-1.383877	0.311987	-1.639643	-0.202031	0.492230	0.085741	0.061228	-0.438774	
2	-1.123839	-0.149122	1.680308	0.634195	0.0	-1.383877	0.311987	0.595827	-0.202031	0.327529	0.085741	0.061228	-0.438774	
3	-1.123839	-0.149122	0.065438	1.554768	0.0	-1.383877	0.311987	-1.477751	-0.202031	0.327529	0.085741	0.061228	-0.438774	
4	0.435262	-0.149122	-0.582482	1.097982	0.0	-1.383877	0.311987	0.595827	-0.202031	-0.680664	0.085741	0.061228	-0.438774	

```
x.skew().sort_values()        # checking skewness after treating
```

```
Condition2      -16.294085
Heating          -7.092503
CentralAir       -3.791556
Functional       -3.491367
Electrical       -3.214496
PavedDrive       -3.164754
GarageCond       -3.047291
LandContour      -2.897333
GarageQual       -2.877134
BsmtCond         -2.621088
ExterCond        -2.390848
BsmtFinType2     -2.219322
SaleType         -1.991210
LotConfig        -1.087959
LotShape         -0.632861
ExterQual        -0.552812
Condition1       -0.399916
KitchenQual      -0.309183
OverallCond      -0.285075
GarageFinish     -0.222245
```

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, visualization and analysis of data.

2. NumPy- used for working with array and various

mathematical techniques.

3. Seaborn- visualization tool for plotting different types of plots.

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 sendingit 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

a) **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.

b) **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.

c) **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 ofmulticollinearity 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.

d) **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 group.

To eliminate the limitations found in lasso, the elastic net includes a quadratic expression ($||\beta||2$) 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 taskswith 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 themin 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:

a) **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.

b) **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.

c) **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, orbinary 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 timeversus initial time, and one technique of measurement versusan alternative technique of measurement.

The MAE measures the average magnitude of the errors in aset 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 individualdifferences 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 overfitting 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 ofthe 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.
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.
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.
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.
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.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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.
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.
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.
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.
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.

1095 rows × 68 columns

Using PCA

An important machine learning method for dimensionality reduction is called Principal Component Analysis. It is a methodthat 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
```

Run and evaluate selected models

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

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

```
#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, thenit 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,te
st_size=.20)
```

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

Training the data and finding the best Model

```
lg  = LinearRegression()
svr = SVR(kernel='rbf')
dtr = DecisionTreeRegressor()
ridr= Ridge(alpha=1e-2, normalize=True)
las = Lasso(alpha=1e-2, normalize=True, max_iter=100)
rfr = RandomForestRegressor()
gbr = GradientBoostingRegressor(loss='quantile', n_estimators=200, max_depth=5)
abr = AdaBoostRegressor(n_estimators=300, learning_rate=1.05, random_state=42)
etr = ExtraTreesRegressor(n_estimators=200, max_features='sqrt', n_jobs=6)


model=[lg,svr,dtr,ridr,las,rfr,gbr,abr,etr]

for m in model:
    print('\n')
    print('*****************',m,'******************')
    print('\n')
    m.fit(x_train,y_train)
    m.score(x_train,y_train)

    pred_train=m.predict(x_train)
    pred_test=m.predict(x_test)

    # calculating scores
    train_score=r2_score(y_train,pred_train)
    test_score=r2_score(y_test,pred_test)
    r2 = r2_score(y_test, pred_test)*100

    #Calculating cross validation score

    for i in range(2,8):
        cvs=cross_val_score(m,x,y,cv=i)

    a = max(cvs)
    cv_score=(a.mean())*100    # taking max value of cvs and calculating mean of it
# taking max value because I am assuming that r2 score of all models to be approx 100 %

   # final result comparing r2 and cross validation
   result = r2 - cv_score


   print('The training accuracy of is', train_score)
   print('The test accuracy of is', test_score)
   print("R2 Score is:", r2)
   print("Cross Validation Scores",cvs)
   print("Maximum of cvs:" , a)
   print("Cross Validation Score:", cv_score)
   print("R2 Score - Cross Validation Score is", result)
   print("Error Calculations:")
   print("Mean absolute error :",mean_absolute_error(y_test,pred_test))
   print("Mean squared error : ",mean_squared_error(y_test, pred_test))
   print("Root mean squared error:", np.sqrt(mean_squared_error(y_test,pred_test)))

   plt.figure(figsize=(8,6))
   plt.scatter(x=y_test,y=pred_test,color='r')
   plt.plot(y_test,y_test,color='b')
   plt.xlabel('Actual Sale Price',fontsize=14)
   plt.ylabel('Predicted Sale Price',fontsize=14)
   plt.title(m,fontsize=18)
   plt.show()
```

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:

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 sameI 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:

```
Performance_table=pd.DataFrame({'Model':['LR','SVR','DTR','ridge','lasso','RFR','GBR','ABR','ETR'],
            'R2 Score': [90.75,-10.55,73.84,90.67,90.75,87.19,78.45,83.84,71.47],
            'CV Score': [88.90,-1.31,77.26,89.00,88.89,90.15,81.79,85.73,73.40],
            'Result' :[1.85,-9.24,-3.42,1.66,1.85,-2.96,-3.34,-1.89,-1.92],
           'RMSE':[23512,81322,39553,23620,23513.01,27672,35901,31088,41304]})
```

