HOUSING PRICE PREDICTION PROJECT

Submitted by:
NIPAM GOGOI

ACKNOWLEDGMENT

I would like to thank Flip Robo Technologies for providing me with the opportunity to work on this project from which I have learned a lot. I am also grateful to Miss Khushboo Garg for her constant guidance and support.

INTRODUCTION

BUSINESS PROBLEM FRAMING

Houses are one of the necessary need 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.

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 variable?
- How do these variables describe the price of the house?

BUSSINESS GOAL

The company wants to enter the Australian Market and hence are looking at prospective properties to buy. They want to understand what are the factors affecting the prices and how exactly are those factors influencing it. The company would then manipulate the strategy of the firm and concentrate on areas that will yield high return.

In this case study, we have been identified by Surprise Housing, an US based housing company which uses data analytics to purchase houses at a price below their actual values and flip them at a higher price.

REVIEW OF LITERATURE AND APPROACH

This study proposes a performance comparison between machine learning regression algorithms and Artificial Neural Network (ANN). The regression algorithms used in this study are Multiple linear, Least Absolute Selection Operator (Lasso), Ridge, Random Forest. Moreover, this study attempts to analyse the correlation between variables to determine the most important factors that affect house prices in Malmö, Sweden. There are two datasets used in this study which called public and local. They contain house prices from Ames, Iowa, United States and Malmö, Sweden, respectively. The accuracy of the prediction is evaluated by checking the root square and root mean square error scores of the training model. The test is performed after applying the required pre-processing methods and splitting the data into two parts. However, one part will be used in the training and the other in the test phase. We have also presented a binning strategy that improved the accuracy of the models. This thesis attempts to show that Lasso gives the best score among other algorithms when using the public dataset in training. The correlation graphs show the variables' level of dependency. In addition, the empirical results show that crime, deposit, lending, and repo rates influence the house prices negatively. Where inflation, year, and unemployment rate impact the house prices positively. Keywords Multiple linear regression, Lasso Regression, Ridge Regression, Random Forest Regression, Artificial Neural Network, Machine Learning.

Multiple Linear Regression (MLR) is a supervised technique used to estimate the relationship between one dependent variable and more than one independent variables. Identifying the correlation and its cause-effect helps to make predictions by using these relations [4]. To estimate these relationships, the prediction accuracy of the model is essential; the complexity of the model is of more interest. However, Multiple Linear Regression is prone to many problems such as multicollinearity, noises, and overfitting, which effect on the prediction accuracy. Regularised regression plays a significant part in Multiple Linear Regression because it helps to reduce variance at the cost of introducing some bias, avoid the overfitting problem and solve ordinary least squares (OLS) problems. There are two types of regularisation techniques L1 norm (least absolute deviations) and L2 norm (least squares). L1 and L2 have different cost functions regarding model complexity [5].

Lasso Regression Least Absolute Shrinkage and Selection Operator (Lasso) is an L1-norm regularised regression technique that was formulated by Robert Tibshirani in 1996 [6]. Lasso is a powerful technique that performs regularisation and feature selection. Lasso introduces a bias term, but instead of squaring the slope like Ridge regression, the absolute value of the slope is added as a penalty term. Lasso is defined as: $L = Mi(sum\ of\ squared\ residuals + \alpha * |slope|)$ (1) Where $Min(sum\ of\ squared\ residuals)$ is the Least Squared Error, and $\alpha * |slope|$ is the penalty term. However, alpha α is the tuning parameter which controls the strength of the penalty term. In other words, the tuning parameter is the value of shrinkage. $|slope|$ is the sum of the absolute value of the coefficients [7]. Cross-validation is a technique that is used to compare different machine learning algorithms in order to observe how these methods will perform in practice. Cross-validation method divides the data into blocks. Each block at a time will be used for testing by the algorithm, and the other blocks will be used for training the model. In the end, the results will be summarised, and the block that performs best will be chosen as a testing block [8]. However, α is determined 4 by using cross-validation. When $\alpha = 0$, Lasso becomes Least Squared Error, and when $\alpha \neq 0$, the magnitudes are considered, and that leads to zero coefficients. However, there is a reverse relationship between alpha α and the upper bound of the sum of the coefficients t. When $t \rightarrow \infty$, the tuning parameter $\alpha = 0$. Vice versa when $t = 0$ the coefficients shrink to zero and $\alpha \rightarrow \infty$ [7]. Therefore, Lasso helps to assign zero weights to most redundant or irrelevant features in order to enhance the prediction accuracy and interpretability of the regression model. Throughout the process of features selection, the variables that still have non-zero coefficients after the shrinking process are selected to be part of the regression model [7]. Therefore, Lasso is powerful when it comes to feature selection and

reducing the overfitting.

