FLIP ROBO

House Price Prediction

Submitted By

K. Naveen Varma

HOUSE PRICE PREDICTION

Problem Statement:

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. 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?

Business Goal:

You 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:

Linear Regression: Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

Naming the Variables. There are many names for a regression's dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressand. The independent variables can be called exogenous variables, predictor variables, or regressors.

Three major uses for regression analysis are (1) determining the strength of predictors, (2) forecasting an effect, and (3) trend forecasting.

First, the regression might be used to identify the strength of the effect that the independent variable(s) have on a dependent variable. Typical questions are what is the strength of relationship between dose and effect, sales and marketing spending, or age and income.

Second, it can be used to forecast effects or impact of changes. That is, the regression analysis helps us to understand how much the dependent variable changes with a change in one or more independent variables. A typical question is, "how much additional sales income do I get for each additional $1000 spent on marketing?"

Third, regression analysis predicts trends and future values. The regression analysis can be used to get point estimates. A typical question is, "what will the price of gold be in 6 months?"

Types of Linear Regression:

Simple Linear Regression: 1 dependent variable (interval or ratio) , 2+ independent variables (interval or ratio or dichotomous).

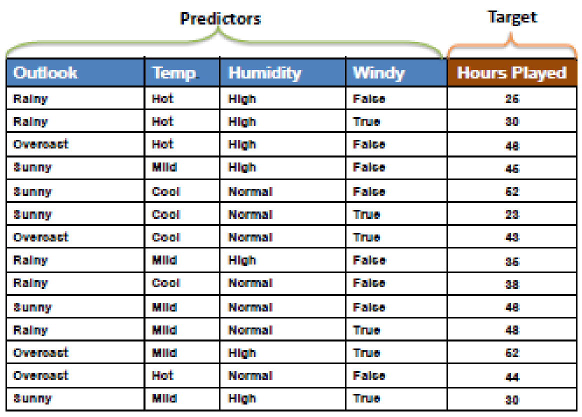
Logistic Regression: 1 dependent variable (dichotomous), 2+ independent variable(s) (interval or ratio or dichotomous)

Ordinal Regression:1 dependent variable (ordinal), 1+ independent variable(s) (nominal or dichotomous)

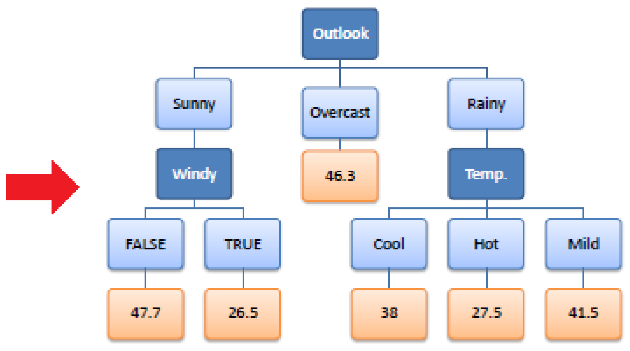
Multinomial Regression: 1 dependent variable (nominal), 1+ independent variable(s) (interval or ratio or dichotomous)

Discriminant Analysis: 1 dependent variable (nominal), 1+ independent variable(s) (interval or ratio)

Decision Tree Regression: 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.



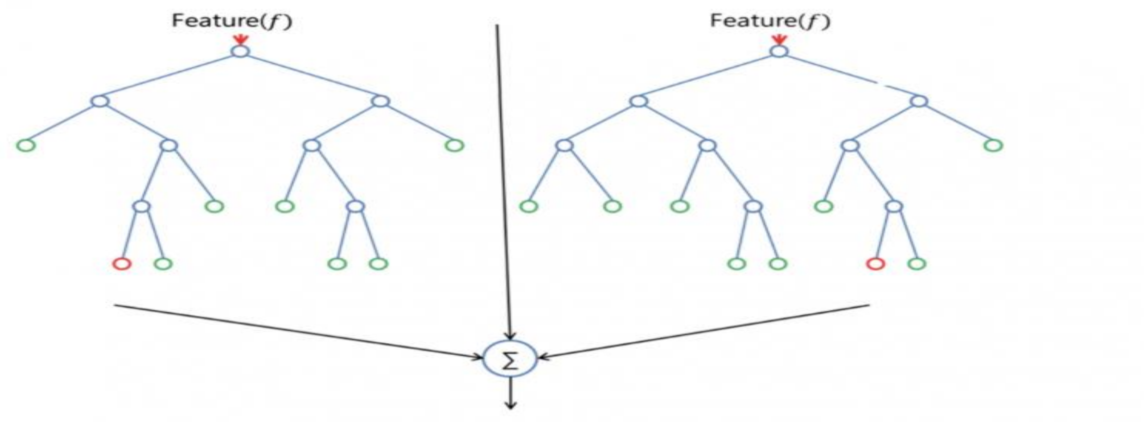
Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Decision Tree Algorithm: The core algorithm for building decision trees called **ID3** by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with *Standard Deviation Reduction*.

Standard Deviation: A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). We use standard deviation to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous its standard deviation is zero.

Random Forest Regression: Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.One big advantage of random forest is that it can be used for both classification and regression problems, which form the majority of current machine learning systems. Let's look at random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.
Random forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.
Therefore, in random forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Lasso And Ridge Regression: Ridge and Lasso might appear to work towards a common goal, the inherent properties and practical use cases differ substantially. If you've heard of them before, you must know that they work by penalizing the magnitude of coefficients of features along with minimizing the error between predicted and actual observations. These are called 'regularization' techniques. The key difference is in how they assign penalty to the coefficients:

1. **Ridge Regression:**
   - Performs L2 regularization, i.e. adds penalty equivalent to **square of the magnitude** of coefficients
   - Minimization objective = LS Obj + α * (sum of square of coefficients)
2. **Lasso Regression:**
   - Performs L1 regularization, i.e. adds penalty equivalent to **absolute value of the magnitude** of coefficients
   - Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

Gradient Boosting Regressor:
Gradient boosting is a technique attracting attention for its prediction speed and accuracy, especially with large and complex data. Don't just take my word for it, the chart below shows the rapid growth of Google searches for xgboost (the most popular gradient boosting R package). From data science competitions to machine learning solutions for business, gradient boosting has produced best-in-class results. In this blog post I describe what is gradient boosting and how to use gradient boosting.

Ensembles and boosting
Machine learning models can be fitted to data individually, or combined in an *ensemble*. An ensemble is a combination of simple individual models that together create a more powerful new model.
Machine learning boosting is a method for creating an ensemble. It starts by fitting an initial model (e.g. a tree or linear regression) to the data. Then a second model is built that focuses on accurately predicting the cases where the first model performs poorly. The combination of these two models is expected to be better than either model alone. Then you repeat this process of boosting many times. Each successive model attempts to correct for the shortcomings of the combined boosted ensemble of all previous models.

Gradient boosting explained
Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error. How are the targets calculated? The target outcome for each case in the data depends on how much changing that case's prediction impacts the overall prediction error:

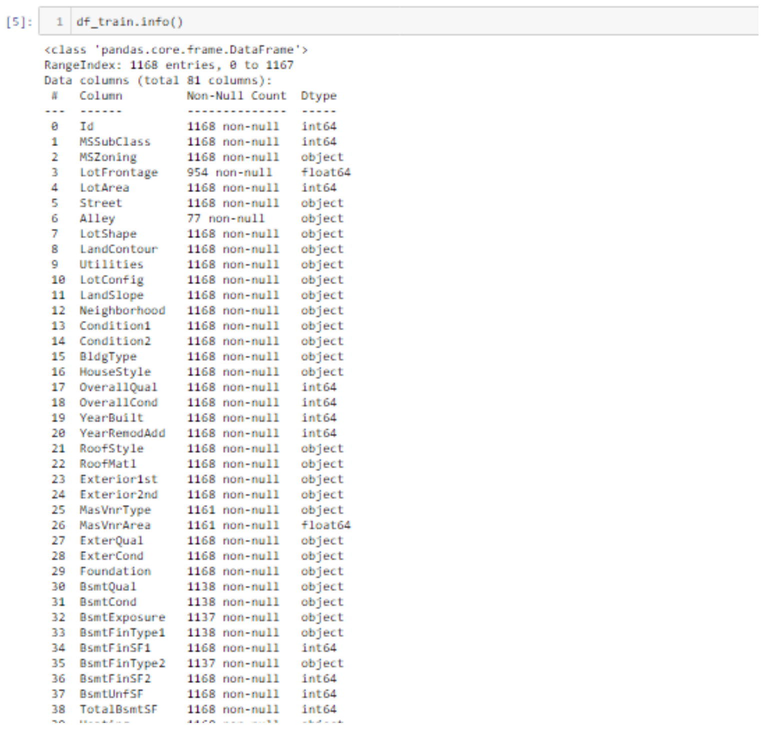
- If a small change in the prediction for a case causes a large drop in error, then next target outcome of the case is a high value. Predictions from the new model that are close to its targets will reduce the error.

- If a small change in the prediction for a case causes no change in error, then next target outcome of the case is zero. Changing this prediction does not decrease the error.

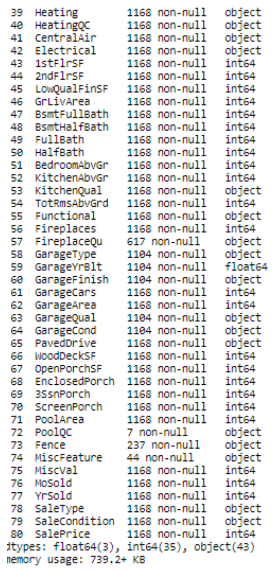
The name *gradient boosting* arises because target outcomes for each case are set based on the gradient of the error with respect to the prediction. Each new model takes a step in the direction that minimizes prediction error, in the space of possible predictions for each training case.

Data Analysis: The Dataset Contains a Data of 1168 entries each having 81 variables, in which some are numerical Data and some are Categorical Data

As the Data having two datasets 1. Train Data, 2. Test Data



```
[5]:  df_train.info()

<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
 31  BsmtCond       1138 non-null   object
 32  BsmtExposure   1137 non-null   object
 33  BsmtFinType1   1138 non-null   object
 34  BsmtFinSF1     1168 non-null   int64
 35  BsmtFinType2   1137 non-null   object
 36  BsmtFinSF2     1168 non-null   int64
 37  BsmtUnfSF      1168 non-null   int64
 38  TotalBsmtSF    1168 non-null   int64
```



```
 39  Heating        1168 non-null   object
 40  HeatingQC      1168 non-null   object
 41  CentralAir     1168 non-null   object
 42  Electrical     1168 non-null   object
 43  1stFlrSF       1168 non-null   int64
 44  2ndFlrSF       1168 non-null   int64
 45  LowQualFinSF   1168 non-null   int64
 46  GrLivArea      1168 non-null   int64
 47  BsmtFullBath   1168 non-null   int64
 48  BsmtHalfBath   1168 non-null   int64
 49  FullBath       1168 non-null   int64
 50  HalfBath       1168 non-null   int64
 51  BedroomAbvGr   1168 non-null   int64
 52  KitchenAbvGr   1168 non-null   int64
 53  KitchenQual    1168 non-null   object
 54  TotRmsAbvGrd   1168 non-null   int64
 55  Functional     1168 non-null   object
 56  Fireplaces     1168 non-null   int64
 57  FireplaceQu    617 non-null    object
 58  GarageType     1104 non-null   object
 59  GarageYrBlt    1104 non-null   float64
 60  GarageFinish   1104 non-null   object
 61  GarageCars     1168 non-null   int64
 62  GarageArea     1168 non-null   int64
 63  GarageQual     1104 non-null   object
 64  GarageCond     1104 non-null   object
 65  PavedDrive     1168 non-null   object
 66  WoodDeckSF     1168 non-null   int64
 67  OpenPorchSF    1168 non-null   int64
 68  EnclosedPorch  1168 non-null   int64
 69  3SsnPorch      1168 non-null   int64
 70  ScreenPorch    1168 non-null   int64
 71  PoolArea       1168 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          237 non-null    object
 74  MiscFeature    44 non-null     object
 75  MiscVal        1168 non-null   int64
 76  MoSold         1168 non-null   int64
 77  YrSold         1168 non-null   int64
 78  SaleType       1168 non-null   object
 79  SaleCondition  1168 non-null   object
 80  SalePrice      1168 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 739.2+ KB
```

As the Above Data-set we are having float -3 int-35 and object-43 so total of 81 variables.