```
Performance_table
```

	Model	R2 Score	CV Score	Result	RMSE
0	LR	90.75	88.90	1.85	23512.00
1	SVR	-10.55	-1.31	-9.24	81322.00
2	DTR	73.84	77.26	-3.42	39553.00
3	ridge	90.67	89.00	1.66	23620.00
4	lasso	90.75	88.89	1.85	23513.01
5	RFR	87.19	90.15	-2.96	27672.00
6	GBR	78.45	81.79	-3.34	35901.00
7	ABR	83.84	85.73	-1.89	31088.00
8	ETR	71.47	73.40	-1.92	41304.00

Lasso and ridge are giving good results. Choosing Ridge

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 parametersand also to increase our scores by using Hyperparameter Tuning and we will be using GridSearchCV method.

Cross Validation:

Cross-validation helps to find out the over fitting and underfitting 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 outas a holdout set for validation and everything else is used for training data. This way we will get the first estimate of the modelquality 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 processto get the remaining estimate of the model quality.

R2 Score:

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

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 riskfunction, corresponding to the expected value of the squared error loss. RMSE is the Root Mean Squared Error.

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.

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 noteis 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.

Hypertuning the model

```
#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': 'svd'}
0.8343754064857425
```

```
#Using the best parameters obtained
rd=Ridge(alpha=1, random_state=42, solver='svd')
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=6,scoring='r2').mean()*100)
```

```
Final r2_score after tuning is:  90.75425387390675
Cross validation score:  85.88868762624138
```

Therefore our final model is giving 90.75 % accurate results.

Final the model

```
#Using the best parameters obtained
rd=Ridge(alpha=1, random_state=42, solver='svd')
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=6,scoring='r2').mean()*100)
```

```
Final r2_score after tuning is:  90.75425387390675
Cross validation score:  85.88868762624138
```

Therefore our final model is giving 90.75 % accurate results.

Saving the Model

```
import joblib
joblib.dump(rd,'HousingPrice_train.csv')
```

```
['HousingPrice_train.csv']
```

Hence we have saved our final model

Using the test dataset and doing pre-processing

Test Dataset

```
3]: # Loading the tesy dataset from the given file
    test_df =pd.read_csv('test.csv')
    test_df.head(15)
```

3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Co
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	Corner	Gtl	StoneBr	Norm	
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	StoneBr	Norm	
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl	Crawfor	Norm	
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	Somerst	Feedr	
5	650	180	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	MeadowV	Norm	
6	1453	180	RM	35.0	3675	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Edwards	Norm	
7	152	20	RL	107.0	13891	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NridgHt	Norm	
8	427	80	RL	NaN	12800	Pave	NaN	Reg	Low	AllPub	Inside	Mod	SawyerW	Norm	
9	776	120	RM	32.0	4500	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Mitchel	Norm	
10	30	30	RM	60.0	6324	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	BrkSide	Feedr	
11	1425	20	RL	NaN	9503	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	NAmes	Norm	
12	423	20	RL	100.0	21750	Pave	NaN	Reg	HLS	AllPub	Inside	Mod	Mitchel	Artery	
13	1185	20	RL	50.0	35133	Grvl	NaN	Reg	Lvl	AllPub	Inside	Mod	Timber	Norm	
14	775	20	RL	110.0	14226	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	NridgHt	Norm	

```
4]: test_df.shape
```

```
4]: (292, 80)
```

Test dataset has 292 rows and 80 columns

```
5]: test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 292 entries, 0 to 291
```

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=joblib.load(open('HousingPrice_train.csv','rb'))
```

```
fitted_model
```

```
Ridge(alpha=1, random_state=42, solver='svd')
```

```
test_preds=fitted_model.predict(x)
```

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

	SalePrice
74	137617.736354
75	108307.412575
76	203537.009696
77	52877.671009
78	308147.605547
79	156721.853360
80	197735.687780
81	240316.444256
82	165847.668633
83	108278.843039
84	156757.514524
85	196990.376531
86	204694.427738

Joining the final results with testdata

```
Test_final1=pd.concat([x,test_preds],axis=1)
```

```
Test_final1.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType
0	6.399074	-2.742431	2.148577	-0.098552	0.268267	1.612956	-0.554345	1.192509	-0.640498	-0.451846	0.087070	0.535139	0.256779
1	4.101183	-0.391579	1.596749	2.066834	0.291815	-0.656000	1.266608	-0.718640	-0.274372	0.028061	-2.126642	0.010097	0.585782
2	-1.736348	1.906490	0.774141	-1.064257	-1.497450	0.363580	-1.336261	-0.312480	-0.254927	0.479195	-0.190424	-2.554627	0.979125
3	4.326318	2.856246	0.538510	0.831858	-0.231175	0.415675	-1.669156	0.415030	0.815468	-0.000433	-0.024079	0.492276	1.819907
4	-7.052475	-3.044251	-4.666660	1.937896	2.536534	3.144730	-2.823072	0.317401	1.077497	-1.406676	-0.513131	-0.658634	-3.443136

Saving Test dataset with Predicted Results

```
Test_final1.to_csv('HousingPrice_Project_TestDataResults.csv')
```

Visualizations

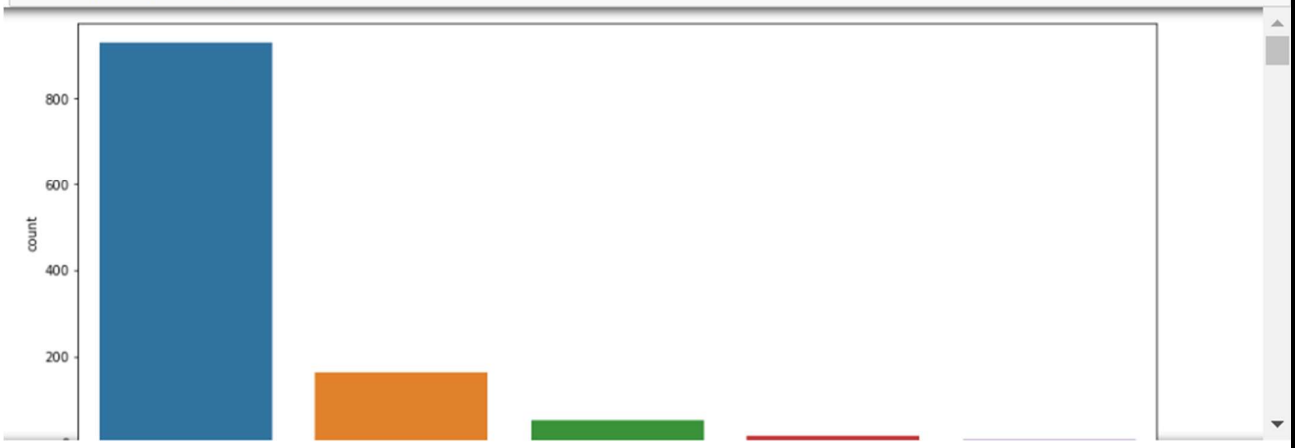
Now, we will see the different plots done with this dataset in order to know the insight of the data present. Below are the codes given for the plots and the output obtained:

Importing required libraries and plotting graphs for categorical data

Visulization

Univariate Analysis

