PROJECT REPORT

ON

House Price Prediction

Submitted by:

Gyan Prakash Tripathi

ACKNOWLEDGMENT

In this blog, I will be **analysing the House prediction using Machine Learning dataset** using essential exploratory data analysis techniques and also, I will be performing some data visualizations to better understand our data.

The development of a housing prices prediction model can assist a house seller or a real estate agent to make better-informed decisions based on house price valuation. By doing data preprocessing, data analysis, feature selection, and many other techniques we built our cool and fancy machine learning model. And at the end, we applied many ml algorithms to get the very good accuracy of our model.

Many thanks to Fliprobo Technology for providing me with this project to understand the Real-Time Field work present in Data Science Industry.

I am very thankful to my friends and family who helped me through this study. So without any further due.

ABSTRACT

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 to the world's economy. It is a very large market and there are various companies working in the domain.

Housing prices are an important reflection of the economy, and housing price ranges are of great interest to both buyers and sellers. In this project, house prices will be predicted given explanatory variables that cover many aspects of residential houses. The goal of this project is to create a regression model that is able to accurately estimate the price of the house given the features.

TAKEAWAYS FROM THE BLOG

In this article, we do prediction using machine learning which leads to the below takeaways:

1. **EDA:** Learn the complete process of EDA
2. **Data analysis:** Learn to withdraw some insights from the dataset both mathematically and visualize it.
3. **Data visualization:** Visualizing the data to get better insight from it.
4. **Feature engineering:** We will also see what kind of stuff we can do in the feature engineering part.

PROBLEM STATEMENT:

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 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. This company wants to know:

• Which variables are important to predict the price of variables?

• How do these variables describe the price of the house?

ABOUT THE DATASET

About the data:

1. Number of data points in train data: 1168
2. Number of features in train data: 81
3. Number of data points in test data: 292
4. Number of features in test data: 80

We will be using two datasets, train data, and test data. This problem involves predicting the prices of the houses which are continuous and real-valued outputs. Thus, this is a **Regression Problem.**

Features:

Here's a brief version of what features is in the data description file:

1. MS Subclass: Identifies the type of dwelling involved in the sale.
2. MS Zoning: Identifies the general zoning classification of the sale.
3. Lot Frontage: Linear feet of street-connected to the property
4. Lot Area: Lot size in square feet
5. Street: Type of road access to the property
6. Alley: Type of alley access to the 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 the 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. SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.

81. ID

Importing Important Libraries:

We need some libraries to be imported to work upon the dataset, we would import the dataset by using pandas' read_csv method.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso,Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score, mean_squared_error
from scipy.stats import skew,norm

import warnings
warnings.filterwarnings("ignore")
```

Loading Data Set into a variable:

Here I am loading the training dataset into the variable df_train and the test dataset into the variable df_test.

```
In [2]:
pd.set_option('display.max_rows', None)
df_train = pd.read_csv("Project-Housing_splitted/train.csv")
df_test = pd.read_csv("Project-Housing_splitted/test.csv")
```

Here we have two datasets, one is train data which is having price column as target variable and 2nd is test data on which we have to predict the target.

Exploratory Data Analysis:

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. We performed some bi-variate analysis on the data to get a better overview of the data and to find outliers in our data-set. Outliers can occur due to some kind of errors while collecting the data and need to be removed so that it doesn't affect the performance of our model.

Checking shape of both the datasets:

```
In [5]: df_train.shape, df_test.shape

Out[5]: ((1168, 81), (292, 80))
```

The Train data contains 1168 rows and 81 columns (features), test data has 292 rows and 80 columns.

We also drop Id column from both the datasets as they are of no use in model prediction.

```
#Drop the ID column since it is unnecessary for the prediction process
df_train.drop("Id",axis =1,inplace = True)
df_test.drop("Id",axis =1,inplace= True)
```

Now train data has 80 columns and test data has 79 columns.

Getting detailed information about both the datasets:

We have two datasets.

Train data has 1168 observations and 80 columns including the target variable. The targetvariable is Saleprice which is of integer data type.

Test data has 292 observations and 79 columns.

After applying the info() function to both the datasets we get to know the data types of all the features and missing values of both the datasets.

We concluded that we have both numerical and categorical data types features in both datasets.

Also, there are so many missing values in both datasets. Some features have more than 80% to 90% missing values. Now we will check the percentage of missing values in both the datasets.

<u>Checking the missing values:</u>

For train data:

```
In [17]:  1  #For Train data
          2  df_train.isnull().sum()

Out[17]: MSSubClass          0
         MSZoning            0
         LotFrontage       214
         LotArea             0
         Street              0
         Alley            1091
         LotShape            0
         LandContour         0
         Utilities           0
         LotConfig           0
         LandSlope           0
         Neighborhood        0
         Condition1          0
         Condition2          0
         BldgType            0
         HouseStyle          0
         OverallQual         0
         OverallCond         0
         YearBuilt           0
         YearRemodAdd        0
         RoofStyle           0
         RoofMatl            0
         Exterior1st         0
         Exterior2nd         0
         MasVnrType          7
         MasVnrArea          7
         ExterQual           0
         ExterCond           0
         Foundation          0
         BsmtQual           30
         BsmtCond           30
         BsmtExposure       31
         BsmtFinType1       30
         BsmtFinSF1          0
         BsmtFinType2       31
         BsmtFinSF2          0
         BsmtUnfSF           0
         TotalBsmtSF         0
         Heating             0
         HeatingQC           0
         CentralAir          0
         Electrical          0
         1stFlrSF            0
         2ndFlrSF            0
         LowQualFinSF        0
         GrLivArea           0
         BsmtFullBath        0
         BsmtHalfBath        0
         FullBath            0
         HalfBath            0
         BedroomAbvGr        0
         KitchenAbvGr        0
         KitchenQual         0
         TotRmsAbvGrd        0
         Functional          0
         Fireplaces          0
         FireplaceQu       551
         GarageType         64
         GarageYrBlt        64
         GarageFinish       64
         GarageCars          0
         GarageArea          0
         GarageQual         64
         GarageCond         64
         PavedDrive          0
         WoodDeckSF          0
         OpenPorchSF         0
         EnclosedPorch       0
         3SsnPorch           0
         ScreenPorch         0
         PoolArea            0
         PoolQC           1161
         Fence             931
         MiscFeature      1124
         MiscVal             0
         MoSold              0
         YrSold              0
         SaleType            0
         SaleCondition       0
```

We can see that features like Alley,PoolQC,Fence,MiscFeatures has more than 80% of data missing. Let's check with test dataset.

. **For test data:**

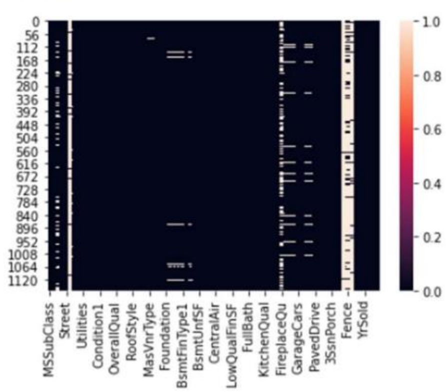
In [18]:
```
# For test data
df_test.isnull().sum()
```

Out[18]:
```
MSSubClass         0
MSZoning           0
LotFrontage       45
LotArea            0
Street             0
Alley            278
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         1
MasVnrArea         1
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual           7
BsmtCond           7
BsmtExposure       7
BsmtFinType1       7
BsmtFinSF1         0
BsmtFinType2       7
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         1
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        0
TotRmsAbvGrd       0
Functional         0
Fireplaces         0
FireplaceQu      139
GarageType        17
GarageYrBlt       17
GarageFinish      17
GarageCars         0
GarageArea         0
GarageQual        17
GarageCond        17
PavedDrive         0
WoodDeckSF         0
OpenPorchSF        0
EnclosedPorch      0
3SsnPorch          0
ScreenPorch        0
PoolArea           0
PoolQC           292
Fence            248
MiscFeature      282
MiscVal            0
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
```

Similarly, we have more than 80% of missing values in the same columns in test dataset also. We will treat the missing values.

In [18]:
```
sns.heatmap(df_train.isnull())
```

Out[18]:
```
<AxesSubplot:>
```



Treating the Missing values in the Train dataset and test dataset:

There are a few columns where null values may represent that that particular facility is not available in the house of those missing observations. So even if the % of missing values is high we will replace it with 'NOT AVAILABLE' instead of applying simple imputation on them or dropping them.