Exploratory Data Analysis:

In EDA we need to Pre-process the Data and Visualization:

Steps include in Pre-Processing Data are

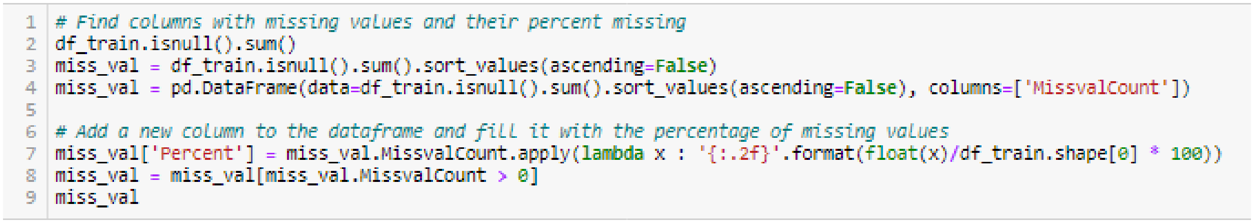
1)**Data Cleaning**: - Removing Outliers, Skewness and imputing Missing Values.

2)**Data Transformation**: - like Normalization by applying normalization we can improve the accuracy and efficiency of the models. And also reduce the errors.

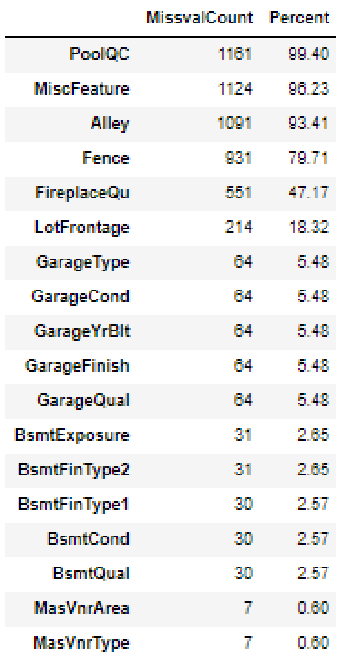
3)**Data Reduction**: By Reducing the no of features by Feature Selection Process, PCA And VIF

1.Data Cleaning: As a Part of EDA we need to do Data cleaning so firstly we need to check any null values in our data, From the below image shows we don't have any null values, so no need to impute any data

First of All Need to check the data is having any missing values



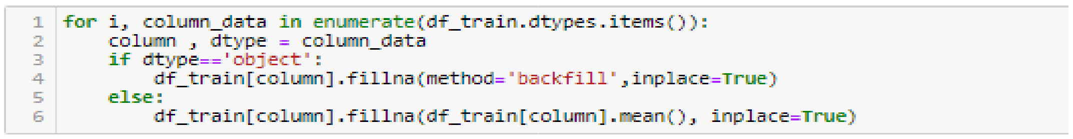
```python
# Find columns with missing values and their percent missing
df_train.isnull().sum()
miss_val = df_train.isnull().sum().sort_values(ascending=False)
miss_val = pd.DataFrame(data=df_train.isnull().sum().sort_values(ascending=False), columns=['MissvalCount'])

# Add a new column to the dataframe and fill it with the percentage of missing values
miss_val['Percent'] = miss_val.MissvalCount.apply(lambda x : '{:.2f}'.format(float(x)/df_train.shape[0] * 100))
miss_val = miss_val[miss_val.MissvalCount > 0]
miss_val
```



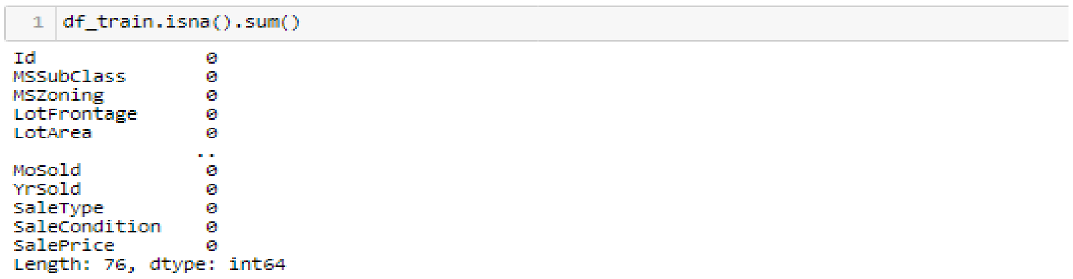
	MissvalCount	Percent
PoolQC	1161	99.40
MiscFeature	1124	96.23
Alley	1091	93.41
Fence	931	79.71
FireplaceQu	551	47.17
LotFrontage	214	18.32
GarageType	64	5.48
GarageCond	64	5.48
GarageYrBlt	64	5.48
GarageFinish	64	5.48
GarageQual	64	5.48
BsmtExposure	31	2.65
BsmtFinType2	31	2.65
BsmtFinType1	30	2.57
BsmtCond	30	2.57
BsmtQual	30	2.57
MasVnrArea	7	0.60
MasVnrType	7	0.60

The Dataset Having the null values and I am removing the columns which is having the percentage more than 45%.

By comparing both train and test data column PoolQC, MiscFeature, Alley, Fence, FireplaceQu having more than 45% data is missing.so removing from data set and remaining all very small percent so fill the null values



```python
for i, column_data in enumerate(df_train.dtypes.items()):
    column , dtype = column_data
    if dtype=='object':
        df_train[column].fillna(method='backfill',inplace=True)
    else:
        df_train[column].fillna(df_train[column].mean(), inplace=True)
```

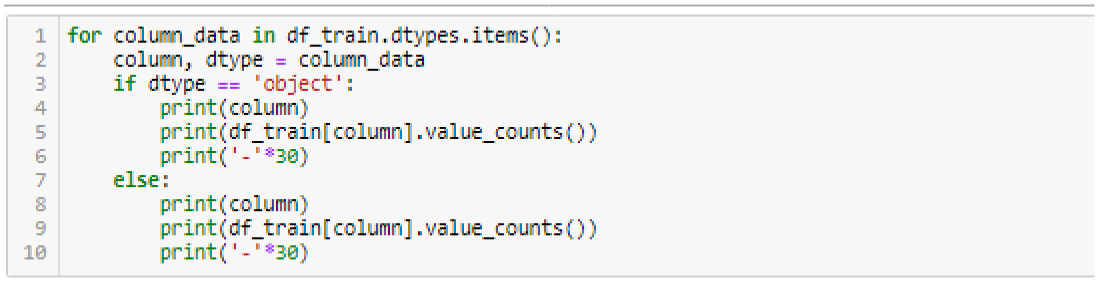


```python
df_train.isna().sum()
```

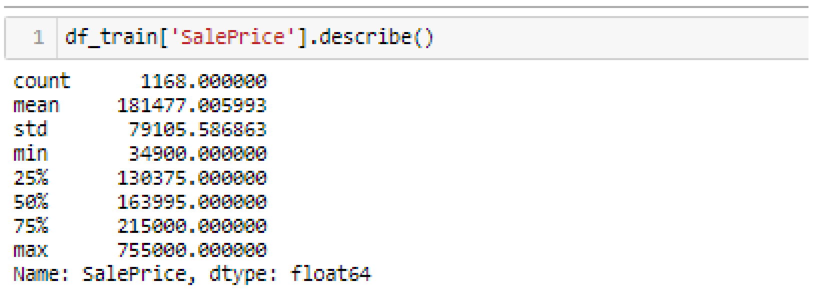
```
Id               0
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
                ..
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
Length: 76, dtype: int64
```

For the remaining missing values , we need to use the impute method

For need to know how many unique values are there with the count by using this formula



```python
for column_data in df_train.dtypes.items():
    column, dtype = column_data
    if dtype == 'object':
        print(column)
        print(df_train[column].value_counts())
        print('-'*30)
    else:
        print(column)
        print(df_train[column].value_counts())
        print('-'*30)
```



```python
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
```

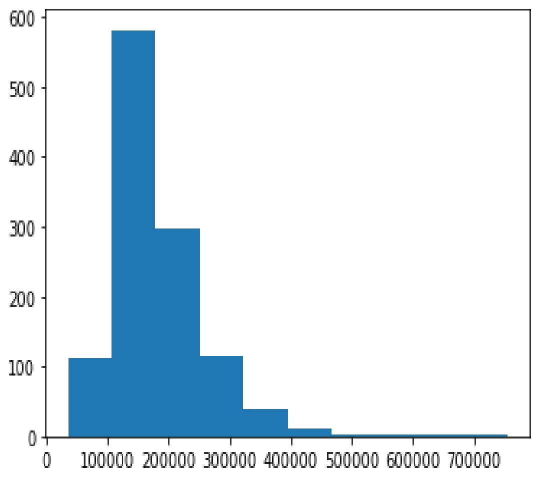
As it shows the 75% and Max having so much difference so our column is rightly skewed...so do skewing for plotting with variables

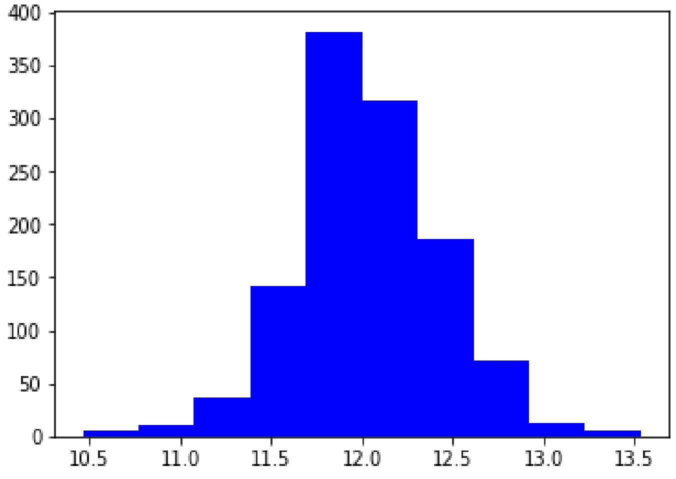
```python

# Determining the Skewness of data
print ("Skew is:", df_train.SalePrice.skew())

plt.hist(df_train.SalePrice)
plt.show()

# After log transformation of the data it looks much more center aligned
df_train['Skewed_SP'] = np.log(df_train['SalePrice']+1)
print ("Skew is:", df_train['Skewed_SP'].skew())
plt.hist(df_train['Skewed_SP'], color='blue')
plt.show()
```