```
# plotting categorical data
for column in train_df.columns:
    if train_df[column].dtypes == object:
        plt.figure(figsize=(15,6))
        sns.countplot(train_df[column])
        plt.show()
        print("Percentage of data: \n")
        print(round(train_df[column].value_counts()/1168*100),2)  #1168 is the entire data
        print('*'*120)
```



Observations

1. We can see that RL that is Residential Low Density has highest count among all other zoning areas followed by RM i.e. Residential Medium Density.While c ie. commercial has lowest count.
2. Raod access to property is indicated by street and we see that Pave street is commonly foubd everywhere.
3. Shape of property is mostly regular as seen in graph.
4. Land Contour tell the flatness of the property and from graph we see maximum plots have flat level.

5. In Lot Configuration, we see we have many of inside lot plots followed by corner plots.And least are Frontage on 3 sides of property.

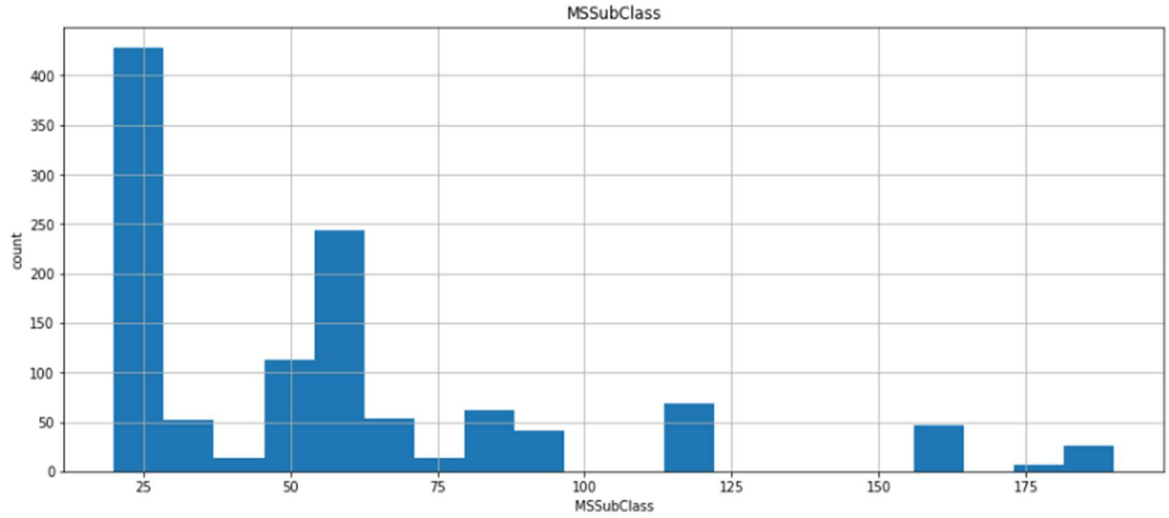
6. Most plots have gentle slope as is shown in sixth graph.
7. Maximum houses are in neighbourhood of Names ie. North Ames followed by College Creek.
8. Proximity to various conditions is mostly normal followed by Feeder Street.
9. Condition 2 also has mostly normal followed by feeder
10. Building type is mostly found as Single-family Detached followed by Townhouse End Unit
11. House styleis mostly one story followed 2story and so on
12. Roof style is mostly found to be gable followed by Hip.
13. Standard (Composite) Shingle is used the most for roof material.
14. Vinyl Siding is most found Exterior covering on house.
15. Exterior covering on house (if more than one material) also has Vinyl Siding as most used.
16. Masonry veneer type is found to be none or Brick face the most used.
17. Average/Typical quality followed by good is most found quality of the material on the exterior.
18. Present condition of the material on the exterior is mostly recorded as Typical/Average
19. Most type of foundation used is Cinder Block or Poured Contrete.
20. Height of the basement is mostly rated as average/typical followed by good.
21. General condition of the basement is also mostly average/typical.
22. Plot no 22 refers to walkout or garden level walls and mostly found as No Exposure followed by Average exposure.
23. Rating of basement finished area is mostly rated as unfinished.
24. Rating of basement finished area (if multiple types) is mostly rated as unfinished.
25. Type of heating is Gas forced warm air furnace, which is mostly found.
26. Heating quality and condition is rated Excellent, followed by Average/typical.
27. All houses have central air conditioning as shown in graph.
28. Electrical system in all houses is Standard Circuit Breakers & Romex.

29. All houses have rated average kitchen quality.
30. Home functionality is mostly found as typical functionality as shown in graph.
31. Maximum houses have No Fireplace.
32. Maximum houses have attached Garage type.

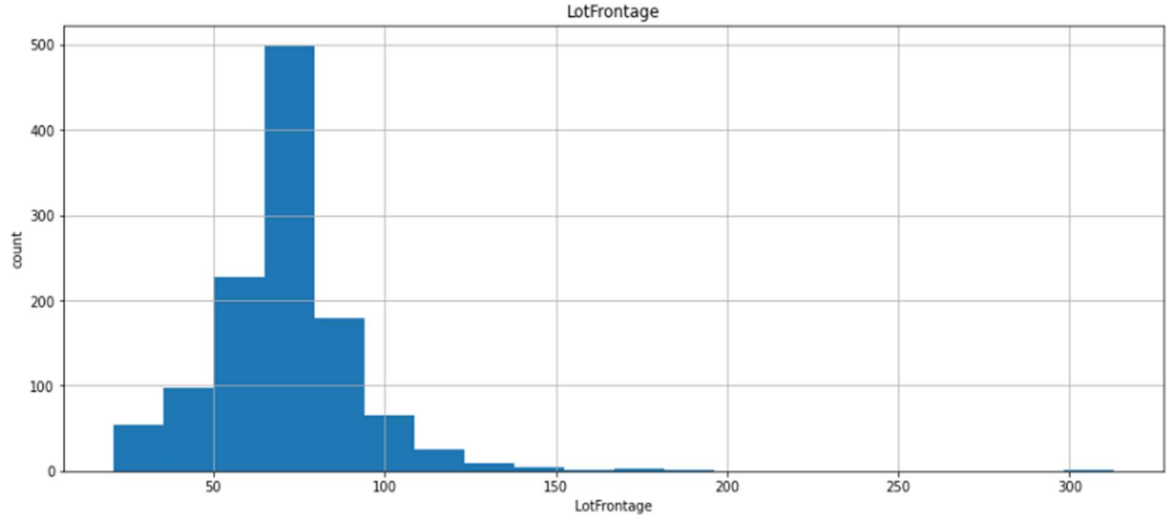
33. There are mostly unfinshed garagefinish.
34. Garage quality is mostly typical/average.
35. Garage condition is also recorded same as garage quality ie. typical.
36. Paved driveway is mostly paved only as shown in graph.
37. Types of house sales are mostly Warranty Deed - Conventional.
38. Condition of sale is noted as normal.

Taking all continuous data and plottinghistogram