```
In [20]:
# have null columns with dtype = object
null_object_col = df_train.loc[:,df_train.isnull().sum() != 0].loc[:,df_train.loc[:,df_train.isnull().sum() != 0].dtypes ==
null_object_col

Out[20]: Index(['Alley', 'MasVnrType', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
       'BsmtFinType1', 'BsmtFinType2', 'FireplaceQu', 'GarageType',
       'GarageFinish', 'GarageQual', 'PoolQC', 'Fence', 'MiscFeature'],
      dtype='object')
```



```
In [22]:
# handle null values on object columns training set
for i in null_object_col:
    df_train[i].fillna('Not available', inplace=True)

# handle null values on object columns test set
for i in null_object_col:
    df_test[i].fillna('Not available', inplace=True)
```

For the rest categorical features' missing values, I have applied simple imputer(most_frequent) and for numerical values, I have replaced null with mean values

```
from sklearn.impute import SimpleImputer
imp= SimpleImputer(strategy="most_frequent")

df_test["Electrical"]= imp.fit_transform(df_test["Electrical"].values.reshape(-1,1))
df_train["GarageCond"]= imp.fit_transform(df_train["GarageCond"].values.reshape(-1,1))
df_test["GarageCond"]= imp.fit_transform(df_test["GarageCond"].values.reshape(-1,1))
```

```
# handle null values on LotFrontage columns on training set
df_train['LotFrontage'].fillna(df_train['LotFrontage'].mean(), inplace=True)
df_train["GarageYrBlt"].fillna(df_train["GarageYrBlt"].mean(), inplace=True)
df_train["MasVnrArea"].fillna(df_train["MasVnrArea"].mean(), inplace=True)


# handle null values on LotFrontage columns on test set
df_test['LotFrontage'].fillna(df_test['LotFrontage'].mean(), inplace=True)
df_test["GarageYrBlt"].fillna(df_test["GarageYrBlt"].mean(), inplace=True)
df_test["MasVnrArea"].fillna(df_test["MasVnrArea"].mean(), inplace=True)
```

```
df_train.isnull().sum().sum()
```

```
0
```

```
df_test.isnull().sum().sum()
```

```
0
```

Now there is no null values in both the datasets.

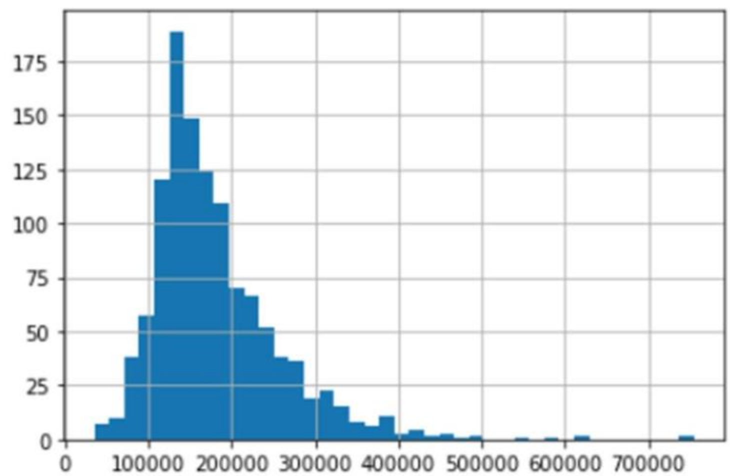
Univariate Analysis:

Uni means one, so in other words, the data has only one variable. Univariate data requires analysing each variable separately. It doesn't deal with causes or relationships (unlike regression) and its major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.

Analyzing Target variable:

```
#histogram
df_train['SalePrice'].hist(bins = 40)
```

```
<AxesSubplot:>
```




```
#descriptive statistics summary
df_train['SalePrice'].describe()
```

```
count      1168.000000
mean     181477.005993
std       79105.586863
min       34900.000000
25%      130375.000000
50%      163995.000000
75%      215000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

We can clearly see that the target variable has a normal distribution that is skewed towards the left. Now let's calculate the Skewness and Kurtosis:

```
#skewness & kurtosis
print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

```
Skewness: 1.953878
Kurtosis: 7.390657
```

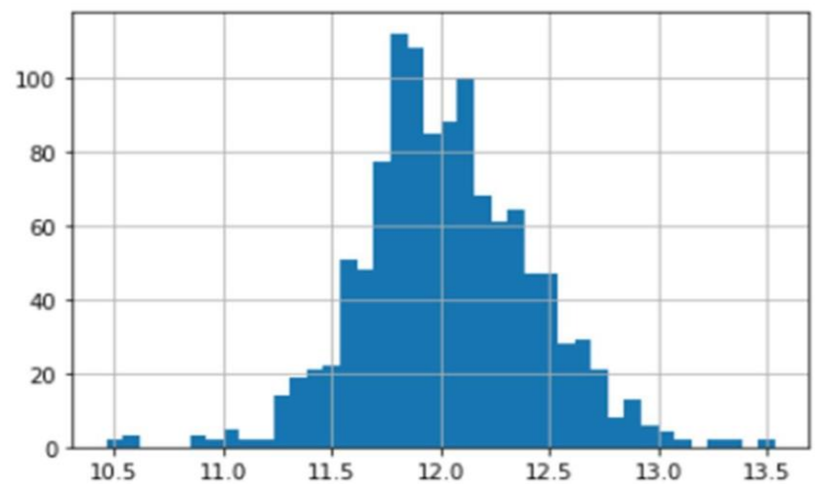
As we saw before, the target variable "SalePrice" is not uniformly distributed and it's skewed towards the left. Therefore**, we will try to use the log transformation to remove the skewness**.