Skew for the variable is 1.98

Before Skew

After Skew

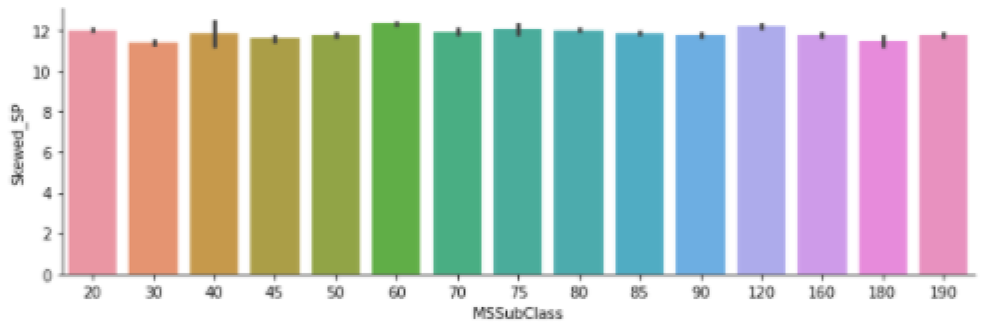


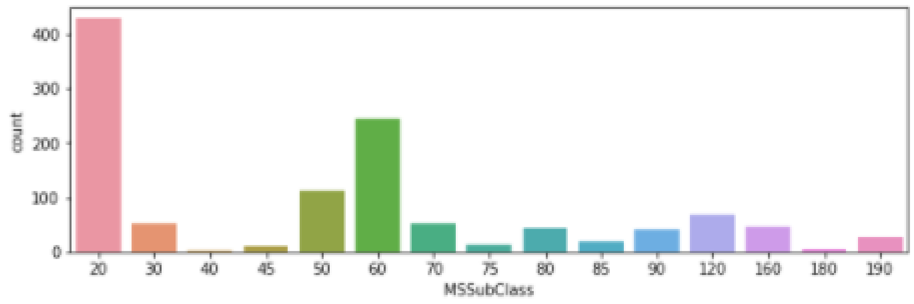


VISUALIZATION:

```python
sns.factorplot('MSSubClass', 'Skewed_SP', data=df_train,kind='bar',size=3,aspect=3)
fig, (axis1) = plt.subplots(1,1,figsize=(10,3))
sns.countplot('MSSubClass', data=df_train)
df_train['MSSubClass'].value_counts()
```

```
20     428
60     244
50     113
120     69
70      53
30      52
160     47
80      43
90      41
190     26
85      19
75      14
45      10
180      6
40       3
Name: MSSubClass, dtype: int64
```

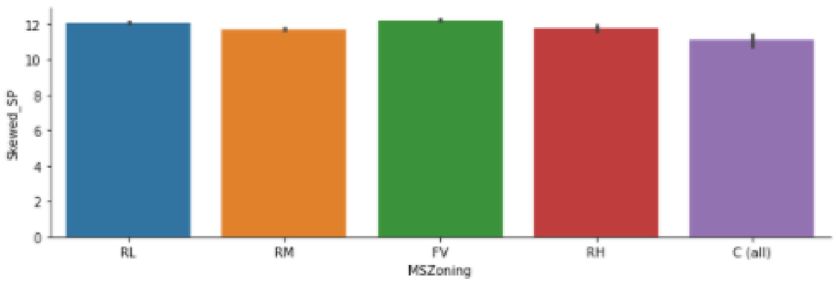


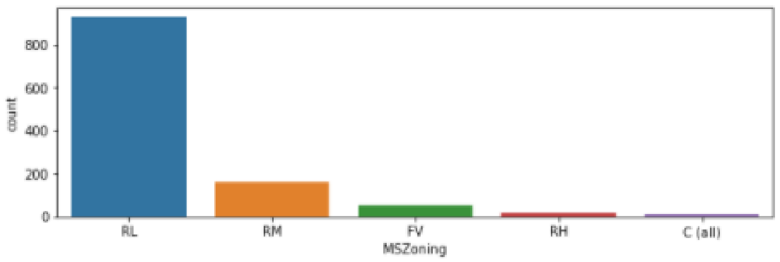


MSSubClass = 60 has highest SalePrice while the sales of houses with MSSubClass = 20 is the highest

```
sns.factorplot('MSZoning', 'Skewed_SP', data=df_train,kind='bar',size=3,aspect=3)
fig, (axis1) = plt.subplots(1,1,figsize=(10,3))
sns.countplot(x='MSZoning', data=df_train, ax=axis1)
df_train['MSZoning'].value_counts()
```

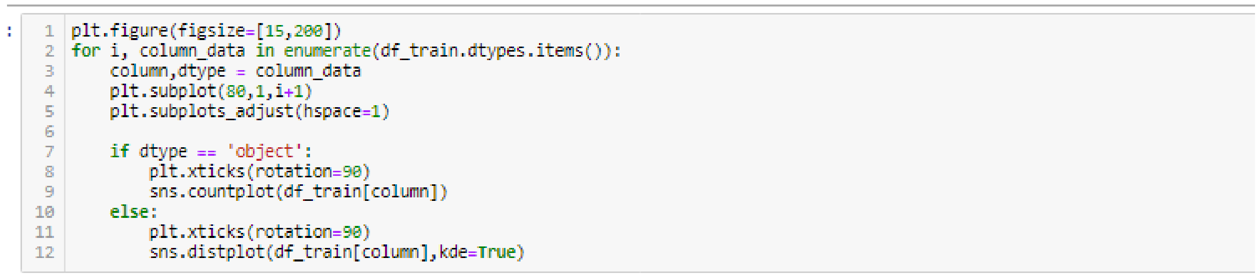
```
RL         928
RM         163
FV          52
RH          16
C (all)      9
Name: MSZoning, dtype: int64
```



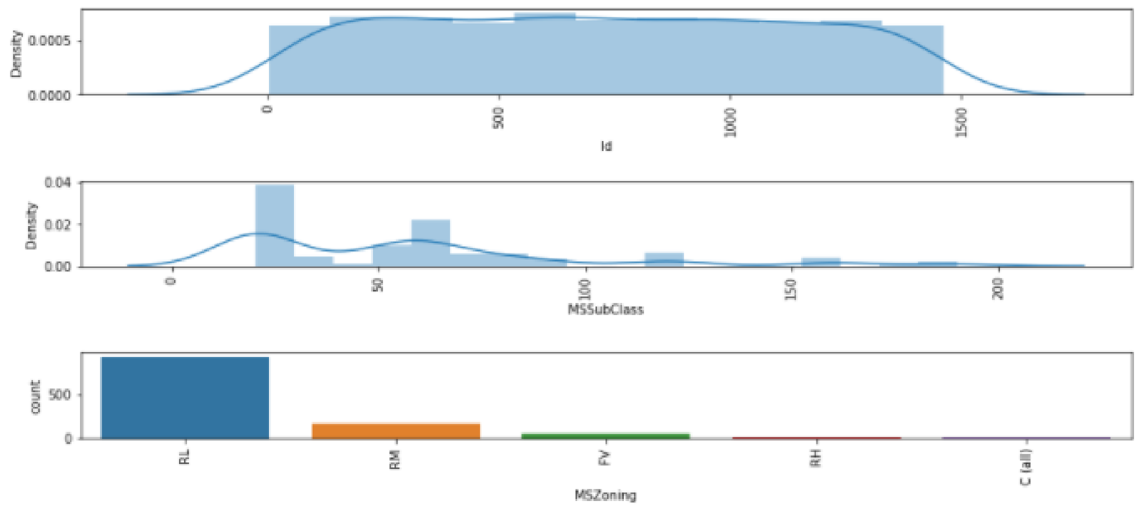


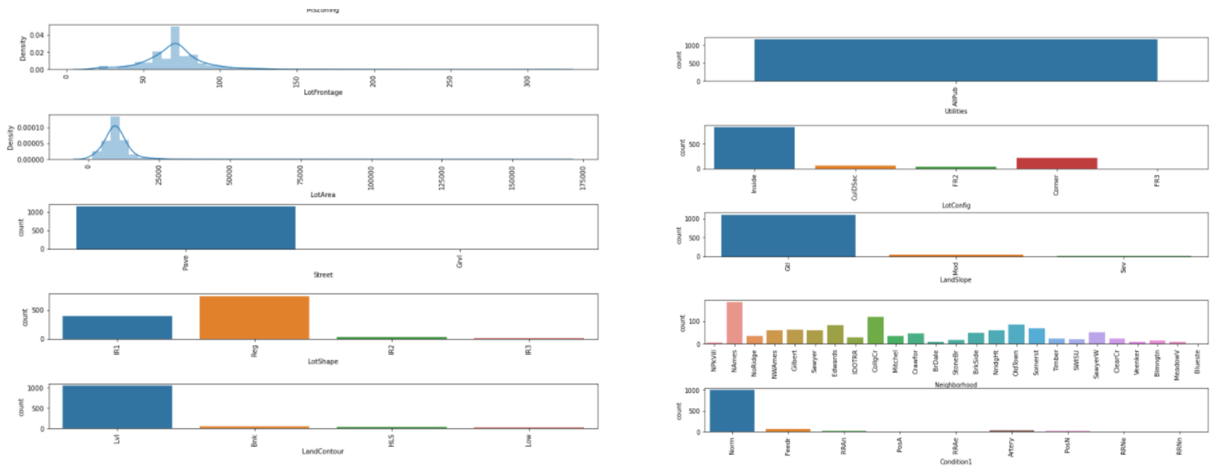
For MsZoning RL is the RL is the Highest and with comparision of Skewed Sales Price FV is Highest

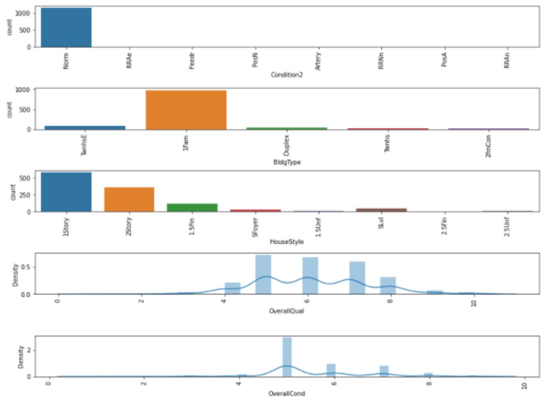
Code for Univariate Variable:

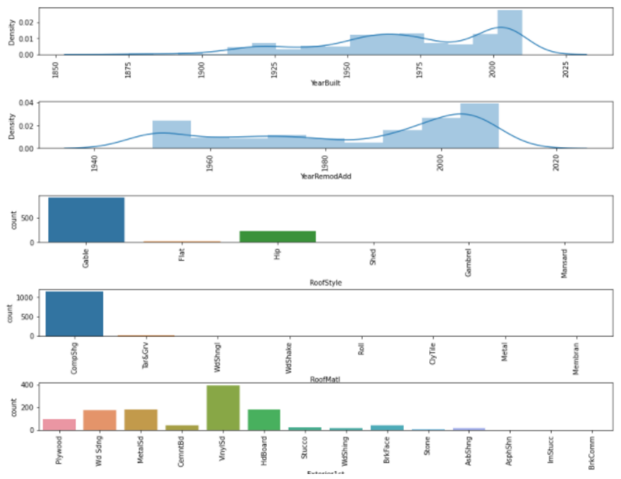


```
plt.figure(figsize=[15,200])
for i, column_data in enumerate(df_train.dtypes.items()):
    column,dtype = column_data
    plt.subplot(80,1,i+1)
    plt.subplots_adjust(hspace=1)

    if dtype == 'object':
        plt.xticks(rotation=90)
        sns.countplot(df_train[column])
    else:
        plt.xticks(rotation=90)
        sns.distplot(df_train[column],kde=True)
```



