HOUSE PRICE PREDICTION PROJECT

Submitted by:

Swati Pandey

ACKNOWLEDGMENT

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

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

I want to thank my SME Srishti Maan for providing theDataset and helping us to solve the problem and addressing out our Query in right time.

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

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

INTRODUCTION

Business Problem Framing

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

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

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

Conceptual Background of the Domain Problem

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

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

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

Motivation for the Problem Undertaken

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

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

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

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

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

ANALYTICAL PROBLEM FRAMING

Mathematical/ Analytical Modelling of the Problem

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

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

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

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

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

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

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

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

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

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.

9. **Support Vector Regressor –** SVR is a supervised learning algorithm, That is used to predict discrete values. SVR uses the same principle As the SVMs. The basic idea behind SVR is to fit best line. In SVR the Best fit line is the hyperplane that has the maximum number of points.

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

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

Data Sources and their formats

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

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

Loading the train dataset

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

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

5 rows × 81 columns

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

```
(1168, 81)
```

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

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

Data description

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

1. ***MSSubClass**: Identifies the type of dwelling involved in the sale.*
2. ***MSZoning**: Identifies the general zoning classification of the sale.*

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

36. ***BsmtFinSF2****: Type 2 finished square feet*
37. ***BsmtUnfSF****: Unfinished square feet of basement area*
38. ***TotalBsmtSF****: Total square feet of basement area*
39. ***Heating****: Type of heating*
40. ***HeatingQC****: Heating quality and condition*
41. ***CentralAir****: Central air conditioning*
42. ***Electrical****: Electrical system*
43. ***1stFlrSF****: First Floor square feet*
44. ***2ndFlrSF****: Second floor square feet*
45. ***LowQualFinSF****: Low quality finished square feet (all floors)*
46. ***GrLivArea****: Above grade (ground) living area square feet*
47. ***BsmtFullBath****: Basement full bathrooms*
48. ***BsmtHalfBath****: Basement half bathrooms*
49. ***FullBath****: Full bathrooms above grade*
50. ***HalfBath****: Half baths above grade*
51. ***Bedroom****: Bedrooms above grade (does NOT include basement bedrooms)*
52. ***Kitchen****: Kitchens above grade*
53. ***KitchenQual****: Kitchen quality*
54. ***TotRmsAbvGrd****: Total rooms above grade (does not include bathrooms)*
55. ***Functional****: Home functionality (Assume typical unless deductions are warranted)*
56. ***Fireplaces****: Number of fireplaces*
57. ***FireplaceQu****: Fireplace quality*
58. ***GarageType****: Garage location*
59. ***GarageYrBlt****: Year garage was built*
60. ***GarageFinish****: Interior finish of the garage*
61. ***GarageCars****: Size of garage in car capacity*
62. ***GarageArea****: Size of garage in square feet*
63. ***GarageQual****: Garage quality*
64. ***GarageCond****: Garage condition*
65. ***PavedDrive****: Paved driveway*
66. ***WoodDeckSF****: Wood deck area in square feet*
67. ***OpenPorchSF****: Open porch area in square feet*
68. ***EnclosedPorch****: Enclosed porch area in square feet*
69. ***3SsnPorch****: Three season porch area in square feet*

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

Checking the data type & info of dataset

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

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

Checking the no. of null values in the dataset

```
## Checking null values
df_train.isnull().sum().sort_values(ascending = False).head(50)
```

```
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
```

Data Pre-processing

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

Checking the value counts of categorical data

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

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


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


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


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

Observations:

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

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

Handling missing data

```
## Let's replace missing data with 'No_alley_access'
df_train['Alley'].fillna('No_alley_access',inplace=True)
print(df_train['Alley'].value_counts())

# Let's replace missing data with 'No_Fence'
df_train['Fence'].fillna('No_Fence',inplace=True)
print(df_train['Fence'].value_counts())

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

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

garage=['GarageFinish','GarageType','GarageQual','GarageCond']
# Let's replace NAs with 'No_Garage'
for i in garage:
    df_train[i].fillna('No_Garage',inplace=True)
    print(df_train[i].value_counts())

#Let's replace GarageYrBlt with median
df_train["GarageYrBlt"].fillna(df_train["GarageYrBlt"].median())
print(df_train['GarageYrBlt'].value_counts())

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

```
35
36 ##Let's replace missing data with median
37 df_train['MasVnrArea'].fillna(df_train['MasVnrArea'].median())
38 print(df_train['MasVnrArea'].value_counts())
39
40 #Let's fill the missing values in MasVnrType with Mode
41 df_train['MasVnrType'].fillna(df_train['MasVnrType'].mode()[0])
```

Dropping some unnecessary columns

```
In [3]:
## Dropping unnecessary columns:
df_train.drop(columns =['PoolQC','MiscFeature','Id','Utilities'],inplace = True)
```

```
## Dropping unnecessary columns which are very weakly correlated with Target:
df_train.drop(columns=['3SsnPorch','Street','Condition2','LandContour','MasVnrType','LandSlope'],inplace= True)
```

Checking the statistical summary of the dataset

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

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

```
## Statistical information about Dataset
df_train.describe()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood
count	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
mean	56.767979	3.013699	70.807363	10484.749144	0.996575	0.995719	1.938356	2.773973	3.004281	0.064212	12.145548
std	41.940650	0.633120	22.440317	8957.442311	0.058445	0.256832	1.412262	0.710027	1.642667	0.284088	6.010364
min	20.000000	0.000000	21.000000	1300.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	20.000000	3.000000	60.000000	7621.500000	1.000000	1.000000	0.000000	3.000000	2.000000	0.000000	7.000000
50%	50.000000	3.000000	70.000000	9522.500000	1.000000	1.000000	3.000000	3.000000	4.000000	0.000000	12.000000
75%	70.000000	3.000000	79.250000	11515.500000	1.000000	1.000000	3.000000	3.000000	4.000000	0.000000	17.000000
max	190.000000	4.000000	313.000000	164660.000000	1.000000	2.000000	3.000000	3.000000	4.000000	2.000000	24.000000

Observations:

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

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

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

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

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

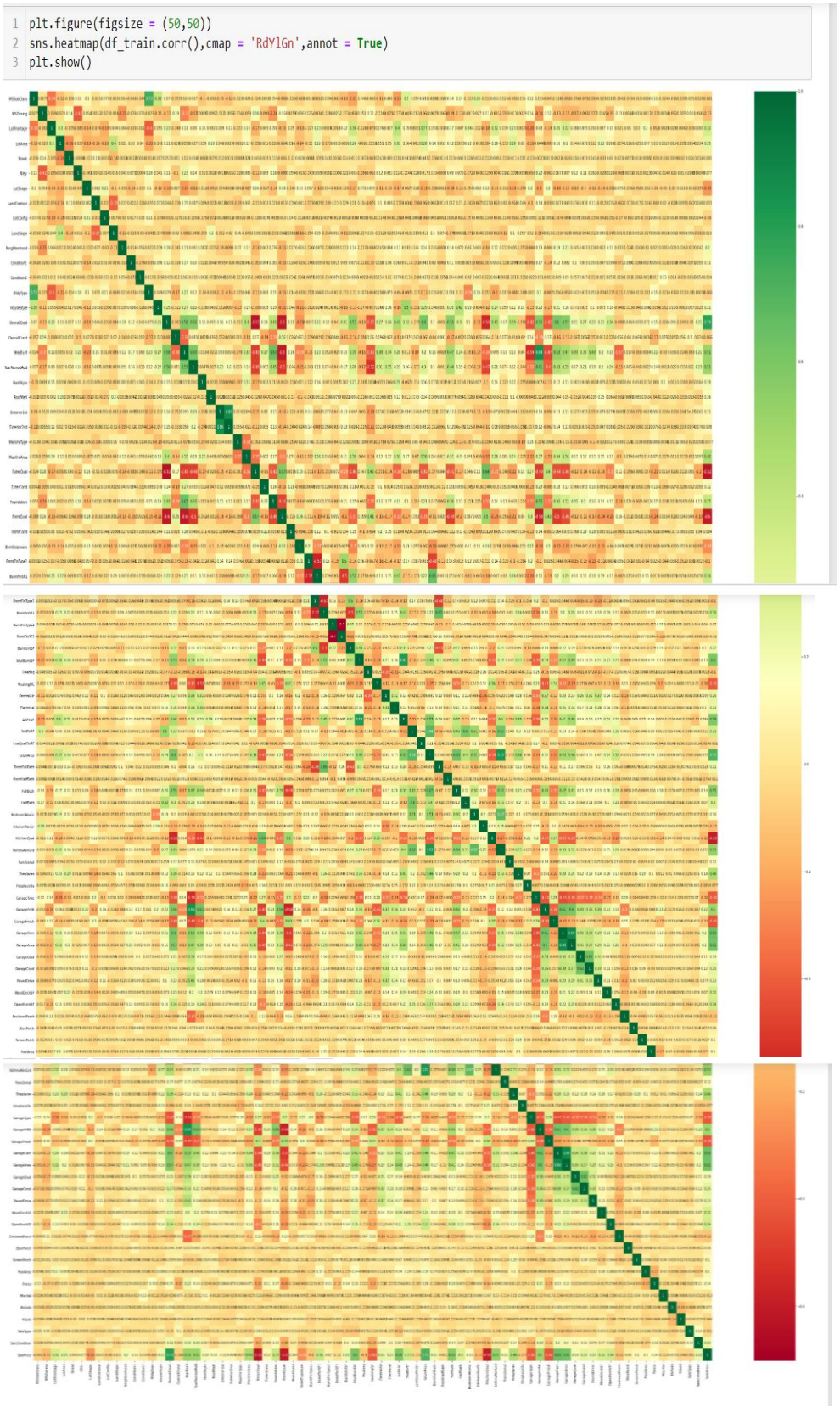
Correlation Factor

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

Correlation matrix and its visualization

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

```
plt.figure(figsize = (50,50))
sns.heatmap(df_train.corr(),cmap = 'RdYlGn',annot = True)
plt.show()
```



Observations:

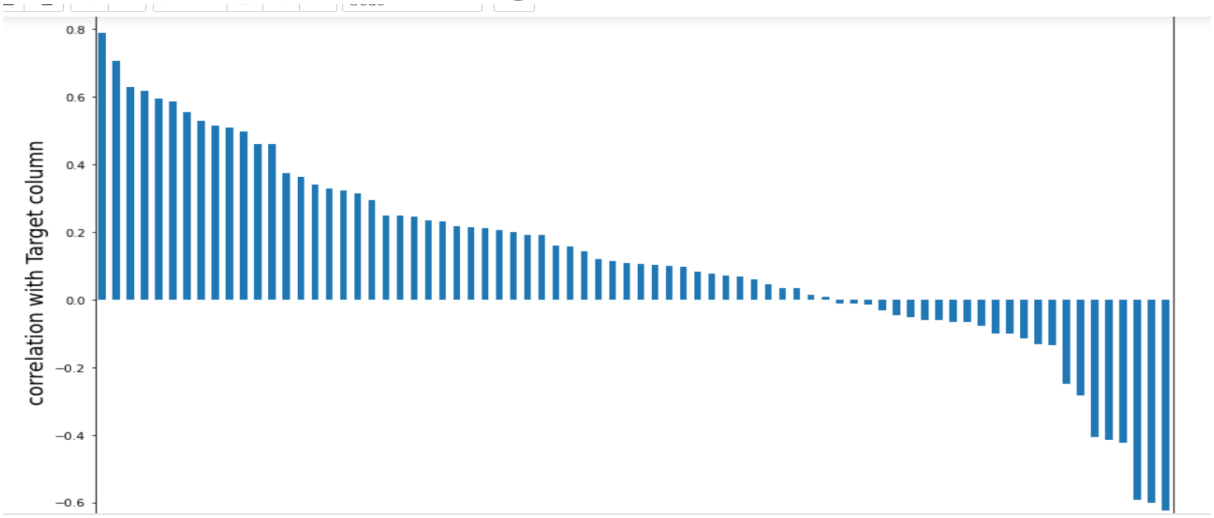
-> SalePrice is highly positively correlated with the columns OverallQual, YearBuilt, YearRemodAdd, TotalBsmtSF, 1stFlrSF, GrLivArea, FullBath, TotRmsAbvGrd, GarageCars, GarageArea.

-> SalePrice is negatively correlated with OverallCond, KitchenAbvGr,

Encloseporch, YrSold.