```
#log transform the target
df_train["SalePrice"] = np.log1p(df_train["SalePrice"])

#histogram
df_train['SalePrice'].hist(bins = 40)
```


```
df_train['SalePrice'].hist(bins = 40)
```


```
<AxesSubplot:>
```

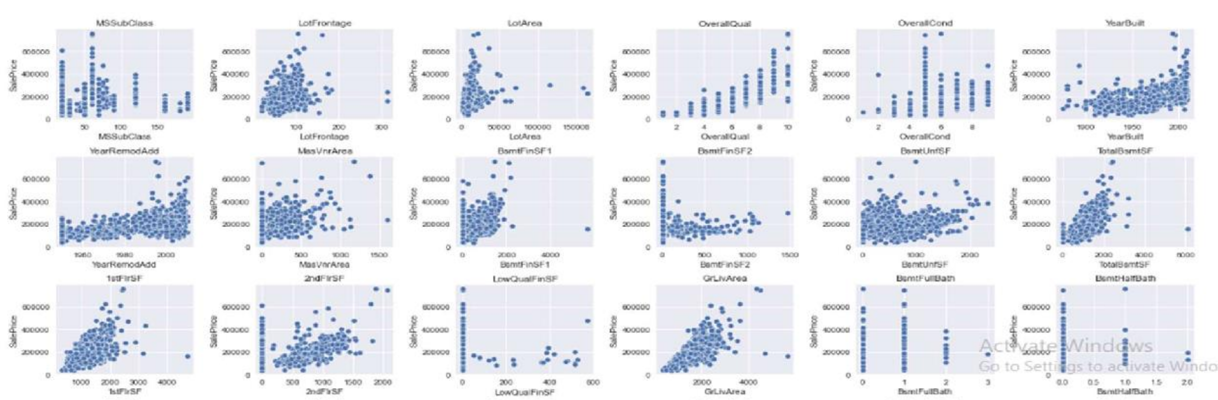


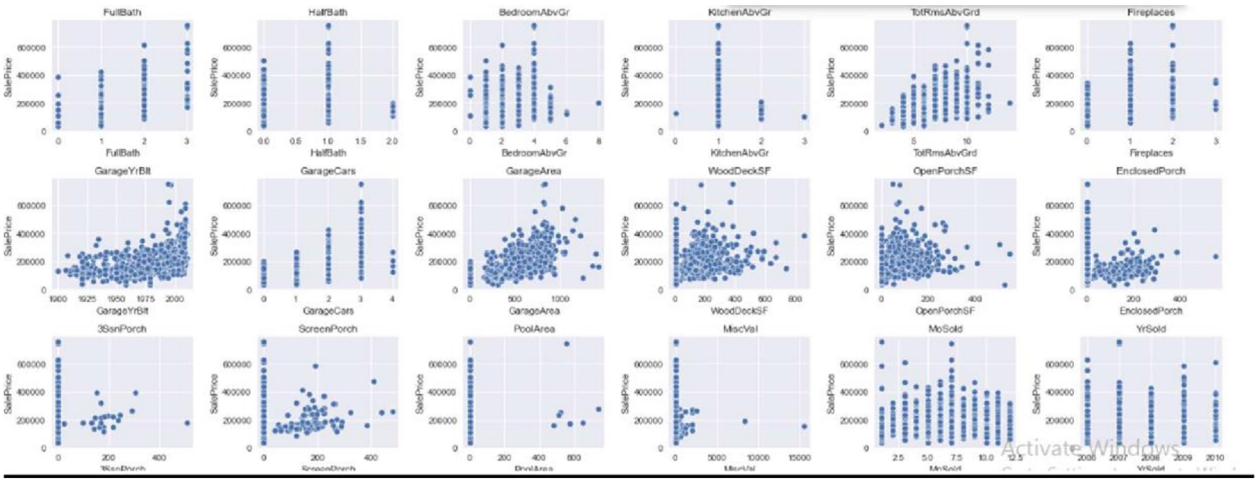

```
#skewness & kurtosis
print("Skewness: %f" % df_train['SalePrice'].skew())
print("Kurtosis: %f" % df_train['SalePrice'].kurt())
```

```
Skewness: 0.073610
Kurtosis: 0.995996
```

Now both the skewness and kurtosis are removed in the target variable.This looks almost normal distribution with a **Skew of 0.073610.**

Analysing Numerical columns:





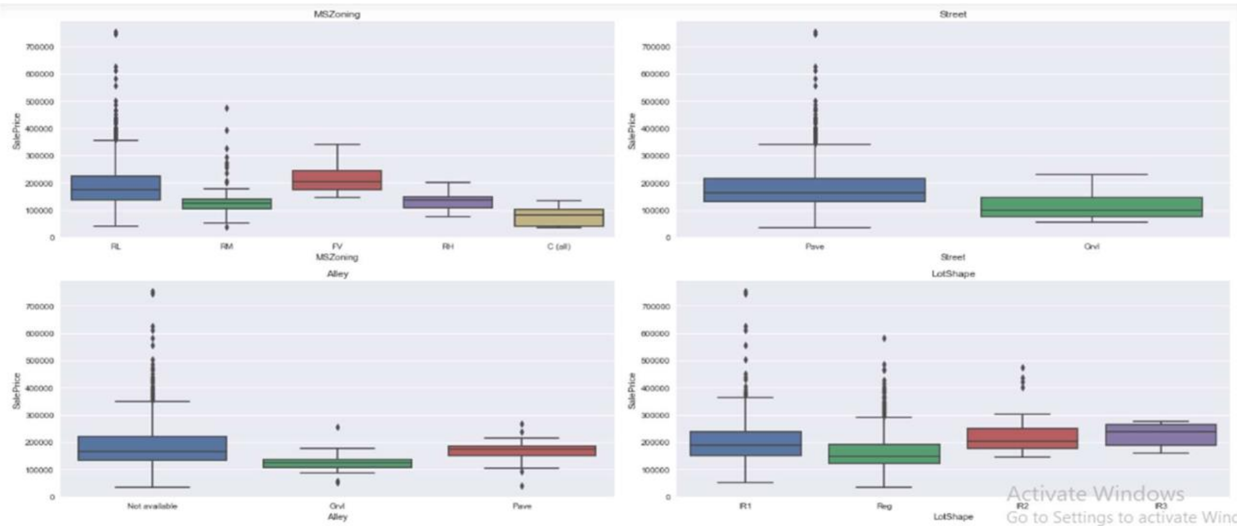
Here in all the numerical columns we can see through scatter plot that many numerical features are skewed on either the left side or right side. Also, there are so many outliers in the features.

Analysing Categorical Columns:

Through bar charts of different categorical columns, we can see unique values and the value counts for categorical features.

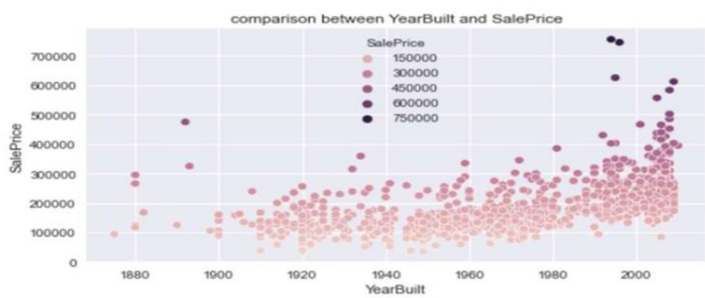
Bivariate Analysis:

Bivariate analysis is finding some kind of empirical relationship between two variables. Specifically, the dependent vs independent Variables




```
1  plt.figure(figsize=[8,4])
2  plt.title("comparison between YearBuilt and SalePrice")
3  sns.scatterplot(df_train['YearBuilt'],df_train['SalePrice'],hue=df_train["SalePrice"])
```

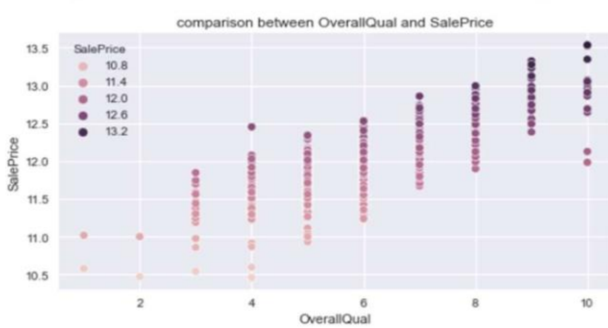
```
<AxesSubplot:title={'center':'comparison between YearBuilt and SalePrice'}, xlabel='YearBuilt', ylabel='SalePrice'>
```



Sale prices of houses were less in the past. But with time price increased.