```
# plotting continuous feature variables
for column in train_df.columns:
    if (train_df[column].dtypes == 'int64') or (train_df[column].dtypes == 'float64'):
        plt.figure(figsize=(15,6))
        train_df[column].hist(bins=20)
        plt.xlabel(column)
        plt.ylabel('count')
        plt.title(column)
        plt.show()
        print('*'*70)
```

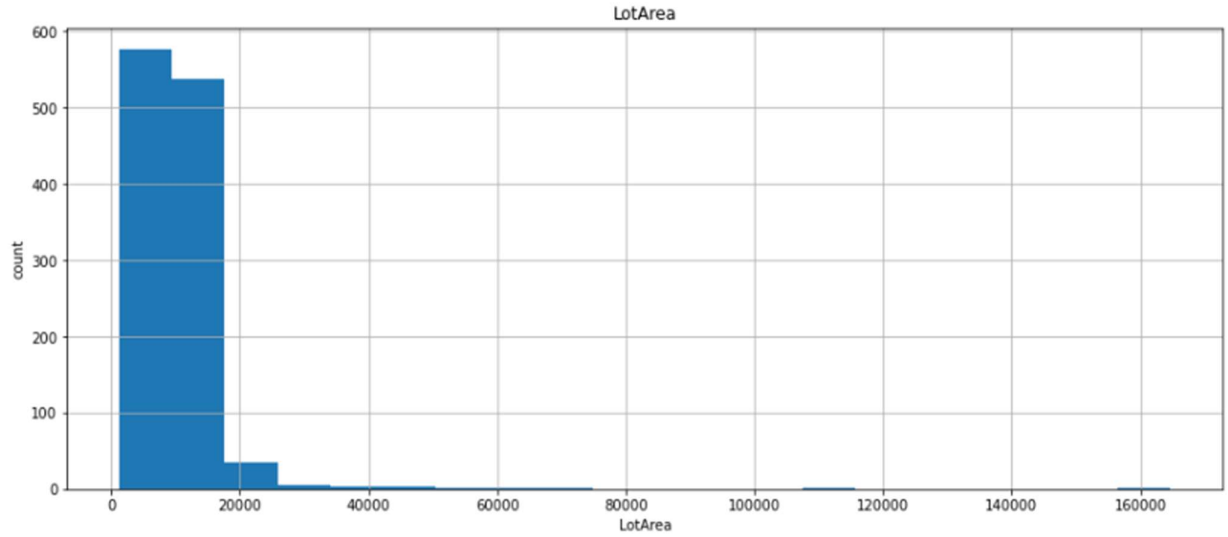

MSSubClass
count
MSSubClass
0
50
100
150
200
250
300
350
400
25
50
75
100
125
150
175

****

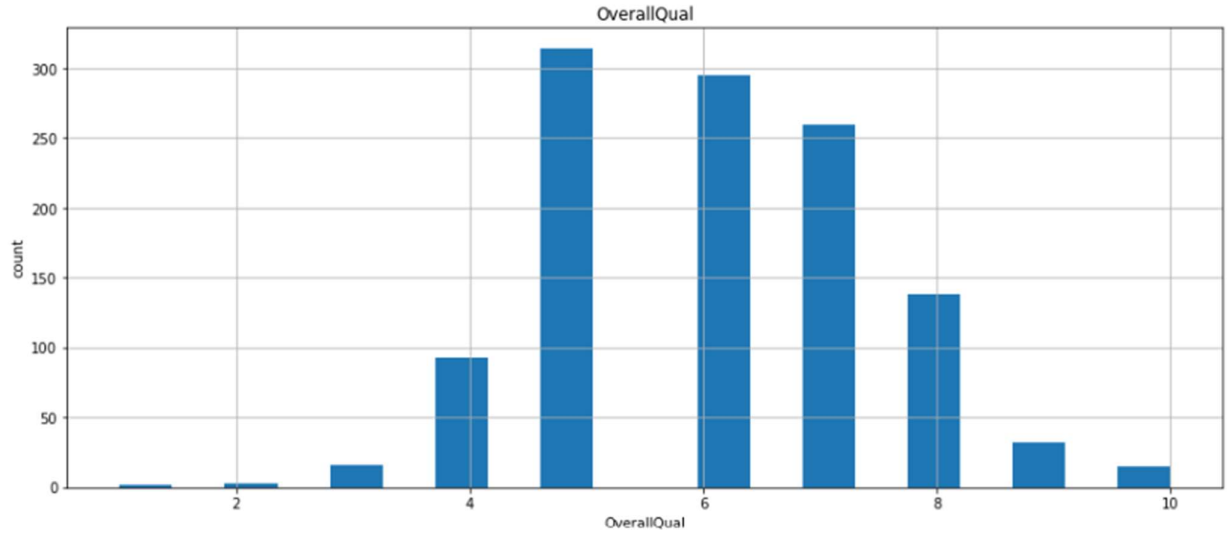

LotFrontage
count
LotFrontage
0
100
200
300
400
500
50
100
150
200
250
300

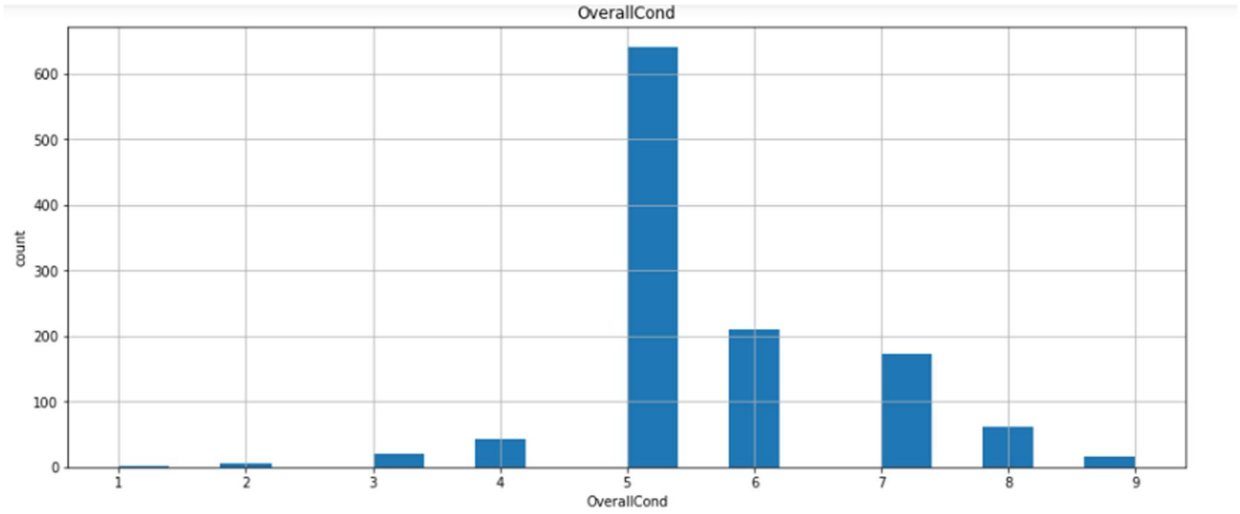
****

****

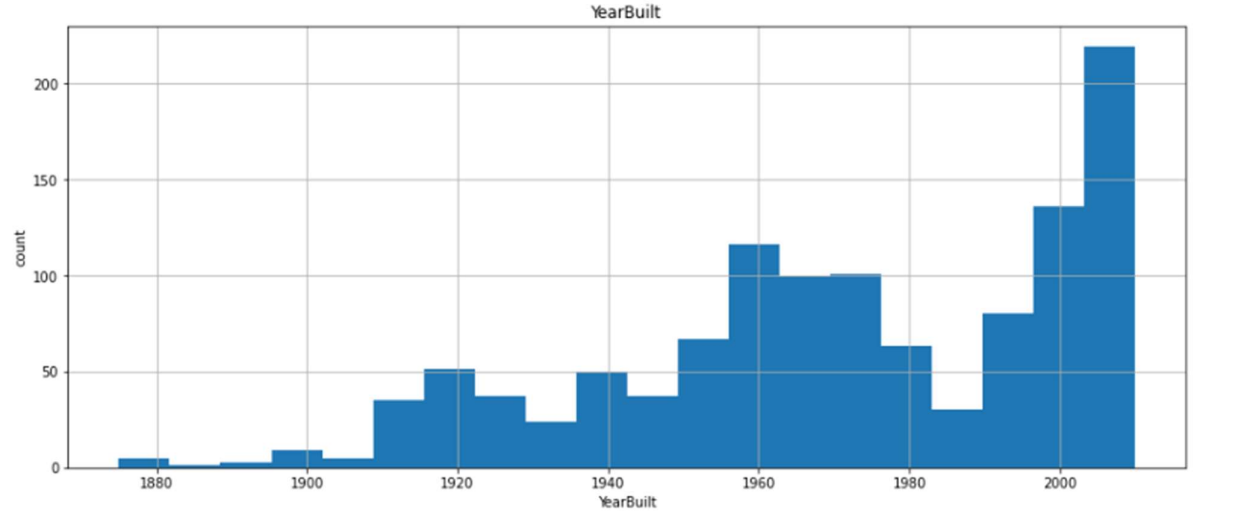

LotArea
count
LotArea
0
20000
40000
60000
80000
100000
120000
140000
160000
0
100
200
300
400
500
600

****


OverallQual
count
OverallQual
2
4
6
8
10
0
50
100
150
200
250
300


OverallCond
count
OverallCond
0
100
200
300
400
500
600
1
2
3
4
5
6
7
8
9

****


YearBuilt
count
YearBuilt
0
50
100
150
200
1880
1900
1920
1940
1960
1980
2000

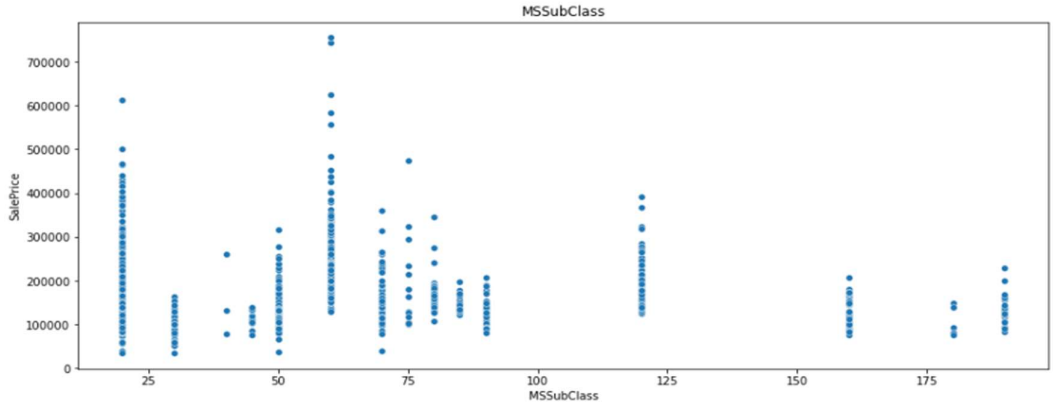
Observations:

1. Plot1 MSSubClass which Identifies the type of dwelling involved in the sale and shows that maximum dwelling lies in range of 20 - 40 then fom 55-60.