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

Correlation with target variable



Observations:

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

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

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

```
## Dropping unnecessary columns which are very weakly correlated with Target:
df_train.drop(columns=['3SsnPorch','Street','Condition2','LandContour','MasVnrType','LandSlope'],inplace= True)
```

Label Encoding

```
from sklearn.preprocessing import LabelEncoder
categorical_cols = [ i for i in df_train.columns if df_train[i].dtype==object]
lec = LabelEncoder()
for column in categorical_cols:
    df_train[column] = lec.fit_transform(df_train[column])
df_train
```

Checking skewness and plotting the distribution plot

Skewness

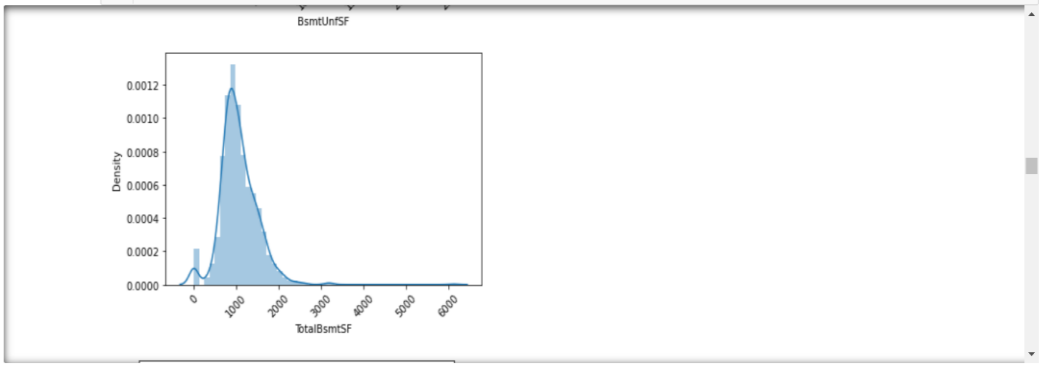
```
new_df.skew().head(50)
```

```
MSSubClass       1.413761
MSZoning        -1.754509
LotFrontage      0.741021
LotArea          3.225453
Alley           -0.216939
LotShape        -0.632205
LotConfig       -1.146376
Neighborhood     0.051969
Condition1       3.079625
BldgType         2.288999
HouseStyle       0.283209
OverallQual      0.126376
OverallCond      0.598534
YearBuilt       -0.593711
YearRemodAdd    -0.495882
RoofStyle        1.524258
RoofMatl         8.558609
Exterior1st     -0.615388
Exterior2nd     -0.594456
MasVnrArea       2.438259
ExterQual       -1.825707
ExterCond       -2.565526
Foundation      -0.006556
BsmtQual        -0.473795
```

In [24]:

```
## Checking skewness using distribution plot:
Numerical_cols = df_train._get_numeric_data().columns

plt.figure(figsize=(20,5))
for i in Numerical_cols:
    sns.distplot(df_train[i])
    plt.xticks(rotation=45)
    plt.show()
```

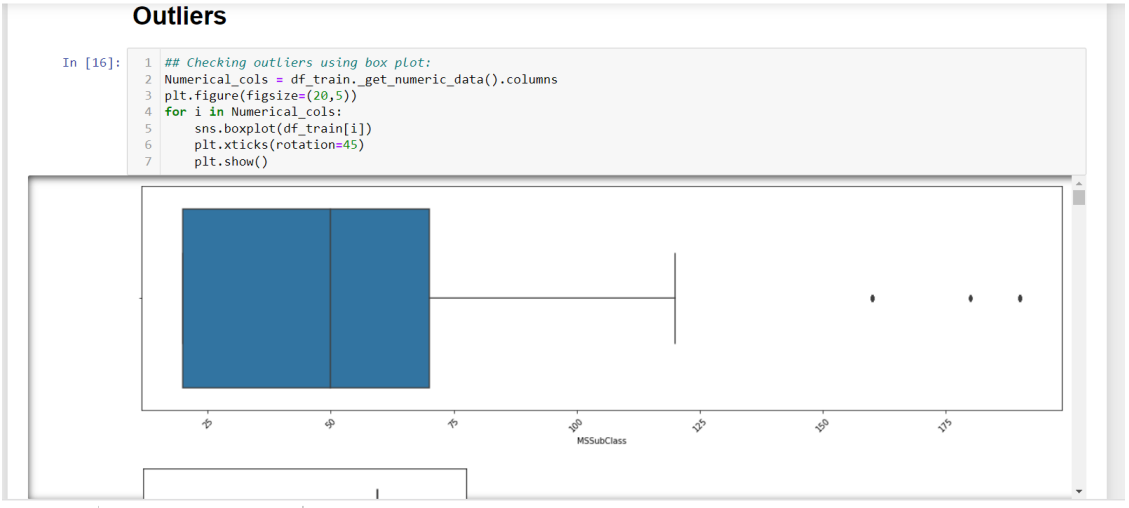


Skewness refers to distortion or asymmetry in a symmetrical bell curve, or normal distribution in a set of data. Besides positive and negative skew, distributions can also be said to have zero or undefined skew. The skewness value can be positive, zero, negative, or undefined.

Checking outliers and plotting it

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

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



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, PowerTransformer,etc.

1. Log Transform

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

2. Square Root Transform

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

3. Box-Cox Transform

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

```
from sklearn.preprocessing import PowerTransformer
skewed_features = ['MSZoning','LotFrontage','LotArea','MasVnrArea','BsmtFinSF1','BsmtFinSF2','TotalBsmtSF','Heating','Centra
'BsmtHalfBath','KitchenAbvGr','Functional','EnclosedPorch','ScreenPorch','PoolArea','MiscVal']
scaler = PowerTransformer(method='yeo-johnson')
```

```
new_df[skewed_features] = scaler.fit_transform(new_df[skewed_features].values)
new_df[skewed_features].head()
```

	MSZoning	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	TotalBsmtSF	Heating	CentralAir	Electrical	LowQualFinSF	BsmtHalfBath	Kitch
0	-0.161785	0.035780	-1.258848	-0.822839	-0.108716	-0.347157	0.110018	-0.09198	0.267261	0.300154	-0.089803	-0.239439	-(
1	-0.161785	1.232779	1.302418	-0.822839	0.398914	2.881370	2.558707	-0.09198	0.267261	0.300154	-0.089803	-0.239439	-(
2	-0.161785	1.095082	0.156607	-0.822839	0.930168	-0.347157	0.200624	-0.09198	0.267261	0.300154	-0.089803	-0.239439	-(
3	-0.161785	1.681925	0.550076	1.383257	0.801744	-0.347157	1.793828	-0.09198	0.267261	0.300154	-0.089803	-0.239439	-(
4	-0.161785	0.035780	1.428002	1.138376	1.181170	-0.347157	1.280792	-0.09198	0.267261	0.300154	-0.089803	4.176437	-(

Hardware and Software Requirements and Tools Used

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

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

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

9. train_test_split- to split the test and train data.
10. Then I used different classification algorithms to find out the best model for predictions.
11. pickle- 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 “SalePrice” has continuous output. So, for further analysis of the problem, we have to import or call out the Regression related libraries in Python work frame.

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

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

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

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

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

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

10. **SupportVectorRegressor:** SVR is a supervised learning algorithm, That is used to predict discrete values. SVR uses the same principle As the SVMs. The basic idea behind SVR is to fit best line. In SVR the Best fit line is the hyperplane that has the maximum number of points.