```
In [39]:  1  plt.figure(figsize=[8,4])
          2  plt.title("comparison between OverallQual and SalePrice")
          3  sns.scatterplot(df_train['OverallQual'],df_train['SalePrice'],hue=df_train["SalePrice"])
```

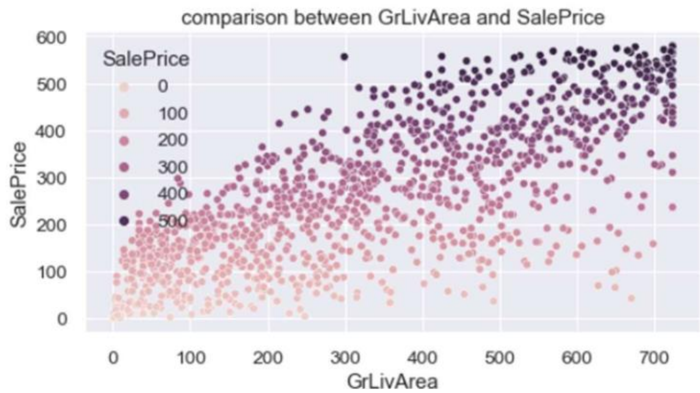
```
Out[39]: <AxesSubplot:title={'center':'comparison between OverallQual and SalePrice'}, xlabel='OverallQual', ylabel='SalePrice'>
```



Saleprice of houses increase with increase in overall quality and ratings of materials used.

```
plt.figure(figsize=[8,4])
plt.title("comparison between GrLivArea and SalePrice")
sns.scatterplot(df_train['GrLivArea'],df_train['SalePrice'],hue=df_train["SalePrice"])
```

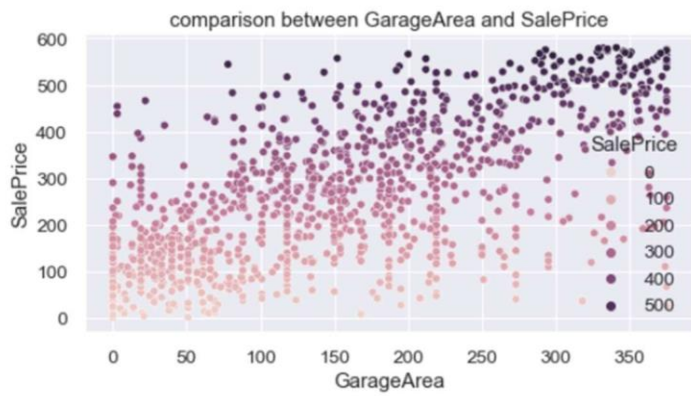
```
<AxesSubplot:title={'center':'comparison between GrLivArea and SalePrice'}, xlabel='GrLivArea', ylabel='SalePrice'>
```



The more is the Ground living area, the more is the sale price.

```
plt.figure(figsize=[8,4])
plt.title("comparison between GarageArea and SalePrice")
sns.scatterplot(df_train['GarageArea'],df_train['SalePrice'],hue=df_train["SalePrice"])
```

```
<AxesSubplot:title={'center':'comparison between GarageArea and SalePrice'}, xlabel='GarageArea', ylabel='SalePrice'>
```



The more is the garage area, the more is the sale price.

Checking for Correlation with Output Features:

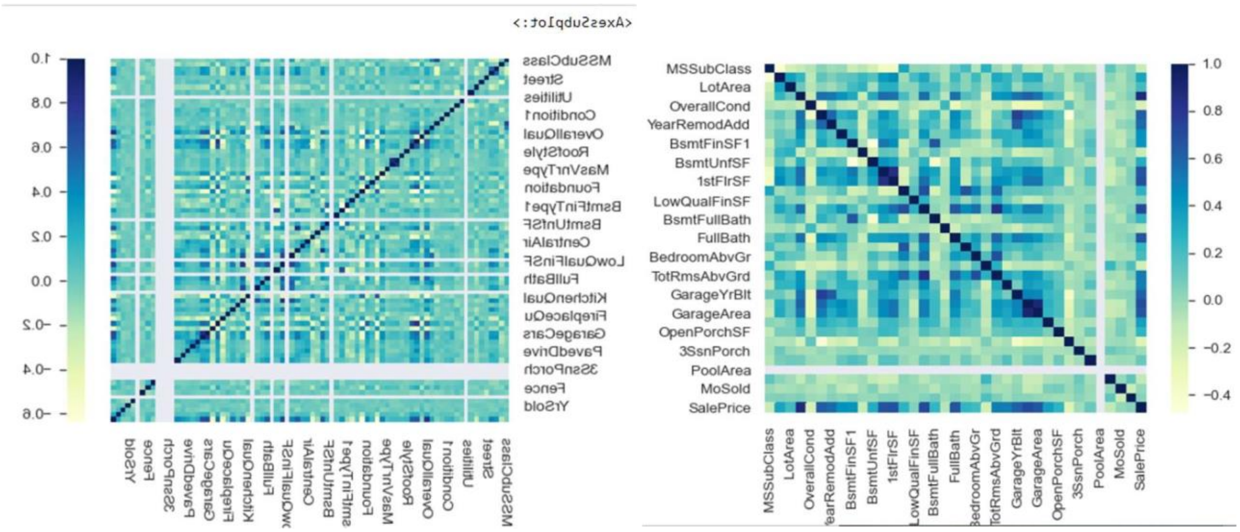
After this, we found the most important features relative to the target by building a correlation matrix. **A correlation matrix** is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The correlation coefficient has values **between -1 to 1.**

Correlations are very useful in many applications, especially when conducting regression analysis.

Multicollinearity:

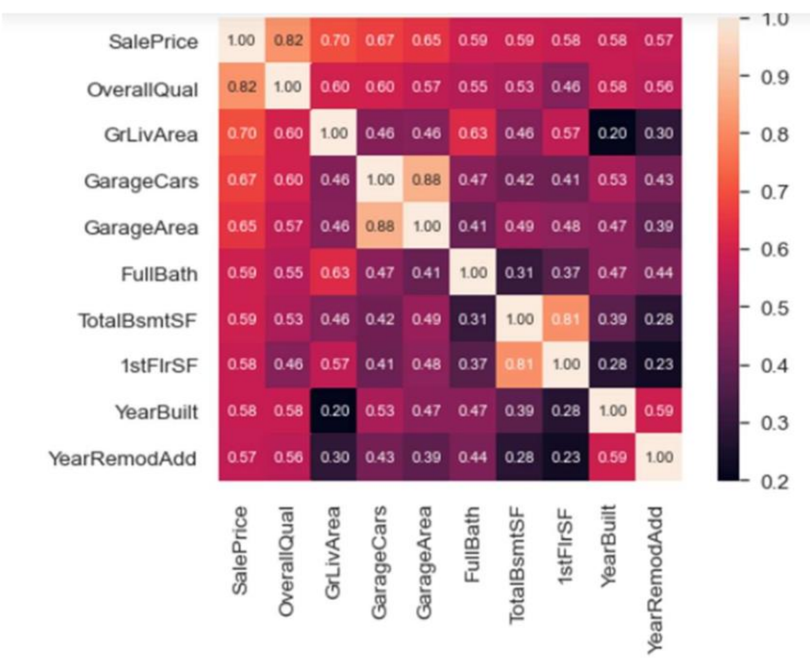
Multicollinearity is **a statistical concept where several independent variables in a model are correlated**. Two variables are considered to be perfectly collinear if their correlation coefficient is +/- 1.0. Multicollinearity among independent variables will result in less reliable statistical inferences.

Let’s check the correlation and multicollinearity through correlation heat map.



Left : For Train Data :: Right: For Test Data

Top 10 highly correlated columns with the target column:



From this I could only tell that OverallQual, TotalBsmtSF, GarageCars, GarageArea have a high positive correlation with SalePrice.

Observation:

We can see that there isn't much correlation among the input features . Thus there is no Multicollinearity among the features. This is good.

Few Features have **greater** than 0.5 Pearson Correlation with output feature.

A value closer to 0 implies weaker correlation (exact 0 implying no correlation)

A value closer to 1 implies stronger positive correlation

A value closer to -1 implies stronger negative correlation.

Separating Independent and Dependent (target) features from Train Data:

```
X = df_train.drop('SalePrice', axis=1)
y = df_train['SalePrice'].values
```

Check for Outliers:

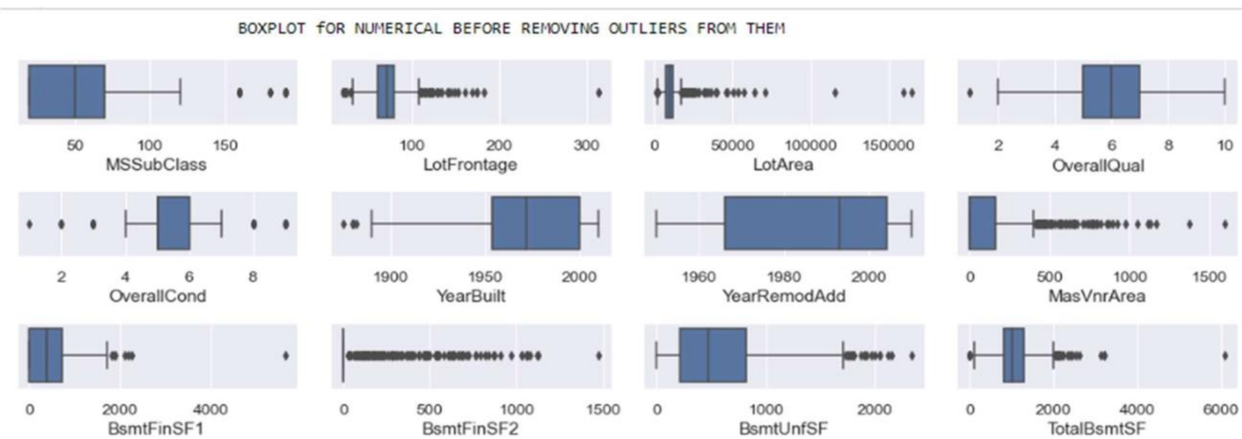
An outlier is **a data point that is noticeably different from the rest**. They represent errors in measurement, bad data collection, or simply show variables not considered when collecting the data. **A value that "lies outside" (is much smaller or larger than) most of the other values in a set of data**.

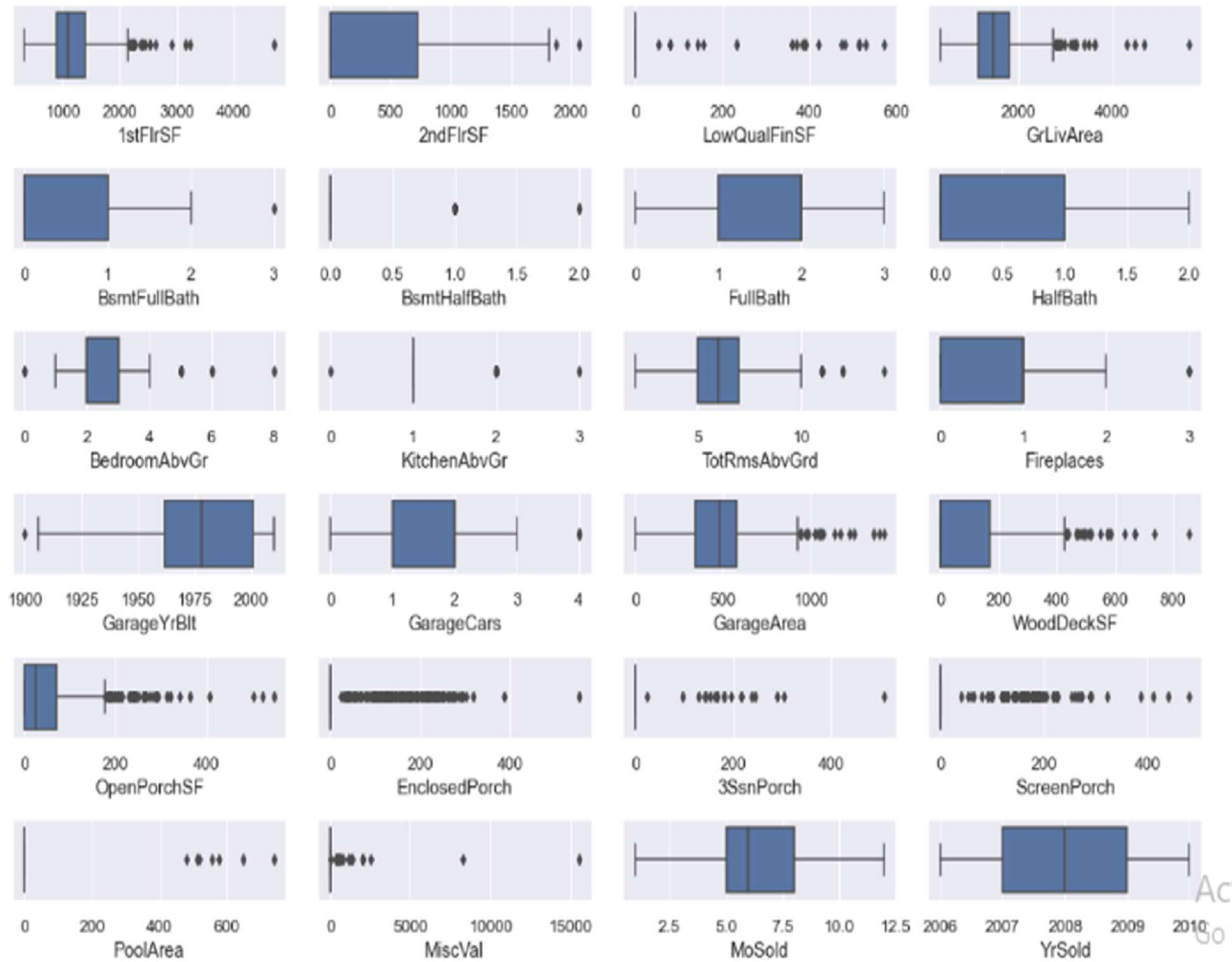
Box Plot

This **is the visual representation of the depicting groups of numerical data through their quartiles.** Boxplot **is also used for detecting the outlier in the data set.**

I used a box plot in this dataset because It captures the summary of the data efficiently with a simple box and whiskers and allows me to compare easily across groups.

Before Removing Outliers





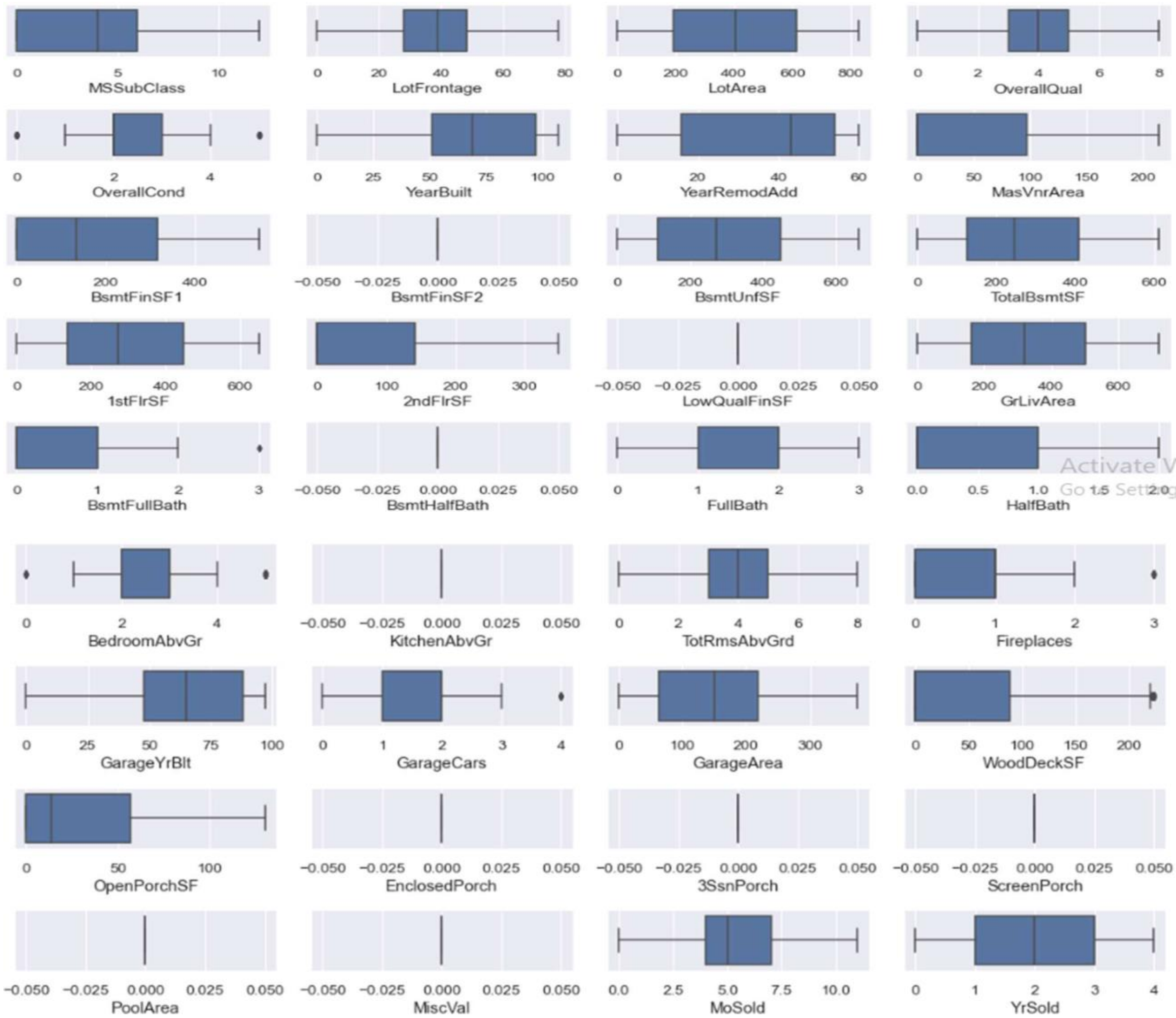
<u>Removing Outliers using interquartile:</u>

Removing Outliers from Numerical Columns