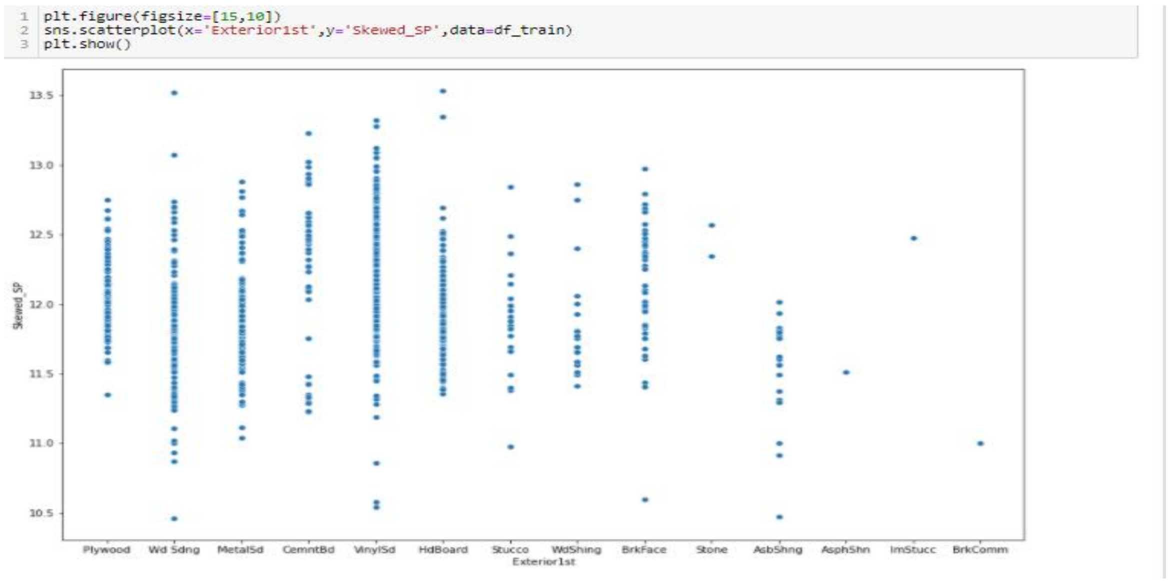




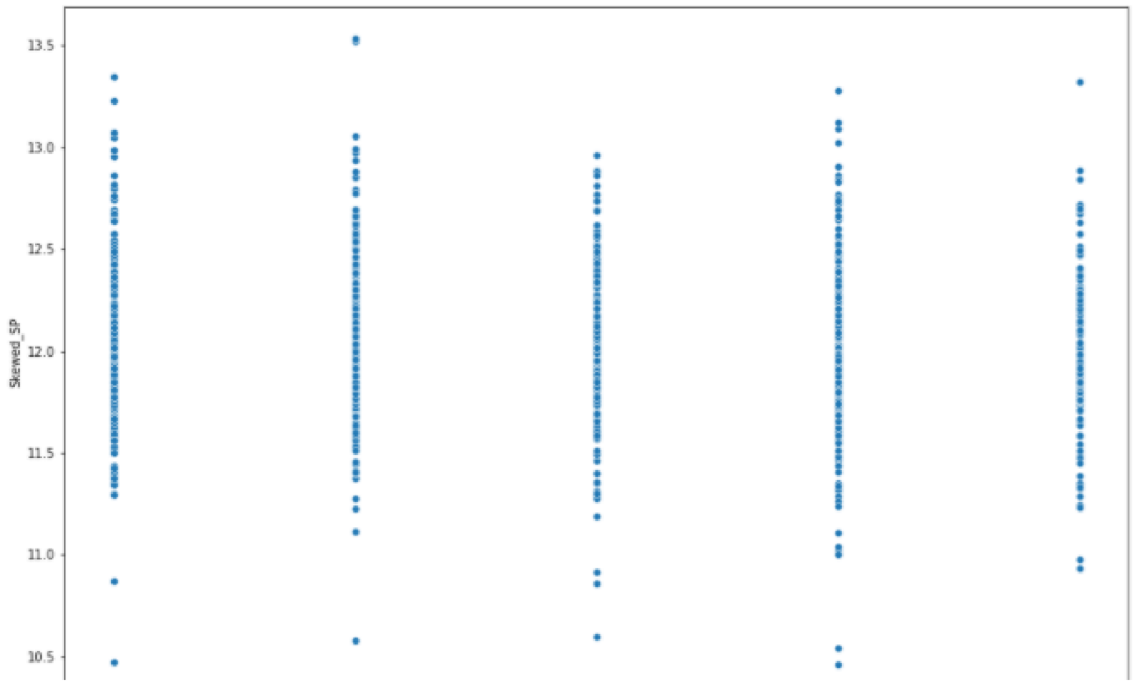


1. The Data mostly all numeric varaibles are skewed
2. Sales Condition is Normal and sale type is Warranty DEED-Conventional
3. Mostly are Paved Drive and Garage Conditions and Garage Quality are Typically/Average
4. Mostly GarageFinish are unfinished and GarageType is Attached
5. Mostly Home Functionality is Typical
6. KitchenQuality is Typically Average
7. Electrical is Standard Circuit Breakers & Romex
8. Central Air is Yes
9. Heating Quality COndition is Excellent
10. Heating is Gas forced warm air furnace
11. Basement Finish Type2 is Unfurnished
12. Basement Finish Type1 is Unfurnished
13. No Exposure to walkout or garden level walls
14. VInylSD are mostly used material for rooms

As per the Analysis from the univariate variables written above.



```
plt.figure(figsize=[15,10])
sns.scatterplot(x='Exterior1st',y='Skewed_SP',data=df_train)
plt.show()
plt.figure(figsize=[15,10])
sns.scatterplot(x='YrSold',y='Skewed_SP',data=df_train)
plt.show()
```

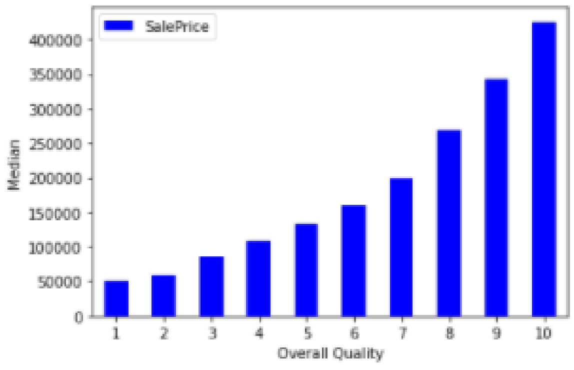


```
df_train.OverallQual.unique()
```

```
array([ 6,  8,  7,  5,  9,  1,  2,  4,  3, 10], dtype=int64)
```

```
#Creating a pivot table
quality_pivot = df_train.pivot_table(index='OverallQual',values='SalePrice', aggfunc=np.median)
```

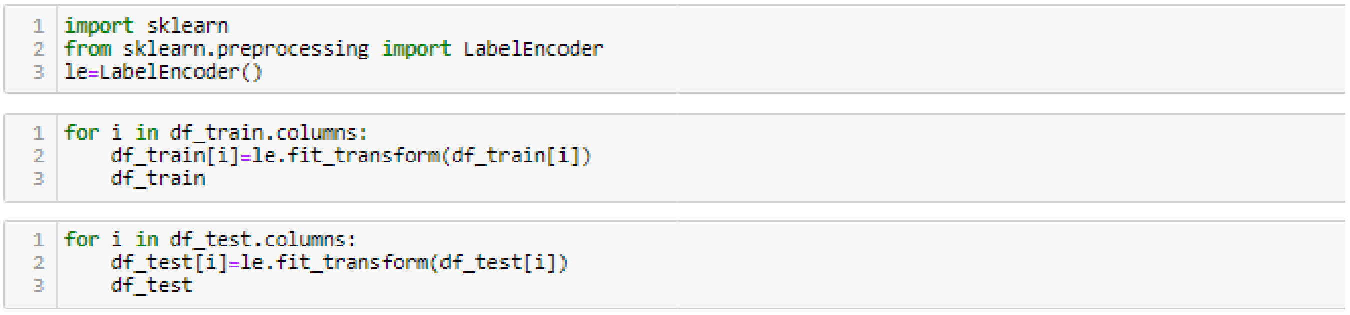
```
quality_pivot.plot(kind='bar',color='blue')
plt.xlabel('Overall Quality')
plt.ylabel('Median')
plt.xticks(rotation=0)
plt.show()
```



If Quality Increases Sales Price also increases

Label Encoding:

Label Encoding is necessary for the data to process to find any outliers are there as of our data consists of both numerical and categorical need to change categorical into numerical values using the encoding methods.

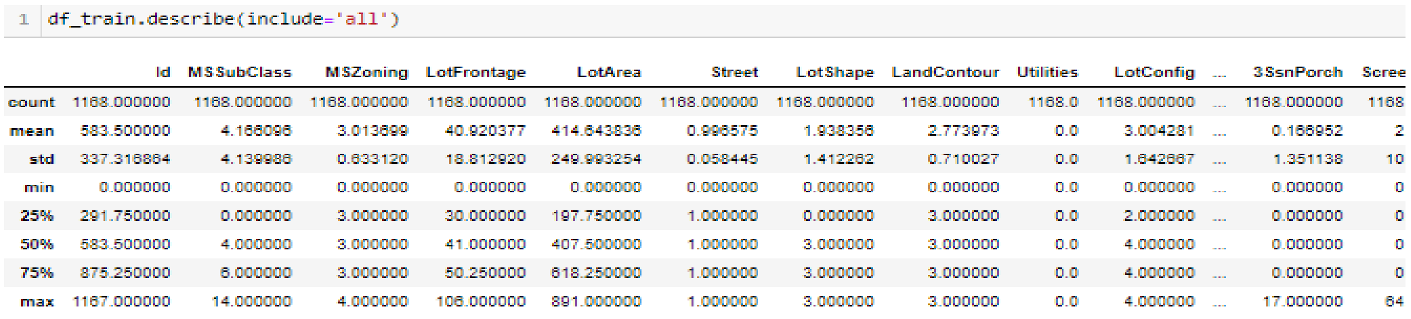


```
import sklearn
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

```
for i in df_train.columns:
    df_train[i]=le.fit_transform(df_train[i])
    df_train
```

```
for i in df_test.columns:
    df_test[i]=le.fit_transform(df_test[i])
    df_test
```

Now our Data set is full of integers, so go for correlation



```
df_train.describe(include='all')
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	...	3SsnPorch	Scree
count	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.0	1168.000000	...	1168.000000	1168
mean	583.500000	4.166096	3.013699	40.920377	414.643836	0.996575	1.938356	2.773973	0.0	3.004281	...	0.166952	2
std	337.316864	4.139986	0.633120	18.812920	249.993254	0.058445	1.412262	0.710027	0.0	1.642667	...	1.351138	10
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	...	0.000000	0
25%	291.750000	0.000000	3.000000	30.000000	197.750000	1.000000	0.000000	3.000000	0.0	2.000000	...	0.000000	0
50%	583.500000	4.000000	3.000000	41.000000	407.500000	1.000000	3.000000	3.000000	0.0	4.000000	...	0.000000	0
75%	875.250000	6.000000	3.000000	50.250000	618.250000	1.000000	3.000000	3.000000	0.0	4.000000	...	0.000000	0
max	1167.000000	14.000000	4.000000	106.000000	891.000000	1.000000	3.000000	3.000000	0.0	4.000000	...	17.000000	64

As per the describe analysis, need to drop some more variables to make our model is accurate

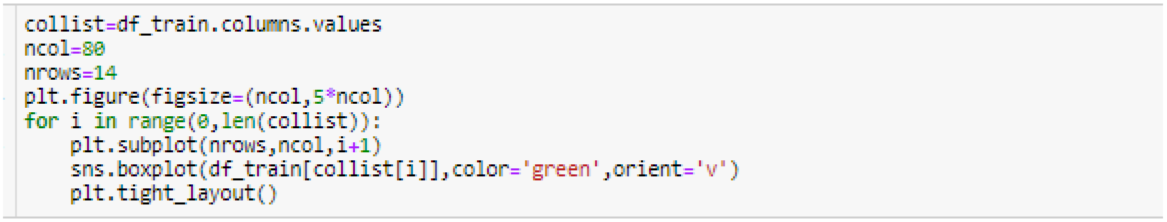
df_train=df_train.drop(['SalePrice', 'Id','Utilities','GarageArea','WoodDeckSF','LowQualFinSF','YearBuilt','YearRemodAdd','GrLivArea','GarageYrBlt','EnclosedPorch','3SsnPorch','PoolArea','MiscVal','LandSlope','LandContour','GarageQual','GarageCond','PavedDrive','ScreenPorch','MSZoning','Street','Condition1','Condition2','BldgType','RoofStyle','RoofMatl','ExterCond','BsmtFinType2','BsmtFinSF2','Heating','CentralAir','Electrical','BsmtHalfBath','KitchenAbvGr','Functional','SaleType','SaleCondition'], axis=1)

Do for testing also parallely…..