2. sklearn.tree –

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

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

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

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

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

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

3. sklearn.ensemble

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

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

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

i. Random Forest Regressor - It is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. A Random Forest's nonlinear nature can give it a leg up over linear algorithms, making it a great option. Random forest is a type of supervised learning algorithm that uses ensemble methods (bagging) to solve both regression and classification problems. The algorithm operates by constructing a multitude of decision trees at training time and outputting the mean/mode of prediction of the individual trees.

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

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

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

Important sklearn.metrics modules used in the project are:

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

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

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

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

5. sklearn.model_selection –

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

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

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

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

Testing of Identified Approaches

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

Splitting Features and Target variable

```
In [27]:
x=new_df.drop(columns=['SalePrice'])
y=new_df['SalePrice']

In [28]:
x.shape,y.shape

Out[28]: ((1125, 70), (1125,))
```

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.

Standardization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x=pd.DataFrame(scaler.fit_transform(x),columns=x.columns)
x.head()
```

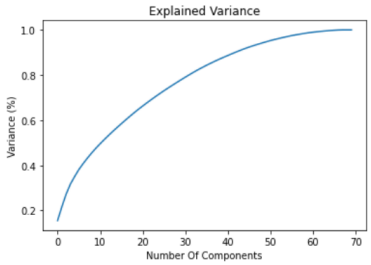
LotConfig	Neighborhood	Condition1	BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd
0.599400	0.153430	-0.041232	2.957007	-0.563447	-0.064735	-0.524013	0.148053	-0.426114	-0.484515	-0.120561	-0.207862	-0.09993
0.599400	-0.013342	-0.041232	-0.406626	-0.563447	1.391812	0.373270	-0.052501	-0.714519	-1.706667	7.113082	1.080541	1.05021
-1.236331	0.486975	-0.041232	-0.406626	1.022569	0.663538	-0.524013	0.816566	0.583305	-0.484515	-0.120561	-0.529963	-0.67500
0.599400	0.320202	-0.041232	-0.406626	-0.563447	-0.064735	0.373270	0.181479	-0.378046	1.959789	-0.120561	-0.207862	-0.09993
-0.624421	0.320202	-0.041232	-0.406626	-0.563447	-0.064735	1.270553	0.181479	0.727508	-0.484515	-0.120561	-1.496265	-1.53761

Using PCA

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

Dimension Reduction using PCA

```
from sklearn.decomposition import PCA
pca = PCA()
comp = pca.fit_transform(x)
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number Of Components')
plt.ylabel('Variance (%)')
plt.title('Explained Variance')
plt.show()
```



Run and evaluate selected models

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

```
## Importing necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Lasso,Ridge
from sklearn.metrics import r2_score
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_absolute_error,mean_squared_error
from sklearn.model_selection import cross_val_score
```

```
#Finding the best random state
max_r2_score=0
for r_state in range(50,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=r_state,test_size=0.25)
    lr=LinearRegression()
    lr.fit(x_train,y_train)
    y_pred=lr.predict(x_test)
    score=r2_score(y_test,y_pred)
    if score>max_r2_score:
        max_r2_score=score
        final_r_state=r_state
print("max r2 score corresponding to",final_r_state,"is",max_r2_score)
```

```
max r2 score corresponding to 59 is 0.8962409527557181
```

Train Test Split

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

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

```
#Creating train_test_split using best random_state

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=r_state,test_size=.25)
```

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

```
lr=LinearRegression()
lasso=Lasso()
ridge=Ridge()
svr=SVR()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
rfr = RandomForestRegressor()
adb = AdaBoostRegressor()
```

```
models= []
models.append(('Linear Regression',lr))
models.append(('Lasso Regression',lasso))
models.append(('Random Forest Regression',rfr))
models.append(('Ridge Regression',ridge))
models.append(('Support Vector Regressor',svr))
models.append(('Decision Tree Regressor',dtr))
models.append(('KNeighbors Regressor',knr))
models.append(('AdaBoost Regressor',adb))

```

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:

```
Model=[]
score=[]
cvs=[]
sd=[]
mae=[]
mse=[]
rmse=[]
for name,model in models:
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pred=model.predict(x_test)
    print('\n')
    S=r2_score(y_test,pred)
    print('r2_score: ',S)
    score.append(S*100)
    print('\n')
    sc=cross_val_score(model,x,y,cv=5,scoring='r2').mean()
    print('cross_val_score: ',sc)
    cvs.append(sc*100)
    print('\n')
    std=cross_val_score(model,x,y,cv=5,scoring='r2').std()
    print('Standard Deviation: ',std)
    sd.append(std)
    print('\n')
    MAE=mean_absolute_error(y_test,pred)
    print('Mean Absolute Error: ',MAE)
    mae.append(MAE)
    print('\n')
```

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

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

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

```
#making dataframe for results got from model:
result_df=pd.DataFrame({'Model':Model, 'r2_score': score, 'Cross_val_score':cvs, 'Standard_deviation':sd,
                        'Mean_absolute_error':mae, 'Mean_squared_error':mse, 'Root_Mean_Squared_error':rmse})
result_df
```

	Model	r2_score	Cross_val_score	Standard_deviation	Mean_absolute_error	Mean_squared_error	Root_Mean_Squared_error
0	Linear Regression	89.495555	84.369406	0.043270	17288.696357	5.495136e+08	23441.706471
1	Lasso Regression	89.496381	84.371008	0.043267	17287.129332	5.494704e+08	23440.784828
2	Random Forest Regression	84.856300	83.291483	0.024884	18476.538014	7.922046e+08	28146.129264
3	Ridge Regression	89.498179	84.376686	0.043229	17283.140635	5.493764e+08	23438.778807
4	Support Vector Regressor	-3.423302	-6.155953	0.055993	52764.402851	5.410330e+09	73554.945200
5	Decision Tree Regressor	72.605944	65.831744	0.057401	25991.297872	1.433051e+09	37855.662123
6	KNeighbors Regressor	84.362641	79.706915	0.039217	20230.235461	8.180291e+08	28601.208646
7	AdaBoost Regressor	80.052427	78.772036	0.032694	24191.208621	1.043507e+09	32303.361215

As we can see from above dataframe Lasso and Ridge are performing better than other algorithms.Now I'll do hyperparameter to find best parameters to increase score of models. We can also say here that our model is not overfitting or underfitting.

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

Key Metrics for success in solving problem under consideration

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

1. Cross Validation:

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

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

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

2. R2 Score:

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

3. Mean Squared Error (MSE):

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

4. Mean Absolute Error (MAE):

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

5. Hyperparameter Tuning:

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

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

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

Lasso