```
def removeOutliers(new_num_var):
    global df_train
    for i in range(len(new_num_var)):
        q1 = df_train[new_num_var[i]].quantile(0.25)
        q3 = df_train[new_num_var[i]].quantile(0.75)
        IQR = q3-q1
        minimum = q1 - 1.5 * IQR
        maximum = q3 + 1.5 * IQR
        df_train.loc[(df_train[new_num_var[i]] <= minimum), new_num_var[i]] = minimum
        df_train.loc[(df_train[new_num_var[i]] >= maximum), new_num_var[i]] = maximum
```

```
removeOutliers(new_num_var)
```

<u>After Removing outliers</u>:



Here we can see that all thee outliers are removed.

Converting categorical columns to numerical columns

We will convert categorical columns into numerical columns using label encoder for further analysis.

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
#For train data
df_train=df_train.apply(LabelEncoder().fit_transform)
print('For train data')
print(df_train.head())
print('\n\n')
#For test data
df_test_le=df_test.apply(LabelEncoder().fit_transform)
print('For test data')
print(df_test_le.head())
```

```
For train data
   MSSubClass  MSZoning  LotFrontage  LotArea  Street  Alley  LotShape  \
0          11         3           39       76       1      1         0
1           0         3           64      804       1      1         0
2           5         3           61      445       1      1         0
3           0         3           74      628       1      1         0
4           0         3           39      817       1      1         0

   LandContour  Utilities  LotConfig  ...  PoolArea  PoolQC  Fence  \
0            3          0          4  ...         0       3      4
1            3          0          4  ...         0       3      4
2            3          0          1  ...         0       3      4
3            3          0          4  ...         0       3      2
4            3          0          2  ...         0       3      4

   MiscFeature  MiscVal  MoSold  YrSold  SaleType  SaleCondition  SalePrice
0            1        0       1       1         8              4        129
1            1        0       9       1         8              4        468
2            1        0       5       1         8              4        470
3            1        0       0       4         0              4        326
4            1        0       5       3         8              4        379

[5 rows x 80 columns]
```

```
For test data
   MSSubClass  MSZoning  LotFrontage  LotArea  Street  Alley  LotShape  \
0           0         2           48      215       1      1         0
1          11         2           31       34       1      1         0
2           0         2           31      180       1      1         3
3           6         2           40      185       1      1         3
4           5         2           48      220       1      1         0

   LandContour  Utilities  LotConfig  ...  ScreenPorch  PoolArea  PoolQC
0            1          0          0  ...            0         0       0
1            3          0          1  ...            0         0       0
2            3          0          4  ...            0         0       0
3            0          0          4  ...            0         0       0
4            3          0          1  ...            0         0       0

   Fence  MiscFeature  MiscVal  MoSold  YrSold  SaleType  SaleCondition
0      4            0        0       6       1         5              2
1      4            0        0       7       3         0              0
2      4            0        0       5       3         5              2
3      4            0        0       6       3         5              2
4      4            0        0       0       2         5              2

[5 rows x 79 columns]
```

We can see all the categorical features in both train data and test data is converted into numerical form.

Features Scaling / Standard Scaler:

Feature Scaling is **a technique to standardize the independent features present in the data in a fixed range**. It is performed during the data pre-processing to handle highly varying magnitudes or values or units.

```
X = df_train.drop('SalePrice', axis=1)
y = df_train['SalePrice'].values
```

```
# Performing Standard scaler
#For train data
sc = StandardScaler()
X = sc.fit_transform(X)

#For test data
price_test = sc.fit_transform(df_test_le)
```

```
X
```

```
array([[ 1.70759409, -0.02164599,  0.02879392, ..., -0.60548713,
         0.33003329,  0.20793187],
       [-1.02115826, -0.02164599,  1.45626837, ..., -0.60548713,
         0.33003329,  0.20793187],
       [ 0.21918372, -0.02164599,  1.28497144, ..., -0.60548713,
         0.33003329,  0.20793187],
       ...,
       [ 1.95566248, -0.02164599, -2.19806622, ...,  0.8992128 ,
         0.33003329,  0.20793187],
       [ 0.46725211, -4.76211672, -1.17028462, ...,  0.14686284,
         0.33003329,  0.20793187],
       [ 0.21918372, -0.02164599,  0.02879392, ..., -1.3578371 ,
         0.33003329,  0.20793187]])
```

By using a standard scaler, I have scaled the data in one range.