Ridge Regression: The Ridge Regression is an L2-norm regularised regression technique that was introduced by Hoerl in 1962 [9]. It is an estimation procedure to manage collinearity without removing variables from the regression model. In multiple linear regression, the multicollinearity is a common problem that leads least square estimation to be unbiased, and its variances are far from the correct value. Therefore, by adding a degree of bias to the regression model, Ridge Regression reduces the standard errors, and it shrinks the least square coefficients towards the origin of the parameter space [10]. Ridge formula is: $R = Mi(sum\ of\ squared\ residuals + \alpha * slope\ 2$) (2) Where $Min(sum\ of\ squared\ residuals)$ is the Least Squared Error, and $\alpha * slope$ 2 is the penalty term that Ridge adds to the Least Squared Error. When Least Squared Error determines the values of parameters, it minimises the sum of squared residuals. However, when Ridge determines the values of parameters, it reduces the sum of squared residuals. It adds a penalty term, where α determines the severity of the penalty and the length of the slope. In addition, increasing the α makes the slope asymptotically close to zero. Like Lasso, α is determined by applying the Cross-validation method. Therefore, Ridge helps to reduce variance by shrinking parameters and make the prediction less sensitive.

ANALYTICAL PROBLEM FRAMING

Dataset description

Data contains 1460 entries each having 81 variables.

• Data contains Null values.

• Extensive EDA has to be performed to gain relationships of important variable and price.

• Data contains numerical as well as categorical variable. You need to handle them accordingly.

MODEL DEVELOPMENT AND EVALUATION

Exploratory Data Analysis

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
train_df.shape, test_df.shape
```

```
((1168, 81), (292, 80))
```

```
train_df.sample(5)
```

ilope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	OverallQual	OverallCond	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exteri
Gtl	NridgHt	Norm	Norm	1Fam	1Story	7	5	2004	2004	Gable	CompShg	VinylSd	V
Gtl	CollgCr	Norm	Norm	1Fam	1Story	7	5	2007	2007	Gable	CompShg	VinylSd	V
Gtl	OldTown	Norm	Norm	1Fam	2Story	7	8	1915	2005	Gable	CompShg	Stucco	S
Gtl	NridgHt	Norm	Norm	1Fam	1Story	8	5	2006	2007	Gable	CompShg	VinylSd	V
Gtl	Somerst	Norm	Norm	1Fam	1Story	6	5	2009	2009	Gable	CompShg	VinylSd	V

- There are 1460 rows and 81 Columns in the entire dataset.

In the Data Analysis Phase we will Analyze the following in step by step:

1. Missing Values
2. All the Numerical variables
3. Distribution of the numerical variables
4. Categorical Variables
5. Cardinality of the categorical variables
6. Outliers
7. Relationship between Dependentn and Independent features

Missing values

```
features_with_na = [features for features in train_df.columns if train_df[features].isnull().sum()>1]
features_with_na
```

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

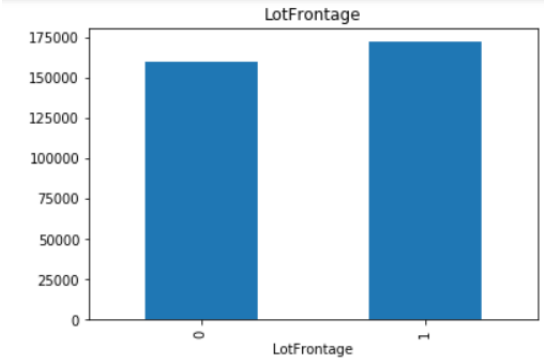
- Na values exist in the Dataset.

```
print("Features with missing values:")
for feature in features_with_na:
    print(feature, np.round(train_df[feature].isnull().mean(), 4))
```

```
Features with missing values:
LotFrontage 0.1832
Alley 0.9341
MasVnrType 0.006
MasVnrArea 0.006
BsmtQual 0.0257
BsmtCond 0.0257
BsmtExposure 0.0265
BsmtFinType1 0.0257
BsmtFinType2 0.0265
FireplaceQu 0.4717
GarageType 0.0548
GarageYrBlt 0.0548
GarageFinish 0.0548
GarageQual 0.0548
GarageCond 0.0548
PoolQC 0.994
Fence 0.7971
MiscFeature 0.9623
```

- Finding relationship between missing values and sales price

```
for x in features_with_na:
    df = train_df.copy()

    # converting missing values to 1 and others to 0
    df[x] = np.where(df[x].isnull(), 1, 0)

    df.groupby(x)['SalePrice'].median().plot.bar() #barplot because we have both numerical & categorical variable
    plt.title(x)
    plt.show()
```



Here, we can see that the missing values are impacting the SalePrice to a great extend.

1. In case of some feature, with high number of missing values, the median SalePrice is also high.
2. In other cases, with less number of missing values, less is the median SalePrice

Numerical Values

```
num_features = [feature for feature in train_df.columns if train_df[feature].dtypes != 'O']

print("No. of numerical features: ", len(num_features))
train_df[num_features].head()
```

```
No. of numerical features:  38
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmt
0	127	120	NaN	4928	6	5	1976	1976	0.0	120	0	958	10
1	889	20	95.0	15865	8	6	1970	1970	0.0	351	823	1043	22
2	793	60	92.0	9920	7	5	1996	1997	0.0	862	0	255	1
3	110	20	105.0	11751	6	6	1977	1977	480.0	705	0	1139	18
4	422	20	NaN	16635	6	7	1977	2000	126.0	1246	0	356	16