2. Linear feet of street connected to property is lot frontage and its range varies from 20 -100.
3. Lot size in square feet is spreaded in range of 0-20000.
4. Overall quality is rated highest between the range 4-6 which denotes average.
5. Overall condition of the house is rated as 5 which denotes average.
6. Houses were mostly built in year 1990 -2000.
7. Remodel date (same as construction date if no remodeling or additions) is highest in year 1950-1951 then in year 2005 -2010.
8. Masonry veneer area in square feet is mostly betweem 0-300.
9. Plot 9 shows Type 1 finished square feet and it is between range 0- 2000.
10. Plot 10 shows Type 2 finished square feet and it is between range 0-300 then again betweem 400-600.
11. Unfinished square feet of basement area i.e.BsmtUnfSF is between range 0-2000
12. Total square feet of basement area is ranged between 0-2000.
13. First Floor square feet is ranged between 300-2000.
14. Second floor square feet is ranged between 0-1500.
15. Low quality finished square feet (all floors) is ranged between 0-10.
16. Above grade (ground) living area square feet ranges between 500-3000.
17. Basement full bathrooms are mostly absent in houses.
18. Basement half bathrooms are also mostly absent in houses on sale.
19. Full bathrooms above grade are mostly found 2 in these houses.
20. Half baths above grade are mostly not present in these houses.
21. Bedrooms above grade (does NOT include basement bedrooms) are 3 in number in most houses.
22. Almost evry house has Kitchens above grade.
23. Total rooms above grade (does not include bathrooms) these are mostly in numbers 5-6 in most houses.
24. Number of fireplaces in houses are mostly not there.
25. Most Garages are built in year 1990-2010.
26. 2 garage cars space is commonly found in all houses.
27. Garage area is between 180-800.
28. Wood deck area in square feet ranges between 0-400.
29. Open porch area in square feet ranges between 0-200.
30. Enclosed porch area in square feet ranges between 0-100.
31. There are no Three season porch areas found in houses.
32. Screen porch area is also not found mostly.
33. Negligible amount of miscellaneous features are found in all houses.
34. Most houses are sold in the month of june.
35. Most houses are sold in the year start of 2007.
36. sales prices of the houses ranges between 100000-500000.

Bivariate Analysis

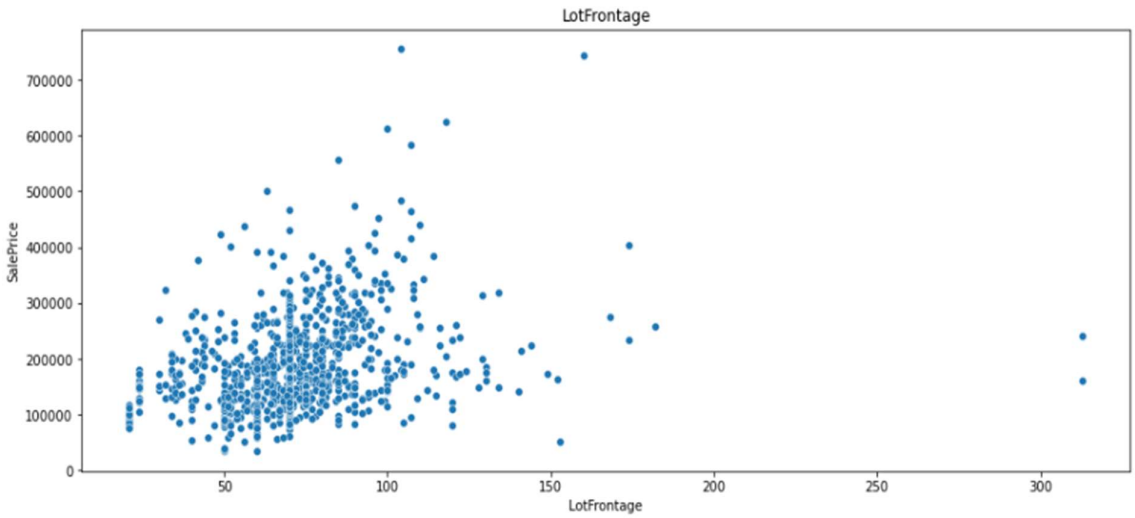
Bivariate Analysis