```
from sklearn.model_selection import GridSearchCV
parameters={'alpha' :[0.001, 0.01, 0.1, 1], 'random_state':range(0, 100), 'selection':['cyclic','random'],}
l = Lasso()
clf=GridSearchCV(l,parameters,cv=5,scoring='r2')
clf.fit(x_train,y_train)
print(clf.best_params_)  #Printing the best parameters obtained
print(clf.best_score_) #Mean cross-validated score of best_estimator
```

```
{'alpha': 1, 'random_state': 96, 'selection': 'random'}
0.8222696011370859
```

```
## providing best parameters:
#Using the best parameters obtained
l=Lasso(alpha=1, random_state=30, selection='random')
l.fit(x_train,y_train)
pred=l.predict(x_test)
print(' r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(l,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(l,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
 r2_score after tuning is:  89.49636600737745
Cross validation score:  84.37100379861427
Standard deviation:  0.04326697046792134
```

Ridge

```
params = {'alpha':[0.001, 0.01, 0.1, 1],'solver' : ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'],'random_s
rd = Ridge()
clf=GridSearchCV(rd,params,cv=5,scoring='r2')
clf.fit(x_train,y_train)
print(clf.best_params_)  #Printing the best parameters obtained
print(clf.best_score_) #Mean cross-validated score of best_estimator

```

```
{'alpha': 1, 'random_state': 45, 'solver': 'saga'}
0.8225579575927064
```

```
## Providing best parameters to Model:
rd=Ridge(alpha=1, random_state=0, solver='sparse_cg')
rd.fit(x_train,y_train)
pred=l.predict(x_test)
print(' r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(l,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(l,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
 r2_score after tuning is:  89.49636600737745
Cross validation score:  84.37100379861427
Standard deviation:  0.04326697046792134
```

After Tuning the best algorithms, we can see that Lasso Regression has not been improved. Now, we will try Ensemble techniques like RandomForestRegressor, AdaBoostRegressor and GradientBoostingRegressor to boost up our scores.

Random Forest

```
params = {'n_estimators': [100,200],
          'criterion' : ["mse","mae"],
          'random_state':range(40,100)
         }
rf = RandomForestRegressor()
clf  = GridSearchCV(rf,params,cv=5,scoring ='r2')
clf.fit(x_train,y_train)
print(clf.best_params_)  #Printing the best parameters obtained
print(clf.best_score_) #Mean cross-validated score of best_estimator
```

```
{'criterion': 'mae', 'n_estimators': 200, 'random_state': 91}
0.8351137314579615
```

```
## Providing best parameters to the model
rf=RandomForestRegressor(criterion='mae', random_state=91, n_estimators=200)
rf.fit(x_train,y_train)
pred=l.predict(x_test)
print(' r2_score after tuning is: ',r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(l,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(l,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
 r2_score after tuning is:  89.49636600737745
Cross validation score:  84.37100379861427
```

AdaBoost Regressor

```
adr=AdaBoostRegressor(random_state=59)   #Using the best random state we obtained
parameters={'n_estimators':[10,50,100,500,1000],'learning_rate':[0.001,0.01,0.1,1],'loss':['linear','square']}
clf=GridSearchCV(adr,parameters,cv=5,scoring='r2')
clf.fit(x_train,y_train)
print(clf.best_params_)  #Printing the best parameters obtained
print(clf.best_score_) #Mean cross-validated score of best_estimator
```

```
{'learning_rate': 0.01, 'loss': 'linear', 'n_estimators': 50}
0.7856816133923608
```

```
## Providing best parameters
adr=AdaBoostRegressor(random_state=59, n_estimators=50, learning_rate=.01, loss='linear')
adr.fit(x_train,y_train)
pred=adr.predict(x_test)
print("r2_score: ",r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(adr,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(adr,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  79.09744328632264
Cross validation score:  78.97163593361334
Standard deviation:  0.02500920379886449
```

After applying Ensemble Techniques, we can see that RandomForestRegressor is the best performing algorithm among all other algorithms as it is giving a r2_score of 89.49 and crossvalidation score of 84.37. It has also the less amount of error values obtained. Lesser the RMSE score, the better the model. Now we will finalize the model.

GradientBoost

```
from sklearn.ensemble import GradientBoostingRegressor
params = {'loss' : ['ls', 'lad', 'huber', 'quantile'],
          'n_estimators':[10,50,100,500,1000],
          'learning_rate':[0.001,0.01,0.1,1],
          'criterion':['mse','mae','friedman_mse']

         }
gbr = GradientBoostingRegressor(random_state = 59)
clf=GridSearchCV(gbr,params,cv=5,scoring='r2')
clf.fit(x_train,y_train)
print(clf.best_params_)  #Printing the best parameters obtained
print(clf.best_score_) #Mean cross-validated score of best_estimator

```

```
{'criterion': 'mae', 'learning_rate': 0.1, 'loss': 'ls', 'n_estimators': 100}
0.8485877950121488
```

```
## Providing best parameter to GradientBoost
gbr=GradientBoostingRegressor(random_state=59, n_estimators=100, learning_rate=.1, criterion='mae',loss='ls')
gbr.fit(x_train,y_train)
pred=gbr.predict(x_test)
print("r2_score: ",r2_score(y_test,pred)*100)
print('Cross validation score: ',cross_val_score(adr,x,y,cv=5,scoring='r2').mean()*100)
print('Standard deviation: ',cross_val_score(adr,x,y,cv=5,scoring='r2').std())
print('\n')
print('Mean absolute error: ',mean_absolute_error(y_test,pred))
print('Mean squared error: ',mean_squared_error(y_test,pred))
print('Root Mean squared error: ',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score:  87.48711636877302
```

Saving the model

Saving the model

```
import pickle
filename = 'train_House_price_prediction.pickle'
pickle.dump(rf,open(filename,'wb'))
```

```
# gbr_prediction=gbr.predict(x)
# print('Predictions of GradientBoosting Regressor: ',gbr_prediction)   #Printing the predicted values
a = np.array(y_test)
predicted = np.array(rf.predict(x_test))
result_df = pd.DataFrame({"Original":a,"Predicted":predicted},index= range(len(a)))
result_df


```

	Original	Predicted
0	178000	191580.460
1	316600	217357.775
2	217500	200647.960
3	160000	164265.170
4	135000	134678.105
...	...	...
277	146000	160737.710
278	150000	164013.515

Using the test dataset and doing pre-processing

Test Dataset

```
df_test = pd.read_csv(r"C:/Users/swati/Downloads/Project-Housing--2-/Project-Housing_splitted/test.csv")
pd.set_option('display.max_columns',None)
df_test
```

	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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
287	83	20	RL	78.0	10206	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	
288	1048	20	RL	57.0	9245	Pave	NaN	IR2	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
289	17	20	RL	NaN	11241	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	NAmes	Norm	
290	523	50	RM	50.0	5000	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	Feedr	
291	1379	160	RM	21.0	1953	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	BrDale	Norm	

292 rows × 80 columns