Temporal Values

```
train_df[year_features].head()
```

	YearBuilt	YearRemodAdd	GarageYrBlt	YrSold
0	1976	1976	1977.0	2007
1	1970	1970	1970.0	2007
2	1996	1997	1997.0	2007
3	1977	1977	1977.0	2010
4	1977	2000	1977.0	2009

```
for i in year_features:
    print('\n', i, train_df[i].unique())
```

```
YearBuilt [1976 1970 1996 1977 2006 1957 1965 1947 1937 2003 1960 1955 1923 1930
2007 2001 1972 1950 1961 1953 1918 2010 1922 1934 2005 1946 1941 1948
1975 1978 1956 2004 1982 2000 2002 1920 1992 1936 1967 1989 1929 1968
1959 1935 1966 1931 1916 1998 1962 1974 1926 1904 1995 1969 1985 1963
1958 1892 2008 1971 1980 1945 1986 1981 1949 1940 1954 1925 1915 1921
1924 1999 1951 1993 1964 1900 1919 1910 1938 1880 1988 1911 1990 1979
1927 1983 1994 2009 1928 1917 1898 1997 1984 1973 1952 1939 1987 1890
1942 1991 1932 1908 1914 1882 1905 1875 1906 1893 1912 1913]

YearRemodAdd [1976 1970 1997 1977 2000 2006 1996 1965 1950 2003 1960 1955 2007 2001
1961 1998 1953 2010 1995 2005 1992 1975 1978 1982 2002 1989 1967 1968
1959 2004 1966 2008 1987 1981 1969 1985 1963 1991 1993 1971 1990 1956
1986 1999 1954 1957 1994 1972 1958 1980 1979 1951 1983 2009 1962 1964
1952 1984 1988 1974 1973]

GarageYrBlt [1977. 1970. 1997. 2006. 1957. 1965. 1947. 1937. 2003. 1974. 1955. 1923.
2002. 2007. 1987. 2001. 1988. 1950. 1961. 1953. 2010. 1922. 1939. 2005.
1991. 1979. 1975. 1976. 1978. 1960. 1956. 2004. 1982. 2000. 1948.   nan
1964. 1920. 1930. 1968. 1946. 1992. 1936. 1967. 1989. 1959. 1966. 1916.
1941. 1998. 1962. 1926. 1925. 1983. 1999. 1969. 1985. 1993. 2008. 1971.
1980. 1945. 1995. 1981. 1994. 1949. 1996. 1921. 1963. 1938. 1958. 1935.
1940. 1990. 1910. 1954. 1927. 2009. 1986. 1929. 1984. 1973. 1924. 1942.
1900. 1931. 1951. 1934. 1972. 1932. 1928. 1918. 1908. 1933. 1906. 1914.
1952. 1915.]

YrSold [2007 2010 2009 2006 2008]
```

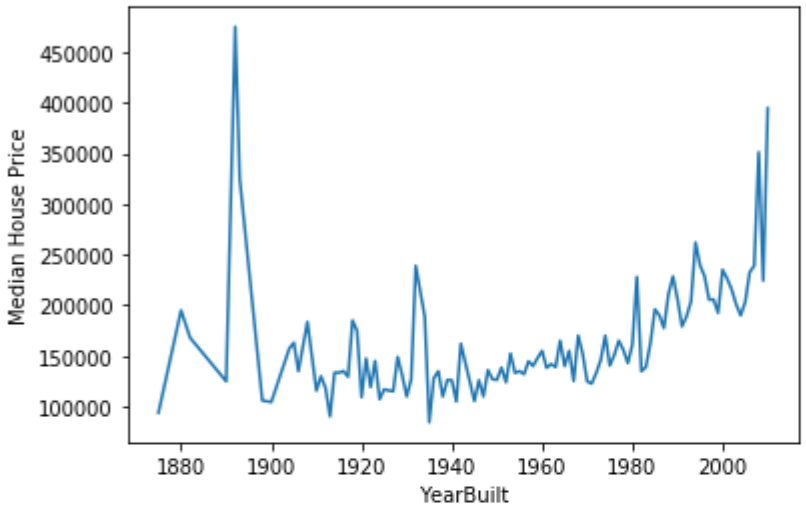
In the dataset, we have 4 Year variables. We need to extract information from these features such as No. of years or No. of days. One example in this specific scenario can be the difference in years between the house was build and the year it was sold. This will be considerd in the feature engineering part.