Checking Any Outliers in our Data and Remove it:

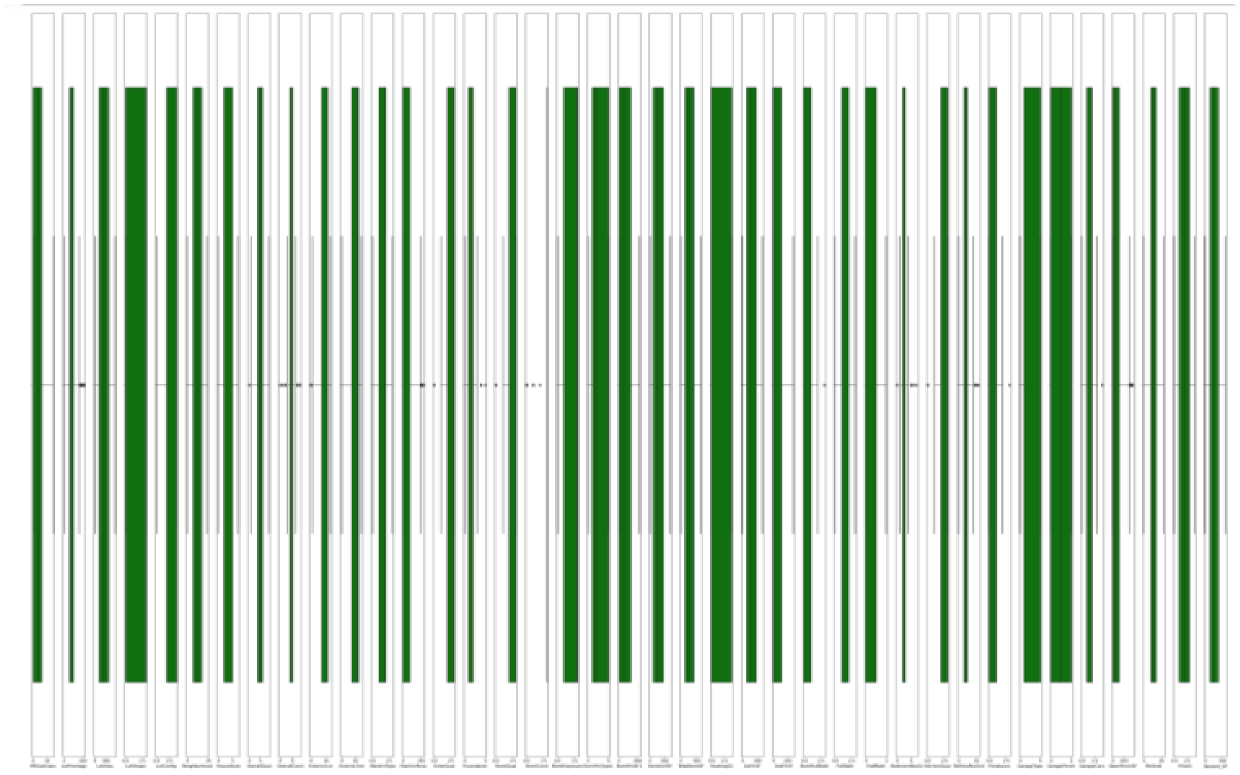
It is defined as the points that are far away from the same points.it can be happen because of the variability of the measurements and may be some error also. If possible, outliers should be removed from the datasets. There are servals methos to remove the outliers. 1)Z score 2) Quantile Method (Capping the data)

1)Z Score: it can call from the SciPy. Stats library. And for most of the case threshold values should be used 3.

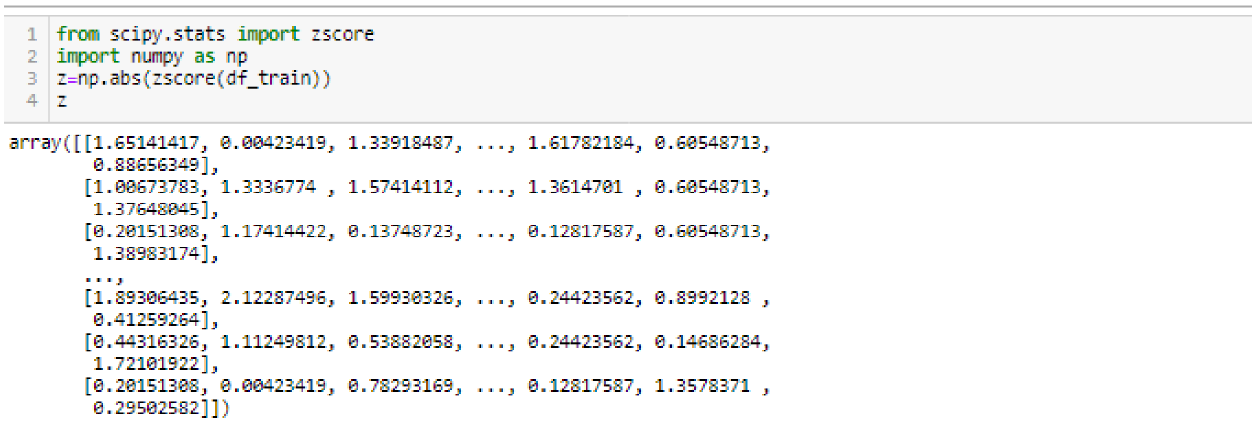


```
collist=df_train.columns.values
ncol=80
nrows=14
plt.figure(figsize=(ncol,5*ncol))
for i in range(0,len(collist)):
    plt.subplot(nrows,ncol,i+1)
    sns.boxplot(df_train[collist[i]],color='green',orient='v')
    plt.tight_layout()
```

By using the Above formula we need to find outliers. And removing using z-score method

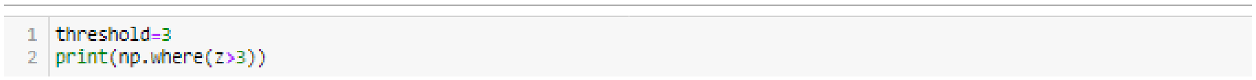


As above shows our data having the outliers so need to remove outlier by one of the prominent method called z-score method

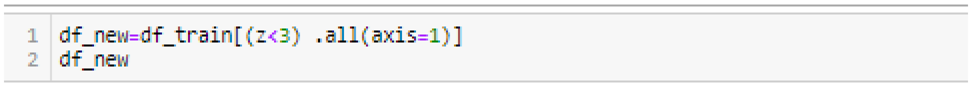


```python
from scipy.stats import zscore
import numpy as np
z=np.abs(zscore(df_train))
z
```

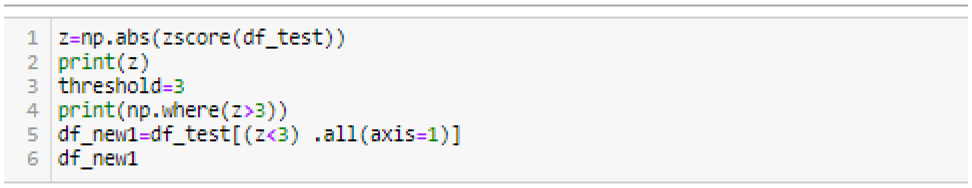
```
array([[1.65141417, 0.00423419, 1.33918487, ..., 1.61782184, 0.60548713,
         0.88656349],
       [1.00673783, 1.3336774 , 1.57414112, ..., 1.3614701 , 0.60548713,
         1.37648045],
       [0.20151308, 1.17414422, 0.13748723, ..., 0.12817587, 0.60548713,
         1.38983174],
       ...,
       [1.89306435, 2.12287496, 1.59930326, ..., 0.24423562, 0.8992128 ,
         0.41259264],
       [0.44316326, 1.11249812, 0.53882058, ..., 0.24423562, 0.14686284,
         1.72101922],
       [0.20151308, 0.00423419, 0.78293169, ..., 0.12817587, 1.3578371 ,
         0.29502582]])
```



```python
threshold=3
print(np.where(z>3))
```



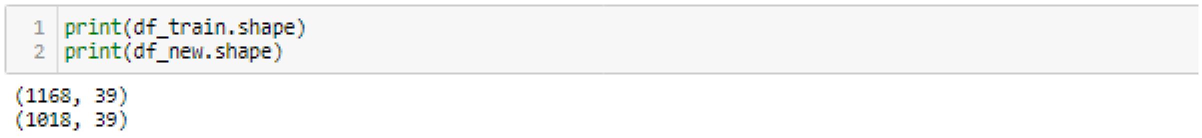
```python
df_new=df_train[(z<3) .all(axis=1)]
df_new
```



```python
z=np.abs(zscore(df_test))
print(z)
threshold=3
print(np.where(z>3))
df_new1=df_test[(z<3) .all(axis=1)]
df_new1
```

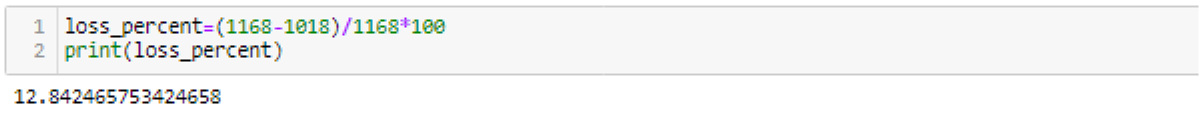
From using above formula we are removing all the outliers

For checking how much data is clean we need to subtract the old data shape from new data shape and then divide with old data shape



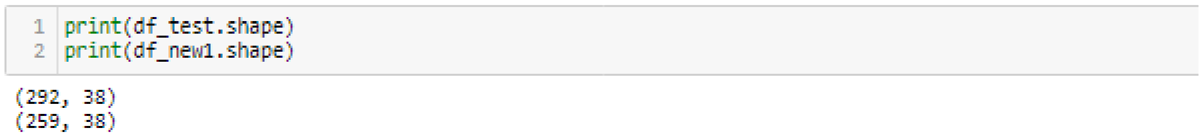
```python
print(df_train.shape)
print(df_new.shape)
```

```
(1168, 39)
(1018, 39)
```