Building Machine Learning Models:

First, I will find the best random state on which I will get the maximum score.

Finding Best Random State

```
maxScore = 0
maxRS = 0

for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=i)
    lr = LinearRegression()
    lr.fit(x_train,y_train)
    pred_train = lr.predict(x_train)
    pred_test = lr.predict(x_test)
    acc=r2_score(y_test,pred_test)
    if acc>maxScore:
        maxScore=acc
        maxRS=i
print('Best score is',maxScore,'on Random State',maxRS)
```

```
Best score is 0.9240826608036915 on Random State 84
```

Applying train-test split with Best Random State and applying ML on Different Algorithms:

Checking with different algorithms

```
model = [LinearRegression(),Lasso(alpha=1.0),Ridge(alpha=1.0),DecisionTreeRegressor(criterion='squared_error'),
         KNeighborsRegressor()]
for i in model:
    X_train1,X_test1,y_train1,y_test1 = train_test_split(X,y, test_size = 0.3, random_state =maxRS)
    i.fit(X_train1,y_train1)
    pred = i.predict(X_test1)
    print('Train Score of', i , 'is:' , i.score(X_train1,y_train1))
    print("r2_score", r2_score(y_test1, pred))
    print("mean_squred_error", mean_squared_error(y_test1, pred))
    print("RMSE", np.sqrt(mean_squared_error(y_test1, pred)),"\n")
```

```
Train Score of LinearRegression() is: 0.8949500885534561
r2_score 0.9240826608036915
mean_squred_error 1818.6729876007134
RMSE 42.645902354161926

Train Score of Lasso() is: 0.8917401227582257
r2_score 0.9261040022874991
mean_squred_error 1770.249805304877
RMSE 42.074336659118906

Train Score of Ridge() is: 0.8949527937715197
r2_score 0.9242053679462536
mean_squred_error 1815.7334198033445
RMSE 42.61142358339304

Train Score of DecisionTreeRegressor() is: 1.0
r2_score 0.7182627273349806
mean_squred_error 6749.28774928775
RMSE 82.15404889162646

Train Score of KNeighborsRegressor() is: 0.8761395853867664
r2_score 0.8369131698871278
mean_squred_error 3906.9021082621084
RMSE 62.505216648389506
```

Conclusions:

Have checked Multiple Model and their score also. I have found that the Decision tree regressor model is overfitting. Other models are working well. But lasso regression is having less train and test score difference with least mean square error and least RMSE. Now I will check with the ensemble method to boost up score.

Using Ensemble Technique to boost up score:

RandomForestRegressor:

```
from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(n_estimators=100,random_state=maxRS,criterion='squared_error', min_samples_split=2, min_samples_lea
#RandomForestClassifier(100)---Default
rf.fit(X_train1,y_train1)
predrf=rf.predict(X_test1)
print('Train Score of', rf , 'is:' , rf.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predrf))
print("mean_squred_error", mean_squared_error(y_test1, predrf))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predrf)))
```

```
Train Score of RandomForestRegressor(random_state=84) is: 0.9813767323691924
r2_score 0.8930452761631832
mean_squred_error 2562.203433333333
RMSE 50.61821246679236
```

There is much difference between train score and test score. so, the model is overfitting.

AdaBoostRegressor:

```
from sklearn.ensemble import AdaBoostRegressor

ABr=AdaBoostRegressor( base_estimator=Lasso(),n_estimators=50,learning_rate=1.0,loss='linear',random_state=maxRS,)
#RandomForestClassifier(50)---Default
ABr.fit(X_train1,y_train1)
predAbr=ABr.predict(X_test1)
print('Train Score of', ABr , 'is:' , ABr.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predAbr))
print("mean_squred_error", mean_squared_error(y_test1, predAbr))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predAbr)))
```

```
Train Score of AdaBoostRegressor(base_estimator=Lasso(), random_state=84) is: 0.8622186362608182
r2_score 0.8681156446260296
mean_squred_error 3159.4167701999227
RMSE 56.208689454566745
```

Here we can see the train and test score is having less difference and the model is working well.

GradientBoostingRegressor:

```
from sklearn.ensemble import GradientBoostingRegressor

Gradient_Boost=GradientBoostingRegressor(n_estimators=100,loss='squared_error',learning_rate=0.1,criterion='friedman_mse', m
#GradientBoostingRegressor(100)---Default
Gradient_Boost.fit(X_train1,y_train1)
predgb=Gradient_Boost.predict(X_test1)
print('Train Score of', Gradient_Boost , 'is:' , Gradient_Boost.score(X_train1,y_train1))
print("r2_score", r2_score(y_test1, predgb))
print("mean_squred_error", mean_squared_error(y_test1, predgb))
print("RMSE", np.sqrt(mean_squared_error(y_test1, predgb)),"\n")
```

```
Train Score of GradientBoostingRegressor() is: 0.9679385176546844
r2_score 0.9188795601235165
mean_squred_error 1943.3182762656959
RMSE 44.08308378806655
```

Conclusion: I have found that AdaBoostRegressor() is working well on the dataset with least train score and test score difference and have given less RMSE score . So i am selecting AdaBoostRegressor for the final Model.

Hyper Parameter Tuning:

Hyperparameter tuning (or hyperparameter optimization) is the process of determining the right combination of hyperparameters that maximizes the model performance. It works by running multiple trials in a single training process.

We are using Randomsearchcv method for hyperparameter tuning to find best parameters for AdaBoostRegressor.

```
Ada_Boost = AdaBoostRegressor()
Para ={'n_estimators' : [50, 100, 150, 200],
        'learning_rate' : [0.001, 0.01, 0.1, 1],
        'loss' : ["linear", "square", "exponential"],
        'random_state' : [21, 42, 104, 111]
        }
Rand_search = RandomizedSearchCV(Ada_Boost,Para,cv = 5,scoring = "r2",n_jobs =-1,verbose = 2)
Rand_search.fit(X_train1,y_train1)
print(Rand_search.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
{'random_state': 21, 'n_estimators': 200, 'loss': 'linear', 'learning_rate': 1}
```

```
prediction = Rand_search.predict(X_test1)
```

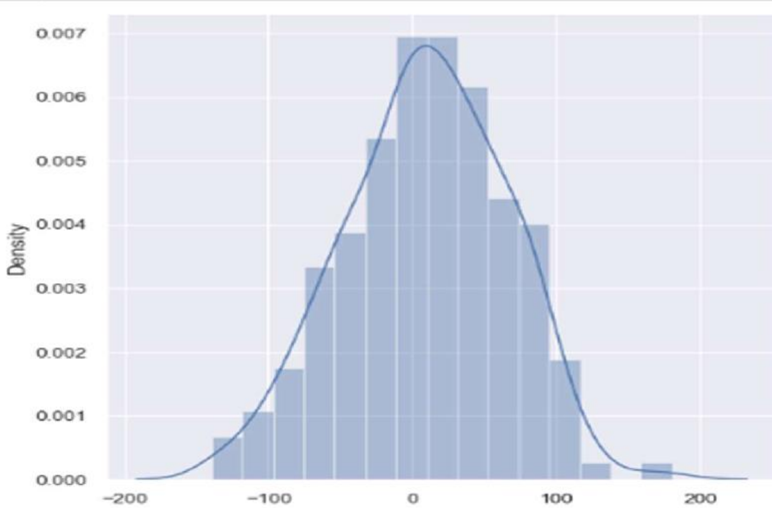
```
SalePrice = AdaBoostRegressor(n_estimators= 200, loss= 'linear', learning_rate =1, random_state=21)
SalePrice.fit(x_train, y_train)
pred = SalePrice.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print("RMSE value:",np.sqrt(mean_squared_error(y_test, pred)))
```

```
R2_Score: 86.21189353986094
RMSE value: 56.714676962897954
```

The predicted y value is having a normalized curve which is good.