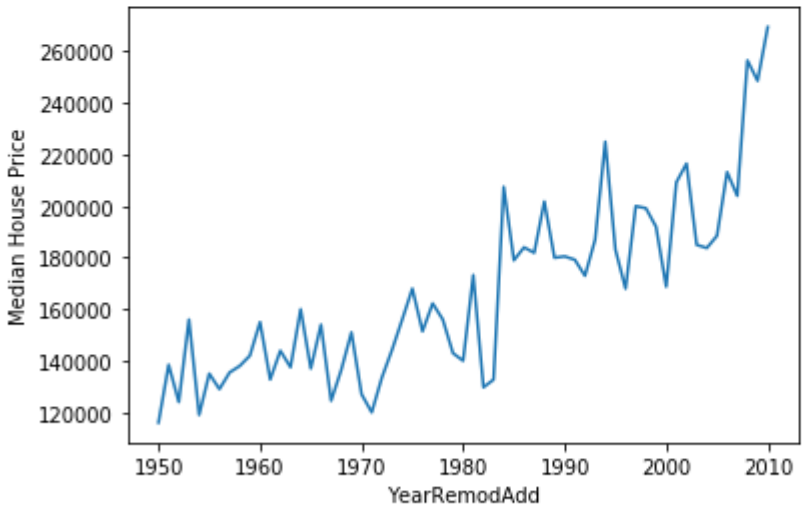
```
year_features = [col for col in num_features if 'Yr' in col or 'Year' in col]

print('Temporal Variables: ', year_features)
```

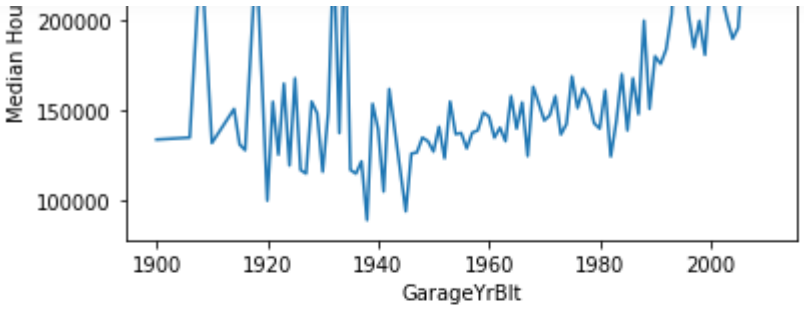
```
Temporal Variables:  ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
```

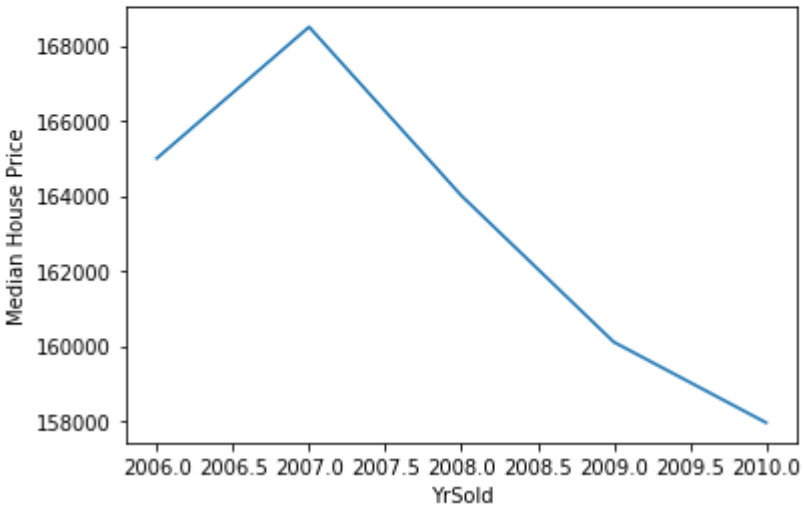
- Ploting Median House Price vs Year Built

```
for feature in year_features:
    train_df.groupby(feature)['SalePrice'].median().plot()
    plt.xlabel(feature)
    plt.ylabel('Median House Price')
    plt.show()
```