```
## size of the dataset
df_test.shape
```

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

Predicting over the test data

Loading the saved model for test data Prediction

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

```
fitted_model
```

```
RandomForestRegressor(criterion='mae', n_estimators=200, random_state=91)
```

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

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

Conclusion

```
b = np.array(y_test)
predicted = np.array(fitted_model.predict(x_test))
result_df_test = pd.DataFrame({"Original":a,"Predicted":predicted},index= range(len(a)))
result_df_test
```

	Original	Predicted
0	178000	191580.460
1	316600	217357.775
2	217500	200647.960
3	160000	164265.170

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

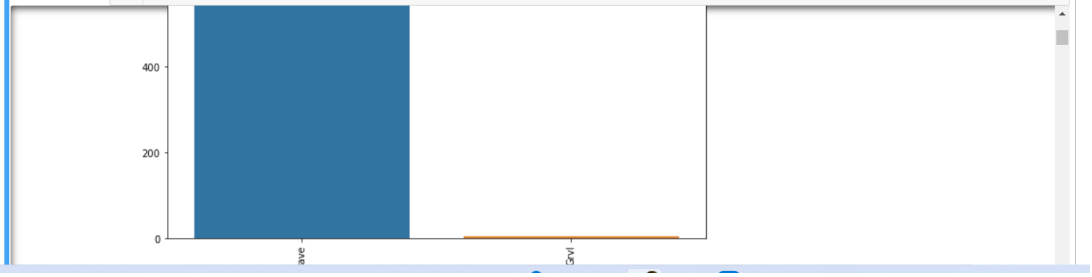
Univariate Analysis

In [11]:
```
categorical_cols = [ i for i in df_train.columns if df_train[i].dtype==object]
#categorical_cols
print(len(categorical_cols))
```

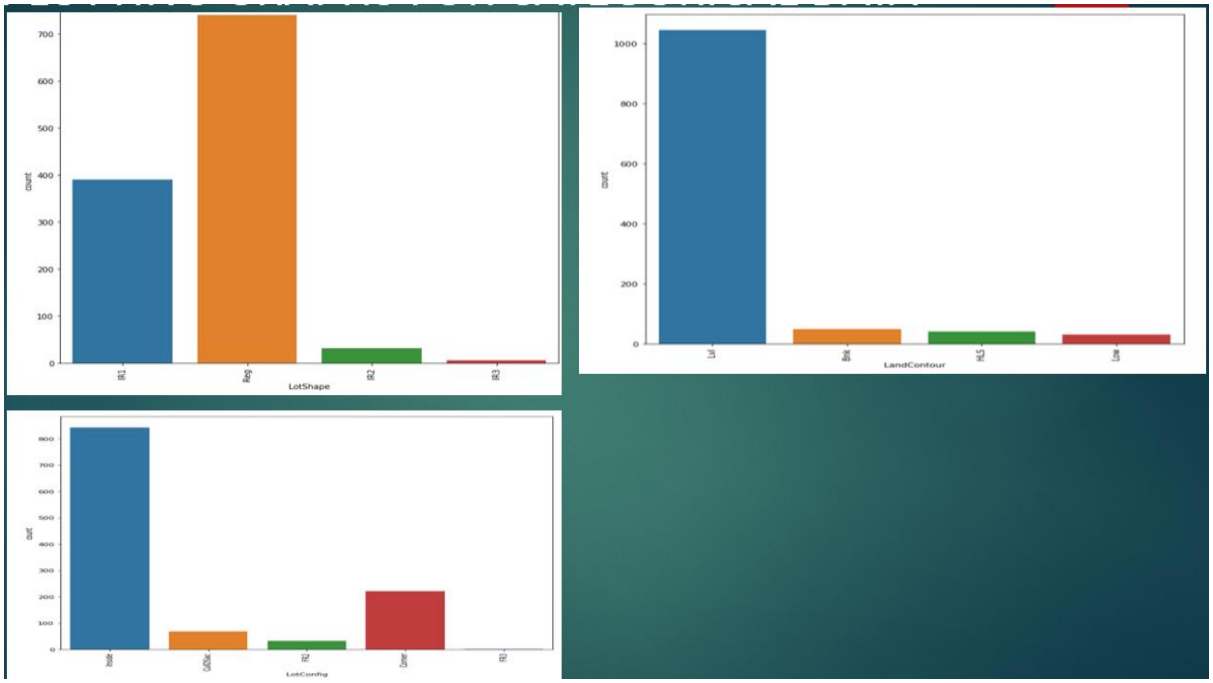
```
40
```

In [28]:
```
#Plotting countplot for each categorical data