```
# plotting continuous feature variables versus sales price
for column in train_df.columns:
    if (train_df[column].dtypes == 'int64') or (train_df[column].dtypes == 'float64'):
        plt.figure(figsize=(15,6))
        sns.scatterplot(data= train_df,x= train_df[column], y= 'SalePrice')
        plt.xlabel(column)
        plt.ylabel('SalePrice')
        plt.title(column)
        plt.show()
        print('*'*100)
```

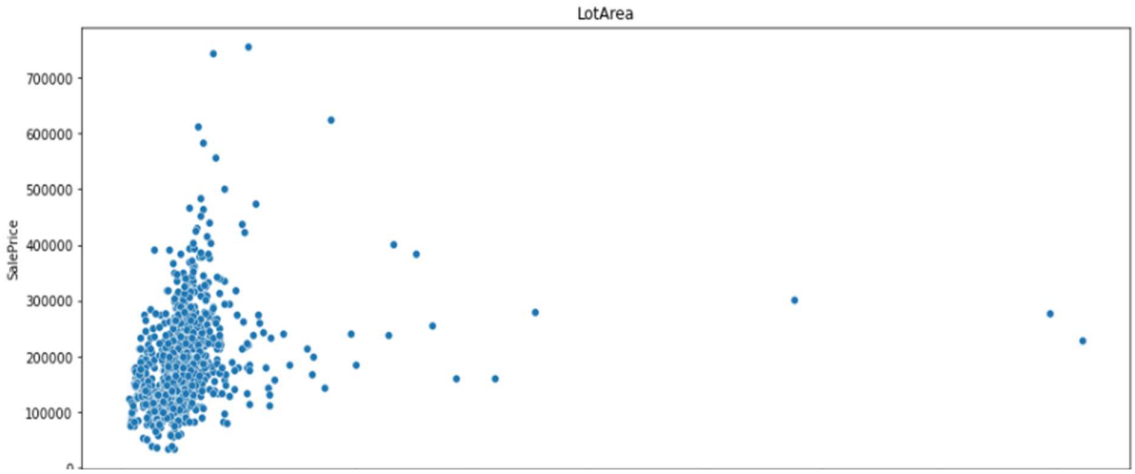


****

****

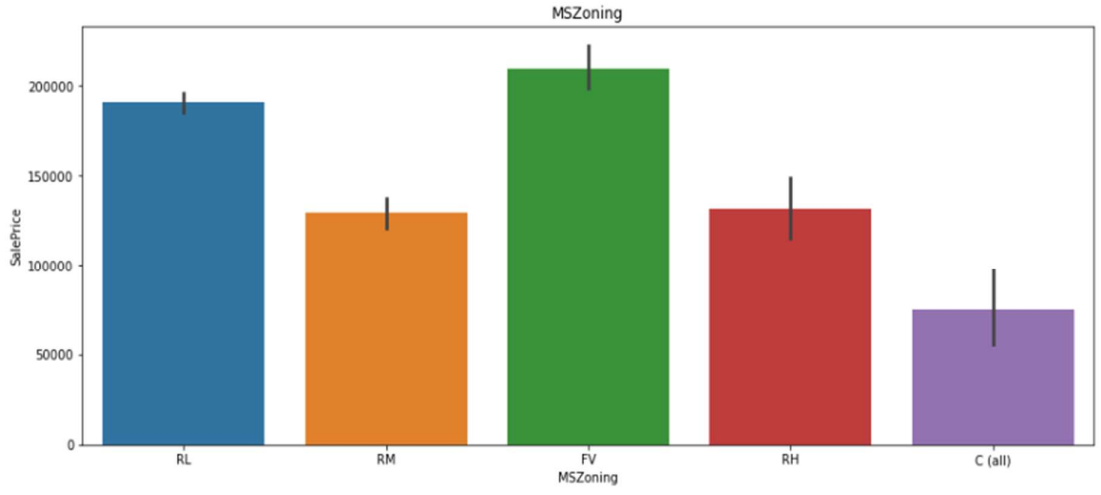

LotFrontage
SalePrice
LotFrontage
0
100000
200000
300000
400000
500000
600000
700000
50
100
150
200
250
300

****


LotArea
SalePrice
0
100000
200000
300000
400000
500000
600000
700000

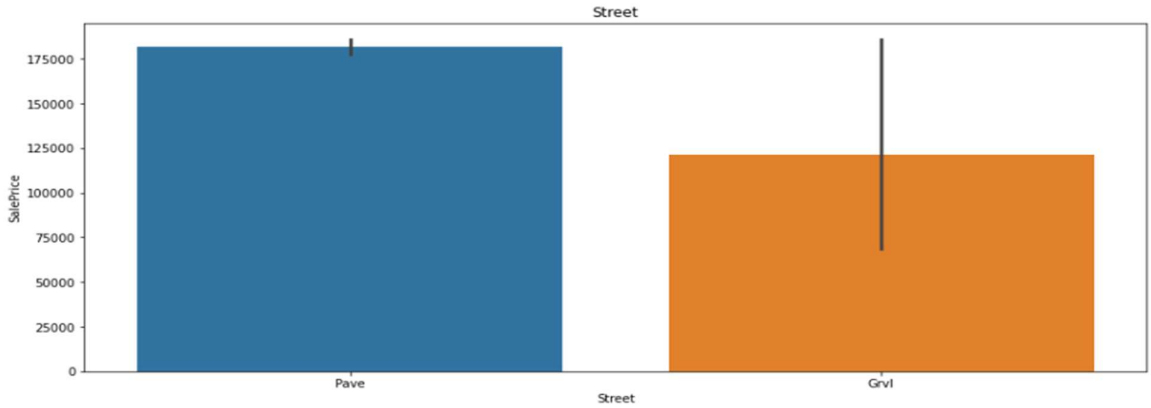
Observations:
1.Highest house price of dwelling involved in the sale is between 50-60 i.e 1-1/2 STORY FINISHED ALL AGES and 2-STORY 1946 & NEWER 2.Linear feet of street connected to property-: Lot frontage did not impact much on sale price since houses with different sale price are having same Lot frontage area.
3.LotArea: Lot size in square feet-: LotArea does not affect sale price of the houses much, as can be seen different sale price are availble within the Lot area range of 0 to 20000.In fact some houses where Lot Area is very large have moderate sale price.
4. OverallQual: Rates the overall material and finish of the house-:Overall quality is directly proportional to the sale price of houses.
5.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.
6.BsmtFinSF1: Type 1 finished square feet-:Total sq ft of basement area is directly proportional to sale price Houses with higher number of full bathrooms seems having high sale price.
7.TotalBsmtSF graph shows high basement area increases sales price of the house.
8.Increase in square foot of first floor and second floor increases the sales price of the house.
9.Above grade (ground) living area square feet is directly proportional to the sales price.
10.Basement full bathrooms shows no such connection with the sales price.
11.Same is with full baths, half baths and bsmt half bath no direct connection with the sales price.
12.Bedroom above grade with grade 4 has most sales.
13.1 Kitchen above grade has shown more sales than 0,2 or 3.
14.Total Rooms above grade in 10 nnumvbers has shown more sales.
15.Houses having 2 fireplaces shows more sales.
16.Garage year built in 2000 has more sales.
17.3 garage car parking shows highest sales in houses.
18.Above 800 garage area has highest sales price.
19.Wood deck area in square feet shows no considerable relation with.
20.0-100 sF area of open porch has shown highes sales price.
21.Enclosed porch area, Three season porch area,Screen porch area shows no consderable realtionship with sales price.
22.Zero Value of miscellaneous feature is showing highest sales price of the house.
23.Most sales are in the month of may.
24.2007 is the year showing highest value sale of the houses.