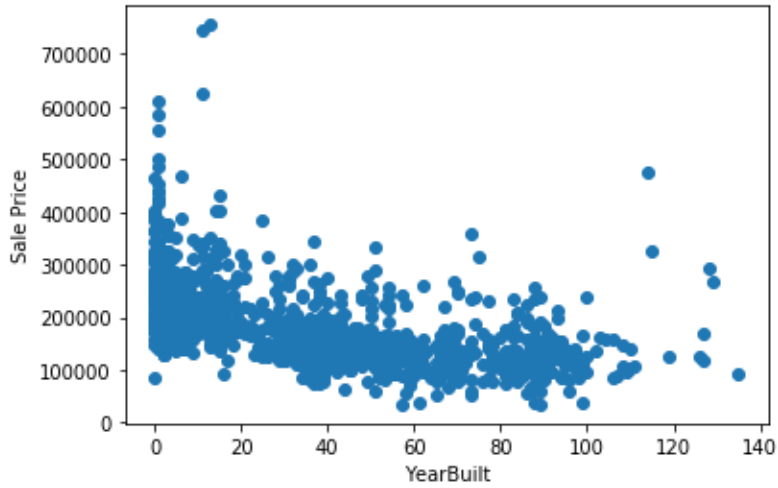


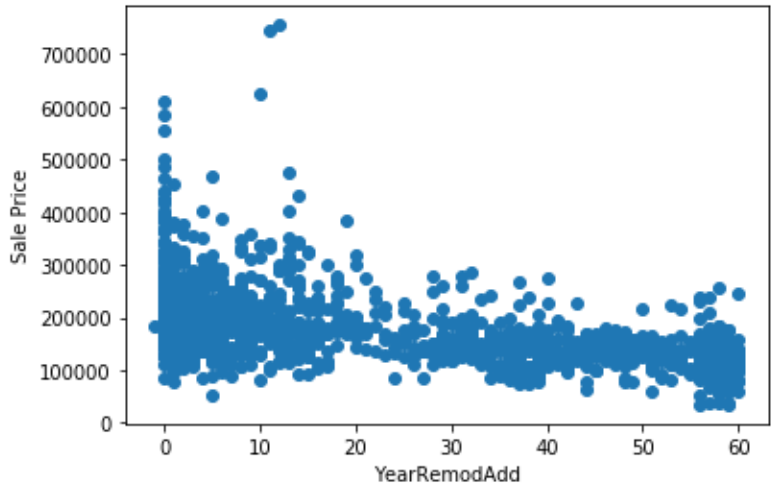
We can see that the Sale Price is decreasing as the advancement in time which is not quite acceptable as we know that in real life, the opposite only happens.

So, we will try to compare thr difference between all year feature with the SalePrice.

- **Scatterd Plot SalePrice vs Year Built**

```
for x in year_features:
    if x != 'YrSold':
        df = train_df.copy()
        df[x] = df['YrSold'] - df[x]
        plt.scatter(df[x], df['SalePrice'])
        plt.xlabel(x)
        plt.ylabel("Sale Price")
        plt.show()
```





Now it makes sense...¶

1. The SalePrice for recently built house is high while SalePrice for house build 140 years ago is low.
2. The SalePrice of house that are remodified recently is high and low for houses that were modified very long ago.
3. Simarly it is true for the feature GarageYrBlt

- **Discrete Variable**

```
# we are considering a feature to be dicrete if it has less than 25 unique values
# and are not part of Year Features and Id

discrete_cols = [col for col in num_features if len(train_df[col].unique())<25
                 and col not in year_features + ['Id']]
discrete_cols
```

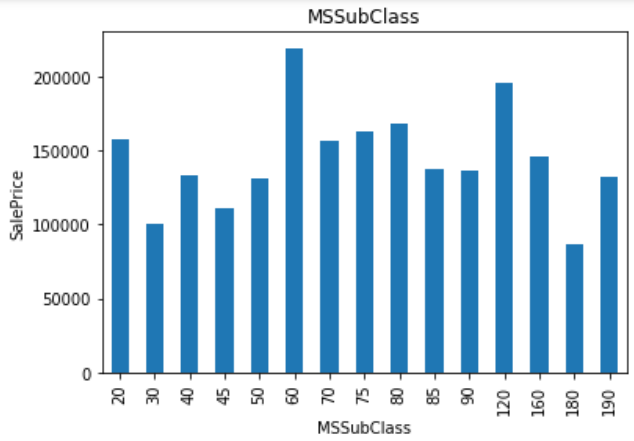
```
['MSSubClass',
 'OverallQual',
 'OverallCond',
 'LowQualFinSF',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
 'KitchenAbvGr',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageCars',
 '3SsnPorch',
 'PoolArea',
 'MiscVal',
 'MoSold']
```

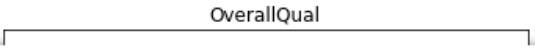
```
train_df[discrete_cols].head()
```

	MSSubClass	OverallQual	OverallCond	LowQualFinSF	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr	KitchenAbvGr	TotRmsAbvGrd	Firep
0	120	6	5	0	0	0	2	0	2	1	5	
1	20	8	6	0	1	0	2	0	4	1	8	
2	60	7	5	0	1	0	2	1	3	1	8	
3	20	6	6	0	0	0	2	0	3	1	7	
4	20	6	7	0	0	1	2	0	3	1	8	

- **Finding relationship between discrete and dependent variable**
- **Finding relationship between discrete and dependent variable**
- **Finding relationship between discrete and dependent variable**

```
for feature in discrete_cols:
    data = train_df.copy()
    data.groupby(feature)['SalePrice'].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel("SalePrice")
    plt.title(feature)
    plt.show()
```





Observations:

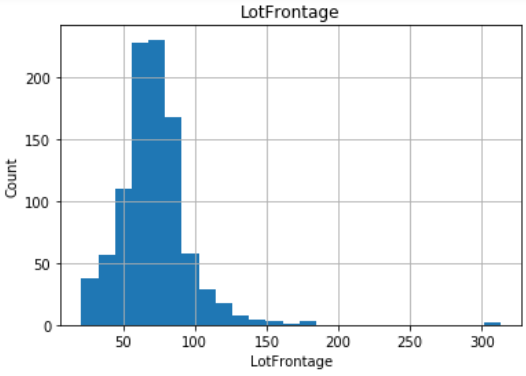
- With the increase in overall quality, the SalePrice is exponentially increasing (a monotonic relationship).
- There are some relationship between discrete and dependent variable