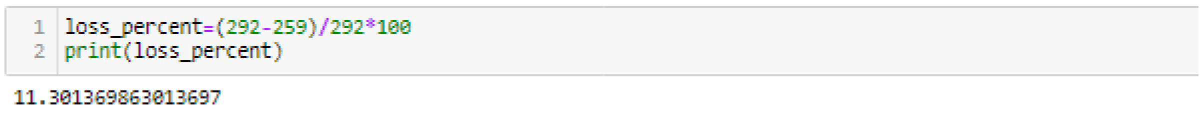
```python
loss_percent=(1168-1018)/1168*100
print(loss_percent)
```

```
12.842465753424658
```



```python
print(df_test.shape)
print(df_new1.shape)
```

```
(292, 38)
(259, 38)
```



```python
loss_percent=(292-259)/292*100
print(loss_percent)
```

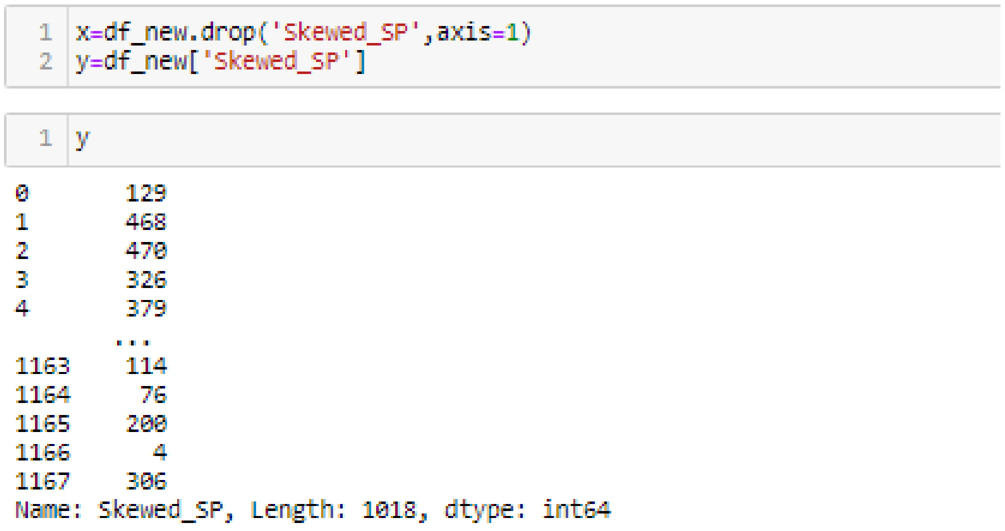
```
11.301369863013697
```

so almost 12% data in train set and 11% data in test data is removed for making our model accurate

Almost 12% of our train data and 11% of test data is removed

2)Quantile Methods: Inter Quantile Range is used to detect or cap the outliers. Calculate the IQR by scipy.stats.iqr Multiply Interquartile range by 1.5 Add 1.5 x interquartile range to the third quartile. Any number greater than this is a suspected outlier. Subtract 1.5 x interquartile range from the first quartile. Any number lesser than this is a suspected outlier.

Now our Data is ready for modelling as our data is clean so only thing need to do is normalizing the data before need to check our dependent variable .



```
x=df_new.drop('Skewed_SP',axis=1)
y=df_new['Skewed_SP']
```

```
y
```

```
0       129
1       468
2       470
3       326
4       379
       ...
1163    114
1164     76
1165    200
1166      4
1167    306
Name: Skewed_SP, Length: 1018, dtype: int64
```

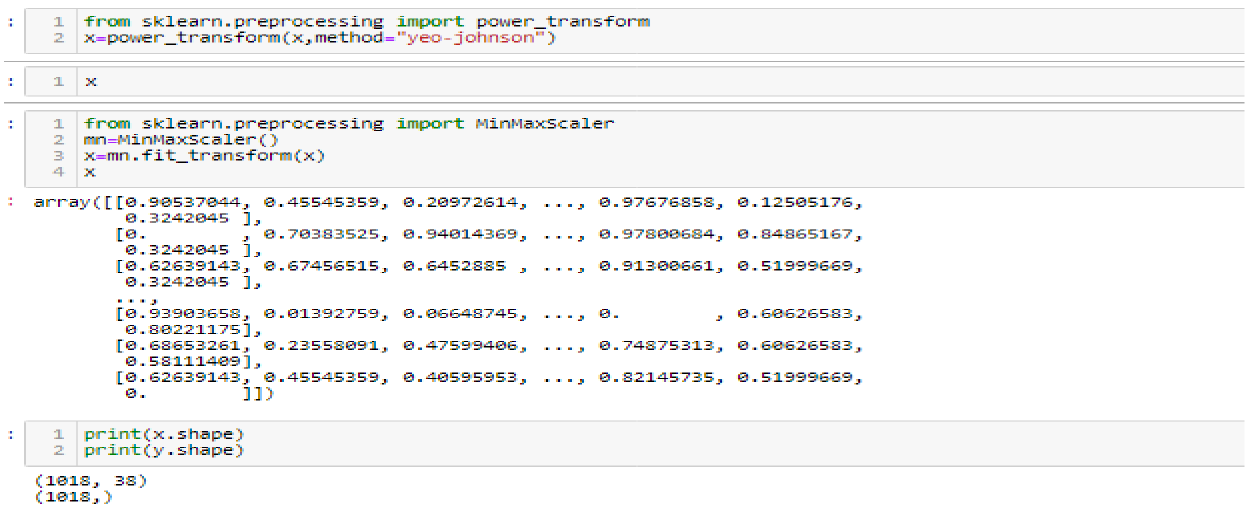
Skewness of Data:

As of our numeric data is skewed we need to do normalization before go for training and testing for that need to check the skewness of data if our data is Greater than 0.5% in both positive and negative sides ,then need to do power transformation and do scaling

Scaling are of two types:

1.**Standard Scaler:** Standard scalar standardizes features of the data set by scaling to unit variance and removing the mean (optionally) using column summary statistics on the samples in the training set.

MIN-MAX Scaler: MinMaxScaler. For each value in a feature, MinMaxScaler 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. MinMaxScaler preserves the shape of the original distribution.



```
from sklearn.preprocessing import power_transform
x=power_transform(x,method="yeo-johnson")
```

```
x
```

```
from sklearn.preprocessing import MinMaxScaler
mn=MinMaxScaler()
x=mn.fit_transform(x)
x
```

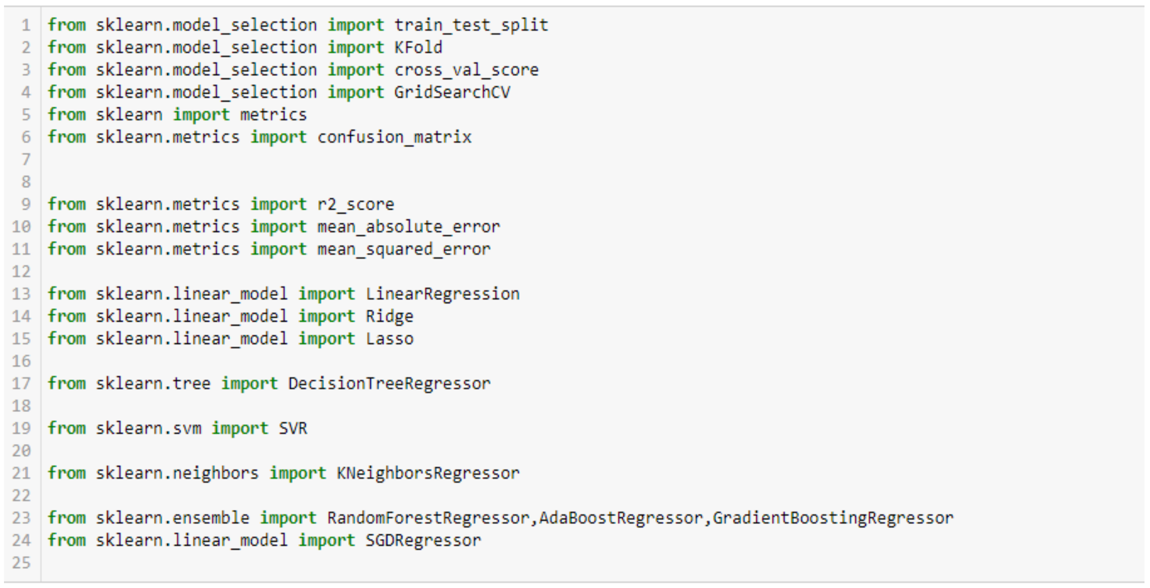
```
array([[0.90537044, 0.45545359, 0.20972614, ..., 0.97676858, 0.12505176,
        0.3242045 ],
       [0.        , 0.70383525, 0.94014369, ..., 0.97800684, 0.84865167,
        0.3242045 ],
       [0.62639143, 0.67456515, 0.6452885 , ..., 0.91300661, 0.51999669,
        0.3242045 ],
       ...,
       [0.93903658, 0.01392759, 0.06648745, ..., 0.        , 0.60626583,
        0.80221175],
       [0.68653261, 0.23558091, 0.47599406, ..., 0.74875313, 0.60626583,
        0.58111409],
       [0.62639143, 0.45545359, 0.40595953, ..., 0.82145735, 0.51999669,
        0.        ]])
```

```
print(x.shape)
print(y.shape)
```

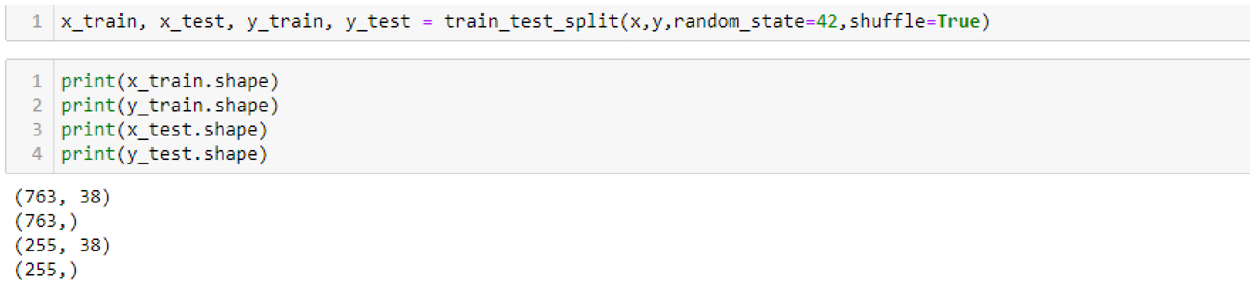
```
(1018, 38)
(1018,)
```

Splitting Data Into train_test_split: -

This function is in sklearn. Model selection splitting the data array into two arrays. Train and Test with this function we don't need to splitting train and test manually.by default it make random partition and we can also set the random state.it gives four o/p like x_train, x_test, y_train, y_test.



```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import confusion_matrix


from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

from sklearn.tree import DecisionTreeRegressor

from sklearn.svm import SVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.linear_model import SGDRegressor

```



```
x_train, x_test, y_train, y_test = train_test_split(x,y,random_state=42,shuffle=True)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(763, 38)
(763,)
(255, 38)
(255,)
```

As we do splitting the data for training and testing the data then now we need to modeling

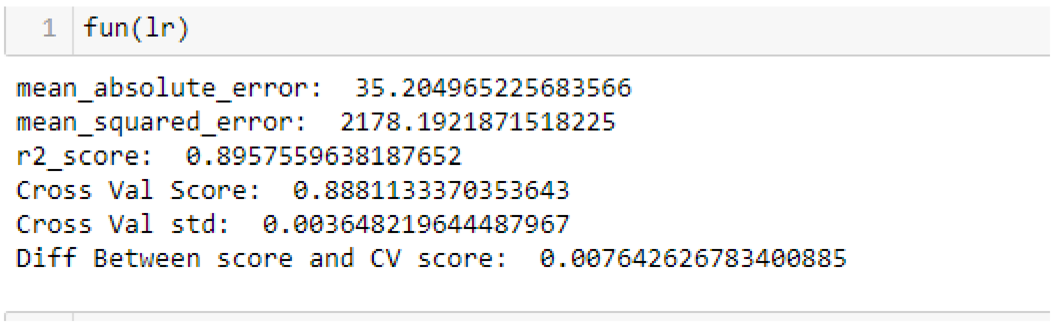
Try Different Models....