```
# plotting categorical data versus sales price
for column in train_df.columns:
    if train_df[column].dtypes == object:
        plt.figure(figsize=(15,6))
        sns.barplot(data= train_df,x=train_df[column],y='SalePrice' )
        plt.xlabel(column)
        plt.ylabel('SalePrice')
        plt.title(column)
        plt.show()
        print('*'*120)
```

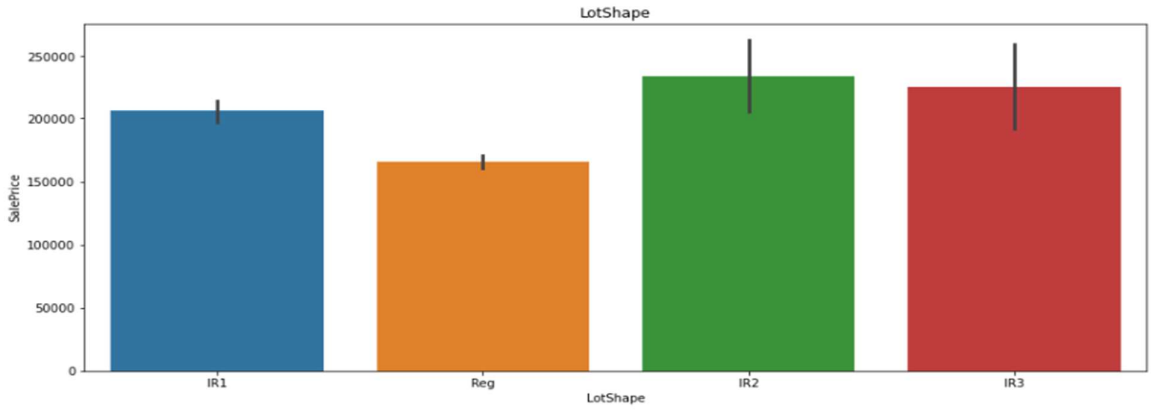


****

****



****



Observations:

1.Floating Village Residential zoning has shown more sales compared to other zones.
2.Pave street houses are sold more than gravel street.
3.Moderately Irregular shape of property is sold more followed by irregular.
4.Hillside property is showing high sales price.
5.Cul-de-sac configuration of land has more salesprice followed by Frontage on 3 sides of property.
6.Severe land slope has greater price followed by moderate then gentle.
7.Northridge neighbourhood has expensive house prices. 8. Houses Adjacent to postive off-site feature i.r. PosA are more expensive.
9.Houses Near positive off-site feature--park, greenbelt, etc are more expensive.
10.Townhouse End Unit buildings have found expensive houses prices.
11.Two and one-half story: 2nd level finished have greater sales price than other house styles.
12.Shed roof style is the most expensive one.
13.Wood Shingles roof material has more price of house.
14.Stone exterier has highest house price.
15.Stone Masonry veneer type has highest sales price.
16.Execellent exterier quality has highest sales price.
17.Poured Contrete foundation is most expensive sales.
18.Excellent bsmt quality and good bsmt condition and bsmt exposure have high sales prices.
19.Good Living Quarters has high sales price.
20.Houses having central air have high prices.
21.Houses having Standard Circuit Breakers & Romex have high prices.
22.excellent kitchen quality houses are sold with high price.
23.typical funtionality houses are sold with high prices.
24.Bulletin garage type has high sale price.
25.Partial sale conditon have high sale price.

Key Findings and Conclusions of the Study

● After getting an insight of 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 Ridge 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 RidgeRegressor 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 90.75 and cross_val_score of 88.89 which is the highest scores among all.

● We saved the model in a joblib 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 and outliers, 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 RidgeRegressor 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 ofData 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 finaldecisions 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 categoriesof 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. Anddropping them had the loss of data.

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 allthe model again