- **Continuous variable**

```
contin_feature=[feature for feature in num_features if feature not in discrete_cols+ year_features+['Id']]
print("No. of Continuous Features: {}".format(len(contin_feature)))
```

```
No. of Continuous Features: 16
```

```
# Analyzing the Continuous variables

# Creating histograms because we are trying to find out the distribution of the continuous variables
for feature in contin_feature:
    data = train_df.copy()
    data[feature].hist(bins=25)
    plt.xlabel(feature)
    plt.ylabel("Count")
    plt.title(feature)
    plt.show()
```

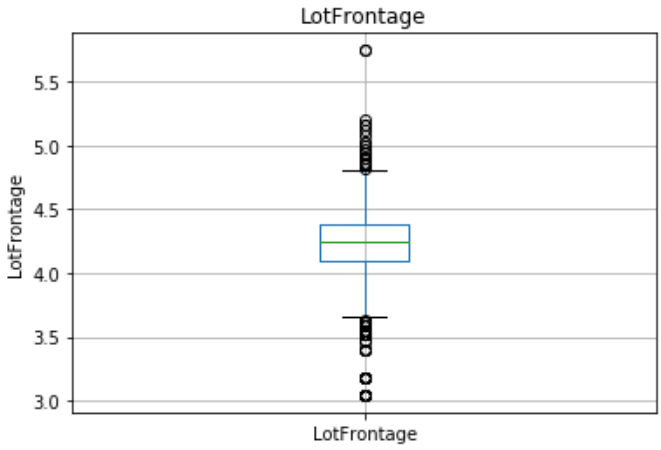


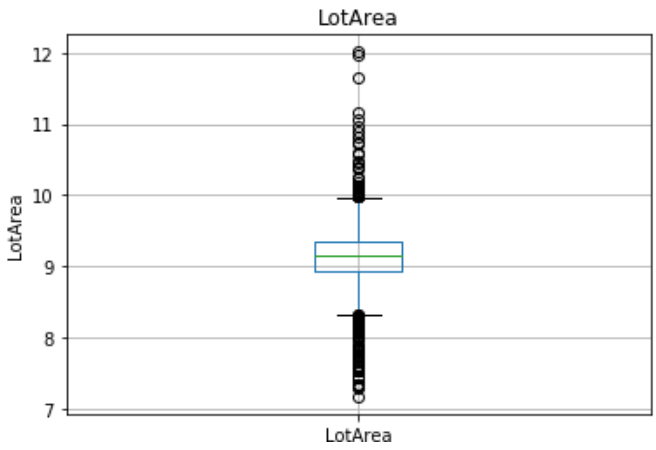
Most of the continuous distribution are not in normal (Gaussian) distribution form but skewed.

We need to keep in mind that, while dealing a regression problem with non-normal data, we need to convert them into normal (by log normal dist) for better working of the model
After applying the log normal transformation and plotting the distribution we can see that there's a **Monotonic Relation** between the continuous features and the dependent variables

- **Outliers**

```
# Finding the outliers

for feature in contin_feature:
    data = train_df.copy()
    if 0 in data[feature].unique():
        pass
    else:
        data[feature] = np.log(data[feature])
        data.boxplot(column=feature)
        #plt.xlabel(feature)
        plt.ylabel(feature)
        plt.title(feature)
        plt.show()
```





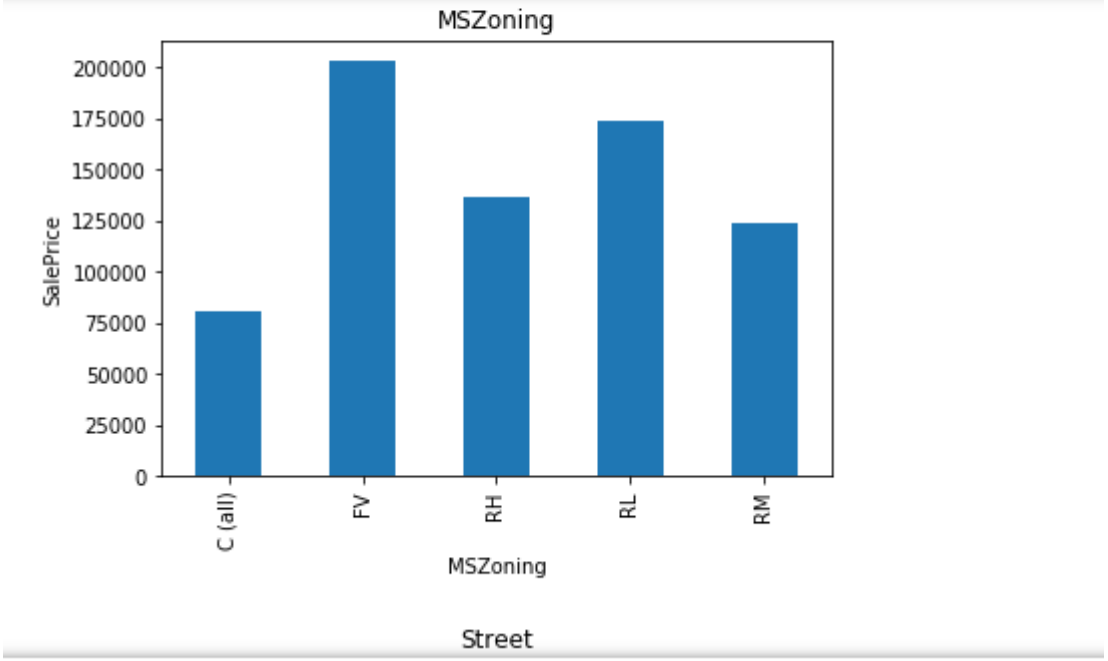
- **Categorical Features**

```
categorical_feature=[feature for feature in train_df.columns if train_df[feature].dtype=='O']

#len(categorical_feature)
categorical_feature
```

```
['MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
```