for i in categorical_cols:
    plt.figure(figsize = (10,10))
    sns.countplot(df_train[i])
    plt.xticks(rotation = 90)
    plt.show()
```



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



Observations:

-> Residential Low Density zone's count is more in count for sale.

-> Street type Paved, road access to property is more in count for sale.

-> No_alley_access type alley is more in count for sale .

-> Regular shape property is more in count for sale followed by slightly irregular shape property.

-> flatness type near flat is nore in count for sale than other type property.

-> Lot configuration Inside is more in count for sale followed by corner lot.

-> Gentle shape of property is more in count for sale.

-> In condition1,Condition2, Normal is more in count for sale than others.

-> Type of dwelling means building type , Single family detached type is more in count for sale.

-> Style of dwelling means House Style ,1Story is more in count for sale.

-> Roofstyle Gable is more in count for sale followed by Roofstyle Hip.

-> Roof material Standard (Composite) Shingle used type house is more in count for sale than others.

-> Most of the house Masonry veneer type is None means either not mentioned or not collected at the time of data collection.

-> quality of the material on the exterior and Exterior condition is Average/Typical type house is more in count for sale.

-> Cinder Block and Poured Contrete type of foundation house is more in count for sale followed by Brick & Tile type foundation.

-> Neighborhood north park villa, Condition is mostly normal.

-> Basement quality Typical type is more in count for sale followed by Good quality.

-> Typical - slight dampness type basement condition house is more in count for sale than others.

-> No refers to walkout or garden level walls type basement exposure is more in count for sale.

-> Rating of basement finished area unfinished type houses are more in count for sale followed by Good Living Quarters rating.

-> Rating of basement finished area (if multiple types),Unfinished type rating is more in count for sale.

-> Heating type forced warm air furnace houses are most in count for sale than others.

-> Heating quality and condition excellent type houses are more in count for sale.

-> Central Air Conditioning type houses are more in count for sale.

-> Standard Circuit Breakers & Romex electrical system houses are more in count for sale.

-> Kitchen quality Typical type houses are more in count for sale.

-> Home functionality Typical are more in count for sale.

-> No Fireplace type houses are more in count for sale followed by Masonry Fireplace in main level and Prefabricated Fireplace in main living area or Masonry Fireplace in basement.

-> Garage location attached to home types are more in count for sale followed by detached from home.

-> Interior finish of garage -unfinished type houses are more in count for sale rough finished and finished.

-> Typical/Average garage quality,garage condition houses are more in count for sale.

-> Paved driveway houses are more in count for sale.

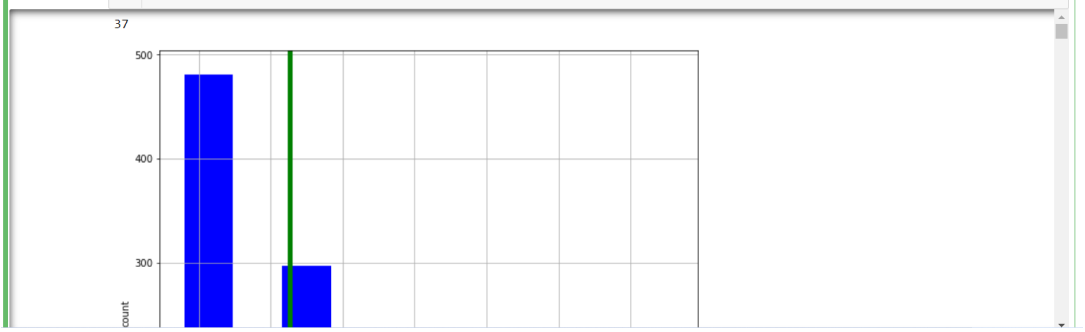
-> No fence type houses are more in count for sale.

-> Warranty Deed - Conventional type sale houses are more in count for sale.

-> Houses with normal condition sale are more in count.

Taking all continuous data and plotting histogram

```
In [29]:
## Plotting Hist plot for Numerical column
Numerical_cols =  df_train._get_numeric_data().columns
print(len(Numerical_cols))
for i in Numerical_cols:
    df_train[i].hist(bins =10,figsize=(10,10), color ="blue")
    plt.axvline(np.mean(df_train[i]), linewidth = 5, color = "green", label = "Mean (i)")
    plt.xlabel(i)
    plt.ylabel('count')
    plt.xticks(rotation = 45)
    plt.show()
    print("MATHEMATICAL SUMMARY OF", i ,":\n",\
          df_train[i].describe())
```



Observations:

-> 20-(1-STORY 1946 & NEWER ALL STYLES), 30-(1-STORY 1945 & OLDER) are high in number followed by 50 1-1/2 STORY FINISHED ALL AGES.

-> LotFrontage (linear feet of street) mean is almost 71 feet.Most of the houses has lotfrontage under the mean value.

-> Overall quality is above average for most of the houses for sale.

-> Overall condition rating below than 4 and above than 7, houses are less in number for sale and rating greater than 5 to 7(Average,Above Average and Good) are more in count for sale.

-> Residential Low Density are also high in number, Street access are mostly payment.

-> Original Construction date -Built in year after 2000 are high and year re-modify after 2010 are higher in counts for sale.

-> Masonry veneer area in square feet under the mean 101 sq feet are more in counts for sale.

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

-> BsmtFinSF2: Type 2 finished square feet-: Above 1000 houses have Type 2 finished tpye basement.

-> BsmtUnfSF: Unfinished square feet of basement area-: Around 310 houses have unfinished basesent of area around 100-500 sqft.

-> TotalBsmtSf: Total square feet of basement area, Above700 houses have less than 1062 square feet basement area, which are out for sale.

-> 1stFlrSF: First Floor square feet Around 310 houses have 1st floor square feet area between 800-1200sqft.

-> 2ndFlrSF: Around 100 houses have 2nd floor square feet area 500 to 1000.

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

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

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

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

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

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

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

few have 12 & 14 rooms.

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

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

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

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

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

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

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

-> Houses with Type 1 finished square feet area under 444.726 feet are more in counts.

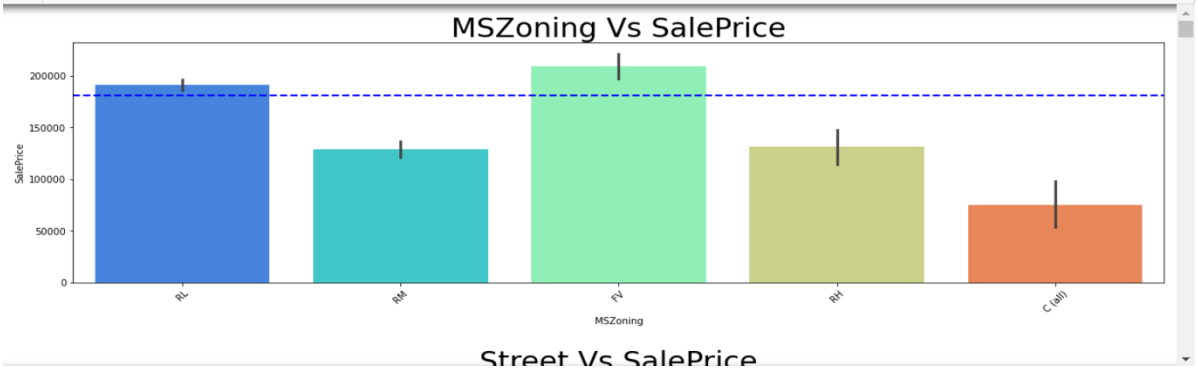
-> Properties out for sale are mostly builtin year 1977, 1970. 1997, 2006, 1957, 1965, 1947, 1937, 2003, 1974.
which means from 1945 to 2006 in all frequency we have properties largely out for sale.

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

Bivariate Analysis with SalePrice for categorical data

Bi-Variate Analysis

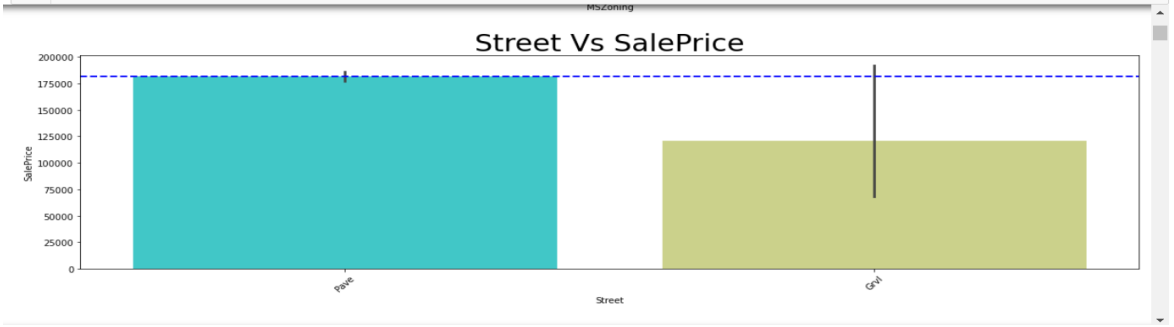
```
## Visualizing categorical  column data  relation with Target column:
for column in categorical_cols:
    plt.figure(figsize=(20,5))
    mean_cost = np.mean(df_train.SalePrice)
    sns.barplot(x = df_train[column], y = "SalePrice", data = df_train, palette = "rainbow")
    plt.axhline(mean_cost, color = "b", linestyle="dashed", linewidth=2)
    plt.title(f"{column} Vs SalePrice", fontsize = 30)
    plt.xticks(rotation=45)
    plt.show()
```

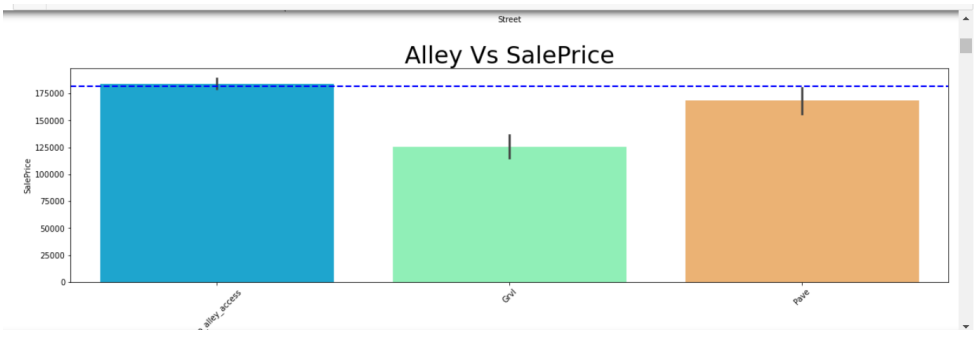


Street Vs SalePrice

Bi-Variate Analysis