1)**Linear Regression:** - Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

```
lr=LinearRegression()
rid=Ridge(alpha=1.0)
ls=Lasso()
dtr=DecisionTreeRegressor()
knr=KNeighborsRegressor()
svr=SVR()
rf=RandomForestRegressor()
ad=AdaBoostRegressor()
gd=GradientBoostingRegressor()
sgd=SGDRegressor()
```

```
def fun(f):
    f.fit(x_train,y_train)
    y_pred = f.predict(x_test)

    mean_absolute_error_ = mean_absolute_error(y_test,y_pred)
    mean_squared_error_ = mean_squared_error(y_test,y_pred)
    r2_score_ = r2_score(y_test,y_pred)

    kFold = KFold(n_splits=3,shuffle=True,random_state=42)
    scores = cross_val_score(f,x,y,cv=kFold,scoring='r2',n_jobs=-1)

    mean_cv_scores = np.mean(scores)
    std_cv_scores = np.std(scores)

    diff_acc_score_cv_score = np.abs(r2_score_ - mean_cv_scores)

    print('mean_absolute_error: ',mean_absolute_error_)
    print('mean_squared_error: ',mean_squared_error_)
    print('r2_score: ',r2_score_)

    print('Cross Val Score: ',mean_cv_scores)
    print('Cross Val std: ',std_cv_scores)
    print('Diff Between score and CV score: ', diff_acc_score_cv_score)


```

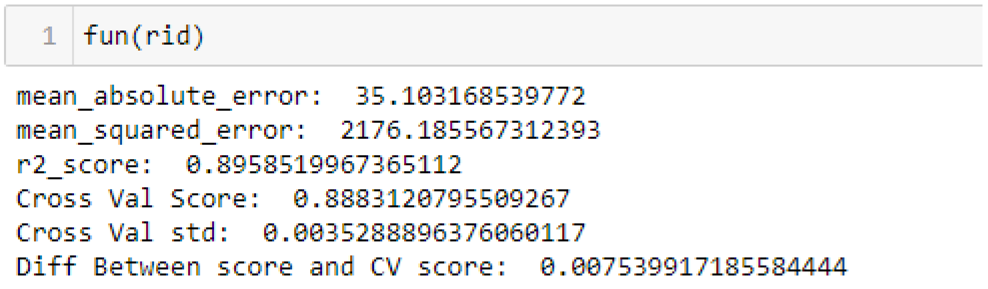
I simply used function method, so no need to write the code each and every time, just call the model



```
fun(lr)
```

```
mean_absolute_error:  35.204965225683566
mean_squared_error:  2178.1921871518225
r2_score:  0.8957559638187652
Cross Val Score:  0.8881133370353643
Cross Val std:  0.003648219644487967
Diff Between score and CV score:  0.007642626783400885
```

2) Ridge Regression: Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values.

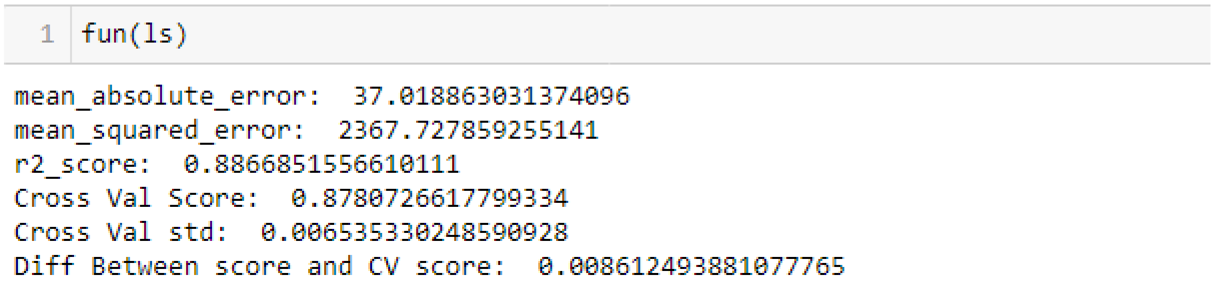


```
fun(rid)
```

```
mean_absolute_error:  35.103168539772
mean_squared_error:  2176.185567312393
r2_score:  0.8958519967365112
Cross Val Score:  0.8883120795509267
Cross Val std:  0.0035288896376060117
Diff Between score and CV score:  0.007539917185584444
```

Lasso Regression: Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as 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.

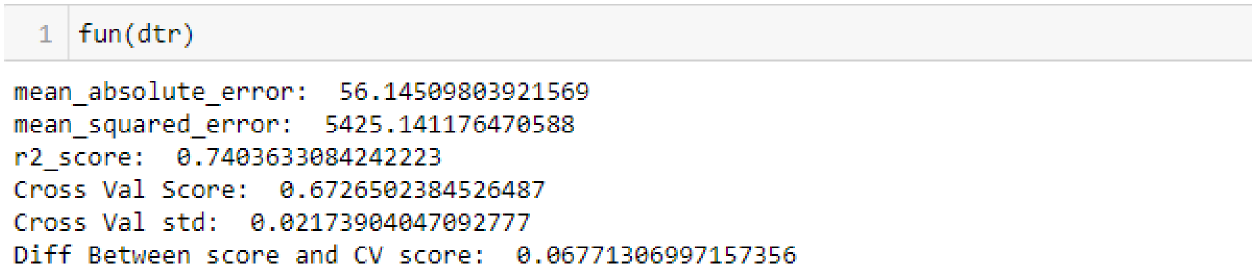
Lasso Regression uses L1 regularization technique (will be discussed later in this article). It is used when we have more number of features because it automatically performs feature selection.



```
fun(ls)
```

```
mean_absolute_error:  37.018863031374096
mean_squared_error:  2367.727859255141
r2_score:  0.8866851556610111
Cross Val Score:  0.8780726617799334
Cross Val std:  0.006535330248590928
Diff Between score and CV score:  0.008612493881077765
```

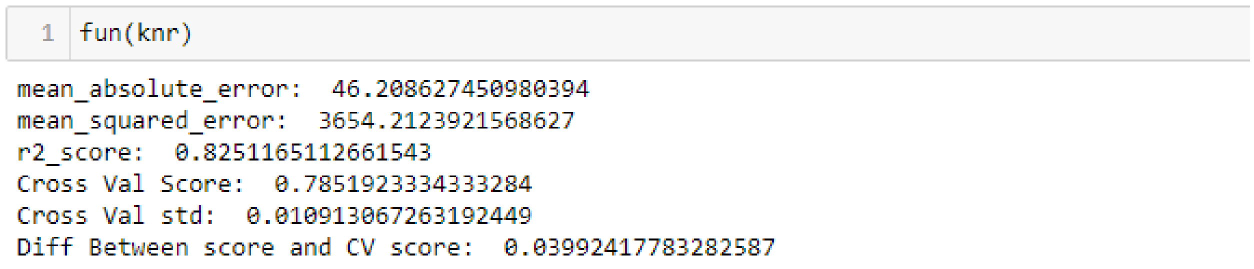
Decision Tree Regression: Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** 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.



```
fun(dtr)
```

```
mean_absolute_error:  56.14509803921569
mean_squared_error:  5425.141176470588
r2_score:  0.7403633084242223
Cross Val Score:  0.6726502384526487
Cross Val std:  0.02173904047092777
Diff Between score and CV score:  0.06771306997157356
```

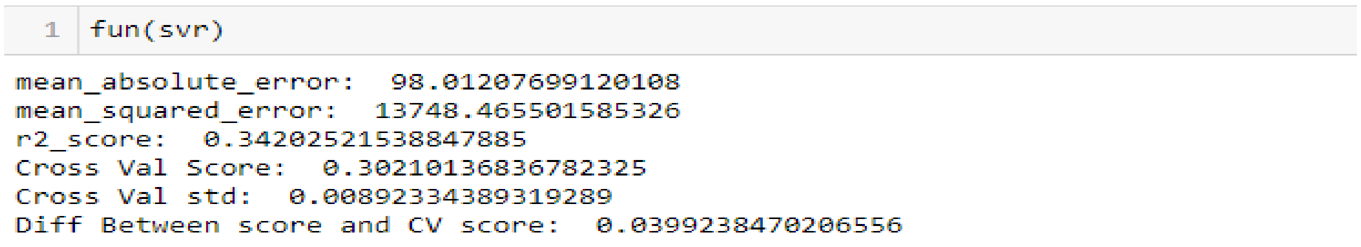
KNN Regressor: KNN regression is a non-parametric method that, in an intuitive manner, approximates the association between independent variables and the continuous outcome by averaging the observations in the same *neighbourhood*. The size of the neighbourhood needs to be set by the analyst or can be chosen using cross-validation (we will see this later) to select the size that minimizes the mean-squared error.



```
fun(knr)
```

```
mean_absolute_error:  46.208627450980394
mean_squared_error:  3654.2123921568627
r2_score:  0.8251165112661543
Cross Val Score:  0.7851923334333284
Cross Val std:  0.010913067263192449
Diff Between score and CV score:  0.03992417783282587
```

SVR Regressor: Support Vector Regression (SVR) is a regression algorithm and it applies a similar technique of Support Vector Machines (SVM) for regression analysis. As we know, regression data contains continuous real numbers. To fit such type of data, the SVR model approximates the best values with a given margin called ε-tube (epsilon-tube, ε identifies a tube width) with considering the model complexity and error rate.