- **Relationship**

```
for feature in categorical_feature:
    data = train_df.copy()
    data.groupby(feature)['SalePrice'].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.title(feature)
    plt.show()
```



- **FEATURE ENGINEERING**

1. The main aim of this phase is to make our data suitable and add more importance to it for the modelling

2. We have to do the exact Feature Engineering for both the Train and the Test data, so we'll be combining the data and then will perform the feature engineering.

missing values

```
# Let us capture all the NAN values
# First Let's handle the categorical features which are missing
feature_nan = [feature for feature in combined.columns if combined[feature].isnull().sum()>1 and combined[feature].dtypes=='O']

for feature in feature_nan:
    print("{}: {}% missing values".format(feature, np.round(combined[feature].isnull().mean(), 4)))
```

```
Alley: 0.9377% missing values
MasVnrType: 0.0055% missing values
BsmtQual: 0.0253% missing values
BsmtCond: 0.0253% missing values
BsmtExposure: 0.026% missing values
BsmtFinType1: 0.0253% missing values
BsmtFinType2: 0.026% missing values
FireplaceQu: 0.4726% missing values
GarageType: 0.0555% missing values
GarageFinish: 0.0555% missing values
GarageQual: 0.0555% missing values
GarageCond: 0.0555% missing values
PoolQC: 0.9952% missing values
Fence: 0.8075% missing values
MiscFeature: 0.963% missing values
```

```
# replacing the missing values of the above categorical feature with a new feature
# replacing NAN values by a label 'Missing'
def replace_cat_feature(dataset, feature_nan):
    data = combined.copy()
    data[feature_nan] = data[feature_nan].fillna('Missing')
    return data

combined = replace_cat_feature(combined, feature_nan)
combined[feature_nan].isnull().sum()
```

```
Alley          0
MasVnrType     0
BsmtQual       0
BsmtCond       0
BsmtExposure   0
```

- **Handling the numerical features which are missing**

```
# Let's handle the Numerical features which are missing

numerical_with_nan = [feature for feature in combined.columns if combined[feature].isnull().sum()>1 and combined[feature].dtypes

for feature in numerical_with_nan:
    print("{}: {}% missing value".format(feature, np.round(combined[feature].isnull().mean(),4)))
```

```
LotFrontage: 0.1774% missing value
MasVnrArea: 0.0055% missing value
GarageYrBlt: 0.0555% missing value
```

```
# Replacing the numerical missing values

for feature in numerical_with_nan:
    #we'll replace missng values by median(because we have some outliers)
    median_value = combined[feature].median()

    # create a new feature to capture the NAN values
    # we are replacing the nan values by 1 and others by 0 in the newly created feature
    combined[feature + 'nan'] = np.where(combined[feature].isnull(), 1, 0)
    # we are doing the above line only to get some more information about the feature
    combined[feature].fillna(median_value, inplace=True)

combined[numerical_with_nan].isnull().sum()
```

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

```
#### Temporal Variables (Date Time Variables)
# we are replacing the years with the difference in years from the Year of SOld

for feature in ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt']:
    combined[feature] = combined['YrSold'] - combined[feature]

combined.head()
```

- **Numerical Variables**

Since the numerical variables are skewed, we'll perform Log Normal Transformation

```
combined.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Cor
0	127	120	RL	69.0	4928	Pave	Missing	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	
1	889	20	RL	95.0	15865	Pave	Missing	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	
2	793	60	RL	92.0	9920	Pave	Missing	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	
3	110	20	RL	105.0	11751	Pave	Missing	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	
4	422	20	RL	69.0	16635	Pave	Missing	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	

```
# from EDA we got the following Skewed features
numerical_features = ['LotFrontage','LotArea', '1stFlrSF', 'GrLivArea']

for feature in numerical_features:
    combined[feature] = np.log(combined[feature])

combined.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Co
0	127	120	RL	4.234107	8.502689	Pave	Missing	IR1	Lvl	AllPub	Inside	Gtl	NPkVill	Norm	
1	889	20	RL	4.553877	9.671871	Pave	Missing	IR1	Lvl	AllPub	Inside	Mod	NAmes	Norm	
2	793	60	RL	4.521789	9.202308	Pave	Missing	IR1	Lvl	AllPub	CulDSac	Gtl	NoRidge	Norm	
3	110	20	RL	4.653960	9.371694	Pave	Missing	IR1	Lvl	AllPub	Inside	Gtl	NWAmes	Norm	
4	422	20	RL	4.234107	9.719264	Pave	Missing	IR1	Lvl	AllPub	FR2	Gtl	NWAmes	Norm	

Now, we can see that our numerical_features columns got their Natural logarithmic values, which reduced their skewness.

KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION

- When it comes to the evaluation of a data science model's performance, sometimes accuracy may not be the best indicator.
- So, we have used f1 score as well as recall, precision to check the performance of the models.

MODEL TRAINING

- Modelling

```
X_train.shape, X_test.shape, Y_train.shape
```

```
((1168, 263), (292, 263), (1168,))
```

```
X_test_id = test_df['Id']
X_test_id.tail()
```

```
287      83
288    1048
289      17
290     523
291    1379
Name: Id, dtype: int64
```

```
from sklearn.linear_model import Lasso, Ridge, ElasticNet
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import mean_squared_error
```

Linear Models

```
X_test.tail()
```

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtS
1455	0.000000	0.485693	0.403297	0.777778	0.50	0.007353	0.032787	0.2925	0.005847	0.0	0.654966	0.25581
1456	0.000000	0.369595	0.383942	0.444444	0.50	0.102941	0.229508	0.0000	0.121545	0.0	0.130137	0.16202
1457	0.000000	0.440313	0.422202	0.555556	0.75	0.294118	0.672131	0.1125	0.102410	0.0	0.182363	0.16432
1458	0.176471	0.321097	0.263645	0.555556	0.75	0.433824	0.934426	0.0000	0.070695	0.0	0.258990	0.16432
1459	0.823529	0.000000	0.079657	0.555556	0.50	0.242647	0.557377	0.2550	0.054748	0.0	0.074486	0.07905

```
# Training the models

lasso_model = Lasso(alpha=0.0009, max_iter=50000).fit(X_train, Y_train)

#elastic_net_model = ElasticNet(alpha=0.0019, max_iter=50000).fit(X_train, Y_train)

#ridge_model = Ridge(alpha=41, max_iter=50000).fit(X_train, Y_train)
```

```
lasso_predictions = lasso_model.predict(X_test)

ridge_predictions = ridge_model.predict(X_test)

elastic_net_predictions = elastic_net_model.predict(X_test)
```

```
lasso_predictions
```

```
...
```

```
lasso_predictions= pd.DataFrame(lasso_predictions, index=X_test_id)
lasso_predictions.columns = ['Lasso_price']

ridge_predictions= pd.DataFrame(ridge_predictions, index=X_test_id)
ridge_predictions.columns = ['Ridge_price']

elastic_net_predictions= pd.DataFrame(elastic_net_predictions, index=X_test_id)
elastic_net_predictions.columns = ['Elastic_price']
```

```
pred = lasso_predictions.merge(ridge_predictions, left_index=True, right_index=True, how='inner')
pred = pred.merge(elastic_net_predictions, left_index=True, right_index=True, how='inner')
pred/head()
```

Id	Lasso_price	Ridge_price	Elastic_price
337	372362.967005	334788.381659	369041.782916
1018	225528.910188	206758.392528	235633.476244
929	240861.112977	236552.426350	247428.133026
1148	178222.069019	175742.842385	171984.630597
1227	229062.225221	251895.792255	233304.625013
650	72459.951106	66763.543813	72152.234305
1453	119088.097016	134613.401364	118597.851984
152	350197.532961	334540.065630	342602.500124
427	244970.262116	231802.156915	232188.944688
776	161095.846180	173903.950296	161204.632317
30	62047.241894	70121.099679	52383.770595

```
pred['SalePrice'] = (pred.Lasso_price + pred.Ridge_price + pred.Elastic_price)/3
pred.head()
```

Id	Lasso_price	Ridge_price	Elastic_price	SalePrice
337	372362.967005	334788.381659	369041.782916	358731.043860
1018	225528.910188	206758.392528	235633.476244	222640.259653
929	240861.112977	236552.426350	247428.133026	241613.890784
1148	178222.069019	175742.842385	171984.630597	175316.514000
1227	229062.225221	251895.792255	233304.625013	238087.547496

```
pred1 = pred.drop(['Lasso_price','Ridge_price', 'Elastic_price'], axis=1)
pred1.to_csv('Predicted_salePrice_based_on_linear_models.csv')
```

ENSEMBLE Method

```
gradient_boost = GradientBoostingRegressor(learning_rate=0.1, n_estimators=50).fit(X_train, Y_train)
```

```
grad_boost_predictions = gradient_boost.predict(X_test)
```

```
grad_boost_predictions = pd.DataFrame(grad_boost_predictions,index=X_test_id)
grad_boost_predictions.columns = ['SalePrice']
```

```
grad_boost_predictions.head()
```

Id	SalePrice
337	368211.918317
1018	223875.262711
929	245582.040891
1148	178198.523383
1227	204319.490807
650	84242.935014
1453	125785.781301
152	324066.685678
427	236699.411117
776	158640.571669
30	73821.730247