Plotting the residuals.

```
plt.figure(figsize = (8,8))
sns.distplot(y_test1-prediction)
plt.show()
```



As we can see that most of the residuals are 0, which means our model is generalizing well.

Cross Validation:

Cross-validation is **a resampling method that uses different portions of the data to test and train a model on different iterations**. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice.

```
best_Ada_Boost = AdaBoostRegressor(n_estimators= 100, loss= 'exponential', learning_rate =1, random_state=104)

for i in range(2,11):
    cross_score = cross_val_score(best_Ada_Boost,X,y,cv = i,n_jobs = -1)
    print(i,"mean",cross_score.mean() ,"and STD" , cross_score.std())
```

```
2 mean 0.8443214554419438 and STD 0.009555629109518848
3 mean 0.8492086515080733 and STD 0.006408722736634356
4 mean 0.8476405781779779 and STD 0.010834588125154383
5 mean 0.8494251839238531 and STD 0.011282459225452415
6 mean 0.847931096299165 and STD 0.015742375342086924
7 mean 0.845274131430127 and STD 0.021166154396682322
8 mean 0.8469293653677799 and STD 0.024957283297494973
9 mean 0.8476671999403956 and STD 0.027442190645623893
10 mean 0.8496127177699639 and STD 0.02682516694571256
```

Applying Cross validation Score=5

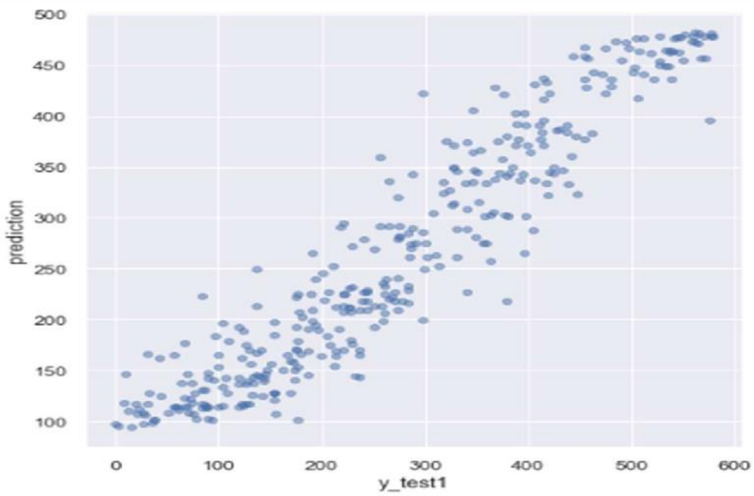
```
# Cross validate of GradientBoostingRegressor using cv=5
from sklearn.model_selection import cross_val_score
score=cross_val_score(best_Ada_Boost,X,y,cv=5,scoring='r2')
print('Score:', score)
print('Mean Score:', score.mean())
print('Standard Deviation:', score.std())
```

```
Score: [0.8547881  0.85718145 0.83800455 0.86299249 0.83415933]
Mean Score: 0.8494251839238531
Standard Deviation: 0.011282459225452415
```

Plotting y_test1 vs predictions:

- Simply plotting our predictions vs the true values.
- Ideally, it should be a straight line.

```
plt.figure(figsize = (8,8))
plt.scatter(y_test1, prediction, alpha = 0.5,)
plt.xlabel("y_test1")
plt.ylabel("prediction")
plt.show()
```



Saving the Model:

We are saving the model by using python's pickle library. It will be used further for the predictio n. Also, we have loaded the prediction file to predict the target of the test data.

```
import pickle
# Saving the AdaBoostRegressor
best_Ada_Boost.fit(X,y)
pred = best_Ada_Boost.predict(price_test)

# Saving model

filename = "House_Saleprice_Prediction.pkl"

with open(filename,"wb") as f:
    pickle.dump(best_Ada_Boost,f)
```

```
loaded_model=pickle.load(open('House_Saleprice_Prediction.pkl','rb'))
```

Predicting the Target of Test Data:

Predicting target on standard scalled test dataset

```
Test_pred=loaded_model.predict(price_test)
Y_tst=pd.DataFrame(data=Test_pred)

Y_tst
```

	0
0	474.281250
1	415.678815
2	445.973958
3	284.173469
4	311.841530
5	98.610778
6	165.640000
7	432.096296
8	420.918773
9	251.873563
10	86.751111

```
df_test.shape, Y_tst.shape
```

```
((292, 79), (292, 1))
```

Here we have predicted the target of test dataset and checked the shape of test dataset and target of test dataset to join them.

Preparing the Final Test data with Target Column:

```
1 df_test['SalePrice']=Y_tst
2 df_test
```

Alley	LotShape	LandContour	Utilities	LotConfig	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
Not ilable	IR1	HLS	AllPub	Corner	...	0	Not available	Not available	Not available	0	7	2007	WD	Normal	474.281250
Not ilable	IR1	Lvl	AllPub	CulDSac	...	0	Not available	Not available	Not available	0	8	2009	COD	Abnorml	415.678815
Not ilable	Reg	Lvl	AllPub	Inside	...	0	Not available	Not available	Not available	0	6	2009	WD	Normal	445.973958
Not ilable	Reg	Bnk	AllPub	Inside	...	0	Not available	Not available	Not available	0	7	2009	WD	Normal	284.173469
Not ilable	IR1	Lvl	AllPub	CulDSac	...	0	Not available	Not available	Not available	0	1	2008	WD	Normal	311.841530
Not ilable	Reg	Lvl	AllPub	Inside	...	0	Not available	MnPrv	Not available	0	12	2007	WD	Normal	96.610778
Not ilable	Reg	Lvl	AllPub	Inside	...	0	Not available	Not available	Not available	0	5	2006	WD	Normal	165.640000
Not ilable	Reg	Lvl	AllPub	Inside	...	0	Not available	Not available	Not available	0	1	2008	New	Partial	432.096296
Not ilable	Reg	Low	AllPub	Inside	...	0	Not available	Not available	Not available	0	8	2009	WD	Normal	420.918773
Not ilable	Reg	Lvl	AllPub	FR2	...	0	Not available	Not available	Not available	0	6	2009	WD	Normal	251.873563
Not ilable	IR1	Lvl	AllPub	Inside	...	0	Not available	Not available	Not available	0	5	2008	WD	Normal	86.751111

CONCLUSION:

So, as we saw that we have done a complete EDA process, getting data insights, feature engineering, and data visualization as well so after all these steps one can go for the prediction using machine learning model-making steps.

We have training and test file separately available with us. we have both numerical and categorical data types features in both datasets and the dependent variable of train data i.e. the price is the numerical data type. So, I applied the regression method for prediction.

Once data has been cleaned for both test and train datasets, Label encoding is applied to them to convert them into Numerical ones. I trained the model on five different algorithms but for most of the models, train and test data was having a high variance, and the model was overfitting.

Only Ada Boost regressor worked well out of all the models, as there was less difference between train score and test score and RMSE was also low hence I used it as the final model and have done further processing.

After applying hyperparameter tuning I got an accuracy(r2_score) of 86% from the Ada Boost Regressor model which is a good score. Then I applied that score to the test dataset to get the target variable which is price.

I hope this article helped you to understand Data Analysis, Data Preparation, and Model building approaches in a much simpler way.

<u>Thank you</u>