```
## Visualizing categorical  column data  relation with Target column:
for column in categorical_cols:
    plt.figure(figsize=(20,5))
    mean_cost = np.mean(df_train.SalePrice)
    sns.barplot(x = df_train[column], y = "SalePrice", data = df_train, palette = "rainbow")
    plt.axhline(mean_cost, color = "b", linestyle="dashed", linewidth=2)
    plt.title(f"{column} Vs SalePrice", fontsize = 30)
    plt.xticks(rotation=45)
    plt.show()
```





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

Observations:

->2-STORY 1946 & NEWER type of dwelling sale prices are more than others.

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

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

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

->OverallCond : Rates the overall condition of the house : - Average condition type houses are more in count and sale price is more for them.

-> YearBuilt: & YearRemodAdd: Houses which are build latest, after 2000, have high sale price in comparison to those build in early years. similar is the case with remodelling date

-> MnsVnrArea

-> 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

-> GrLivAreaAbove:- grade (ground) living area square feet has linear relationship with sale price.

-> BsmtFullBath:- Basement full bathrooms, having one bathroom's houses sale price is more.

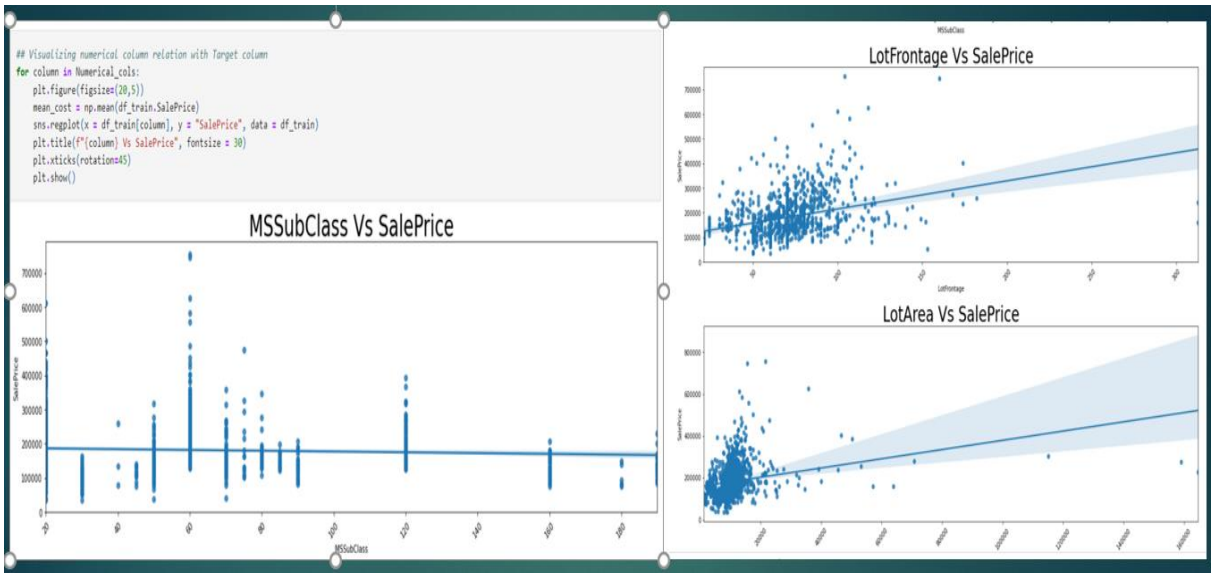
-> Kitchen: Kitchens above grade-: houses with 1 kitchen above ground have high sale price in comparison to those having 2 kitchens

-> Bedrooms above grade, having 4 bedroom's sale price is more followed by having 2 bedrooms.

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

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

Plotting for continuous data



Observations:

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

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

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

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

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

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

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

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

CONCLUSION

Key Findings and Conclusions of the Study

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

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

-> I got Lasso and 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 HyperparameterTuning.

-> 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 RandomForestRegressor 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 89.47 and cross_val_score of 84.37 which is the highest scores among all.

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

-> I predicted the values obtained and saved it.

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

-> After treating skewness, I loaded the saved model that I obtained and did the predictions over the test data .

-> From this project, I 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 andRandomForestRegressor is the best working algorithm model we obtained.

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

Learning Outcomes of the Study in respect of Data Science

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

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

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

4. I 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

Limitations of this work and Scope for Future Work

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

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

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

3. The high skewness of data reduces the effectivity

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

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