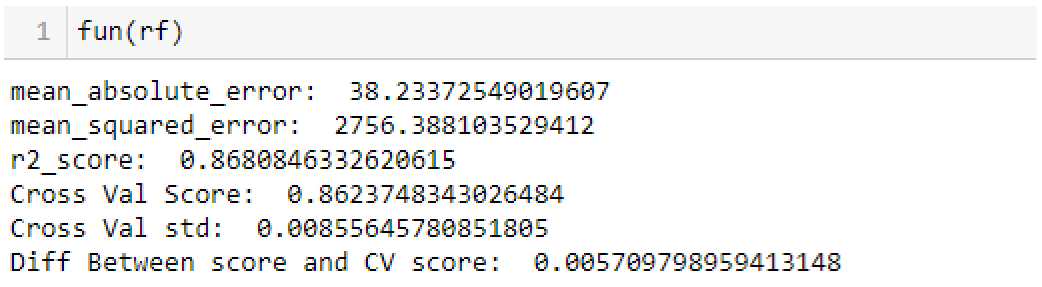


```
fun(svr)
```

```
mean_absolute_error:  98.01207699120108
mean_squared_error:  13748.465501585326
r2_score:  0.34202521538847885
Cross Val Score:  0.30210136836782325
Cross Val std:  0.008923343893119289
Diff Between score and CV score:  0.0399238470206556
```

Random Forest Regression: Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Fortunately, there's no need to combine a decision tree with a bagging classifier because you can easily use the classifier-class of random forest. With random forest, you can also deal with regression tasks by using the algorithm's regressor.



```
fun(rf)
```

```
mean_absolute_error:  38.23372549019607
mean_squared_error:  2756.388103529412
r2_score:  0.8680846332620615
Cross Val Score:  0.8623748343026484
Cross Val std:  0.00855645780851805
Diff Between score and CV score:  0.005709798959413148
```

Ada Boost Regression: An 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. As such, subsequent regressors focus more on difficult cases.

```
fun(ad)
```

```
mean_absolute_error:  44.85191210440428
mean_squared_error:  3361.2825776910363
r2_score:  0.8391355617236196
Cross Val Score:  0.8335198124133797
Cross Val std:  0.012259652288031058
Diff Between score and CV score:  0.0056157493102398925
```

Gradient Boost Regressor: Gradient boosting is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to **set the target outcomes for this next model in** order to minimize the error.

```
fun(gd)
```

```
mean_absolute_error:  34.44113664027432
mean_squared_error:  2234.6351676557183
r2_score:  0.8930547126910957
Cross Val Score:  0.8859073161581277
Cross Val std:  0.012191549603981226
Diff Between score and CV score:  0.007147396532968031
```

Stochastic Gradient Regressor: Stochastic Gradient Descent (SGD) regressor basically implements a plain SGD learning routine supporting various loss functions and penalties to fit linear regression models. Scikit-learn provides SGDRegressor module to implement SGD regression.

```
fun(sgd)
```

```
mean_absolute_error:  36.21968016755267
mean_squared_error:  2339.014005447402
r2_score:  0.8880593447857782
Cross Val Score:  0.8802828573828604
Cross Val std:  0.004633494834691614
Diff Between score and CV score:  0.007776487402917809
```

Now Lets Discuss About each variable in output:

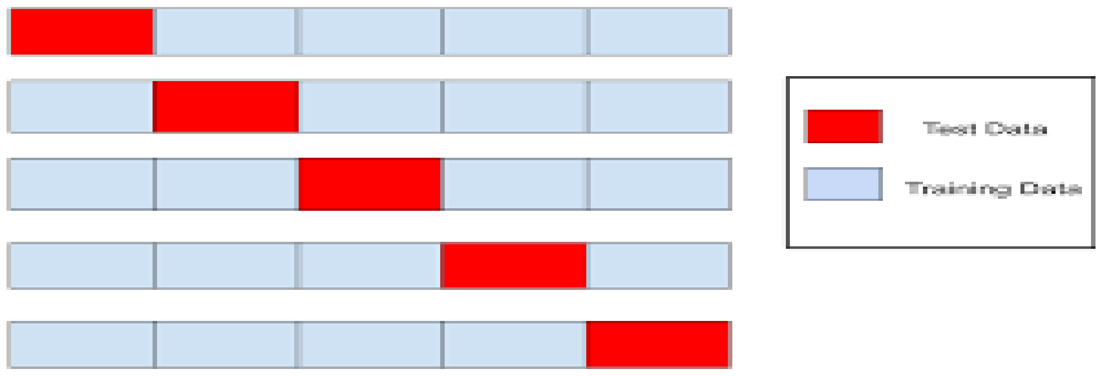
Mean Absolute Error: In statistics, mean absolute error (MAE) is **a measure of errors between paired observations expressing the same phenomenon**. ... This is known as a scale-dependent accuracy measure and therefore cannot be used to make comparisons between series using different scales.

Mean Squared Error: Mean Squared Error (MSD) of an estimator measures the average of error squares i.e. the average squared difference between the estimated values and true value. It is a risk function, corresponding to the expected value of the squared error loss. It is always non – negative and values close to zero are better. The MSE is the second moment of the error (about the origin) and thus incorporates both the variance of the estimator and its bias.

R2_score: Coefficient of determination also called as R^2 score is used to evaluate the performance of a linear regression model. It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable(s). It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.

Cross Validation: - This technique is used to check weather out data set is over fitting or under fitting. If model score is high and cv score is less it means model perform well in train dataset but did not perform well in unseen or test dataset. Feature selection is the best way to overcome the overfitting problem. There are 3 ways for the validation. KFold Cross validation score, Hold Out Methods and LOOCV.

KFold: - In this technique it will rotate the data into the k-fold times.

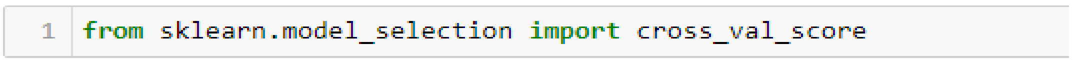


1 st Iteration: 1-3 as Test and 4-9 Train

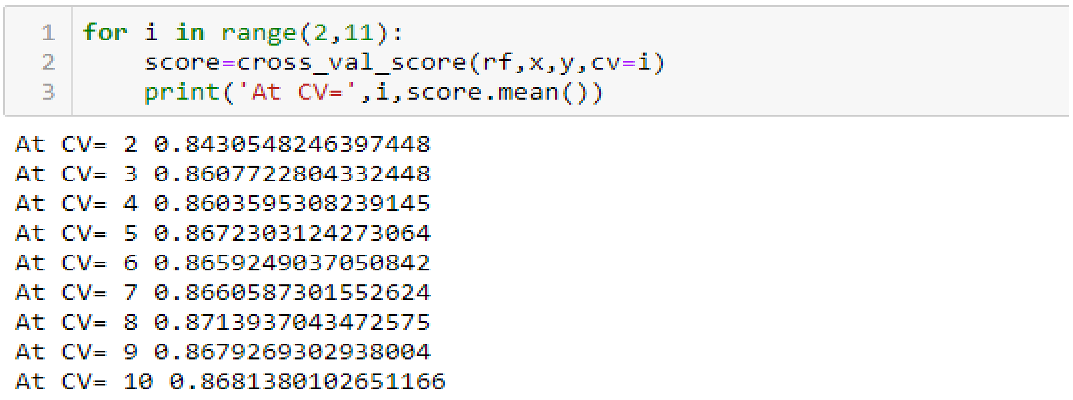
2 nd Iteration:4-6 as Test and 1-3 & 7-9

Train 3 rd Iteration:7-9 as Test and 1-6 as Train

It means all the data (9 rows) go for training.

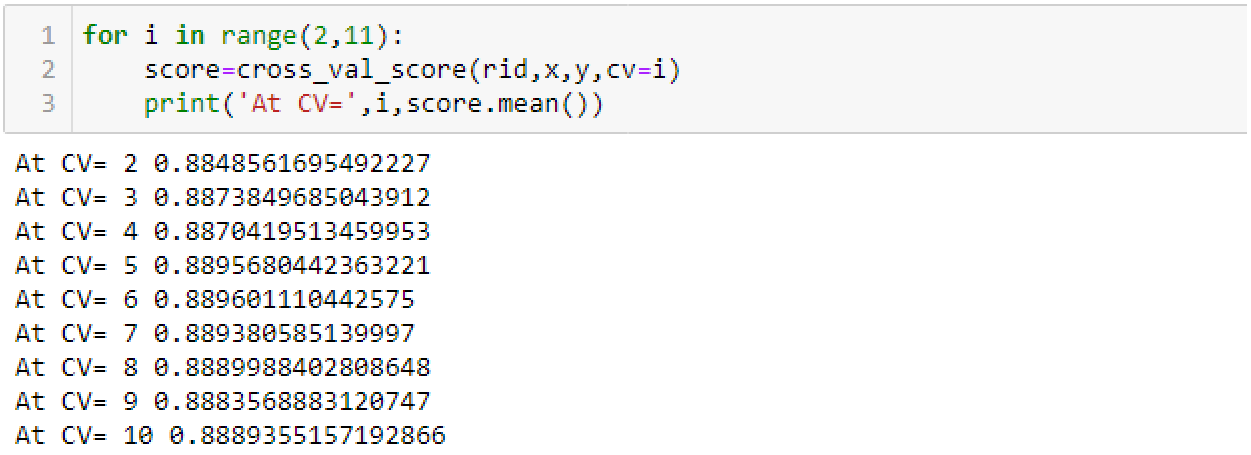


```python
from sklearn.model_selection import cross_val_score
```



```python
for i in range(2,11):
    score=cross_val_score(rf,x,y,cv=i)
    print('At CV=',i,score.mean())
```

```
At CV= 2 0.8430548246397448
At CV= 3 0.8607722804332448
At CV= 4 0.8603595308239145
At CV= 5 0.8672303124273064
At CV= 6 0.8659249037050842
At CV= 7 0.8660587301552624
At CV= 8 0.8713937043472575
At CV= 9 0.8679269302938004
At CV= 10 0.8681380102651166
```



```python
for i in range(2,11):
    score=cross_val_score(rid,x,y,cv=i)
    print('At CV=',i,score.mean())
```

```
At CV= 2 0.8848561695492227
At CV= 3 0.8873849685043912
At CV= 4 0.8870419513459953
At CV= 5 0.8895680442363221
At CV= 6 0.889601110442575
At CV= 7 0.889380585139997
At CV= 8 0.8889988402808648
At CV= 9 0.8883568883120747
At CV= 10 0.8889355157192866
```

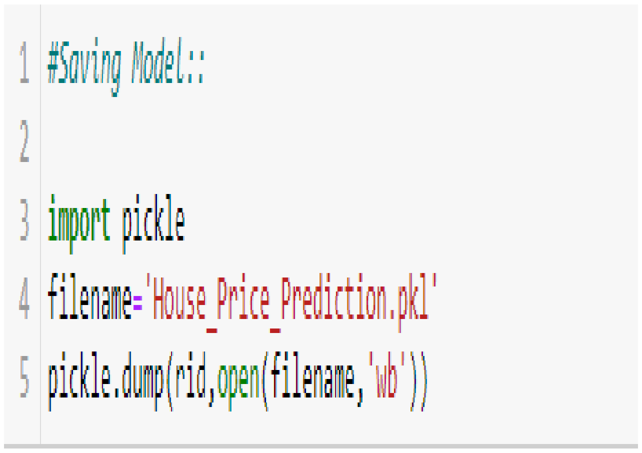
LOOCV: Leave one out cross validation It will take one row for test and remaining for training so each and every row go for test so its time-consuming processing

Concluding Remarks: - From this model we can predict the House price prediction of different variables how the prices are varying according to that each one can take their prescribed house.

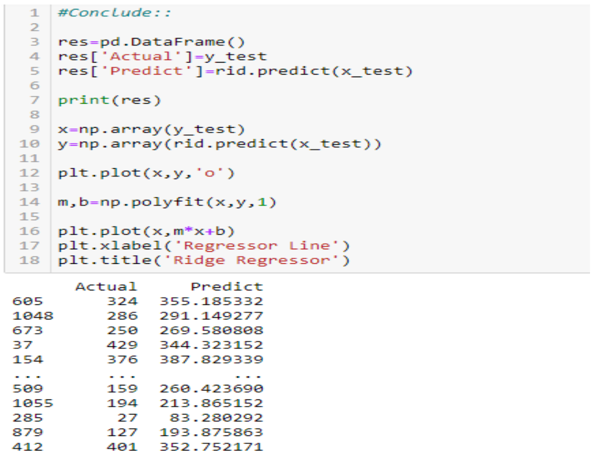
We used different regression methods to test the process of the model like Linear Regression, Decision tree Regression, Lasso Regression, Ridge Regression and Ada boosting Regression, Random Forest Regression, Gradient Regression, SGD Regression .

We get good score in Ridge Regressor got r2_score of 89.58% on training data, mean absolute error is 35.10% , Diff Between score and validation score also nice but not in 1st position but comparatively with all the variables and cross validation value also is high in Ridge Regressor .the model performance is excellent.

We further proceed to test the object that we saved using pickle, and create a data frame of predicted values



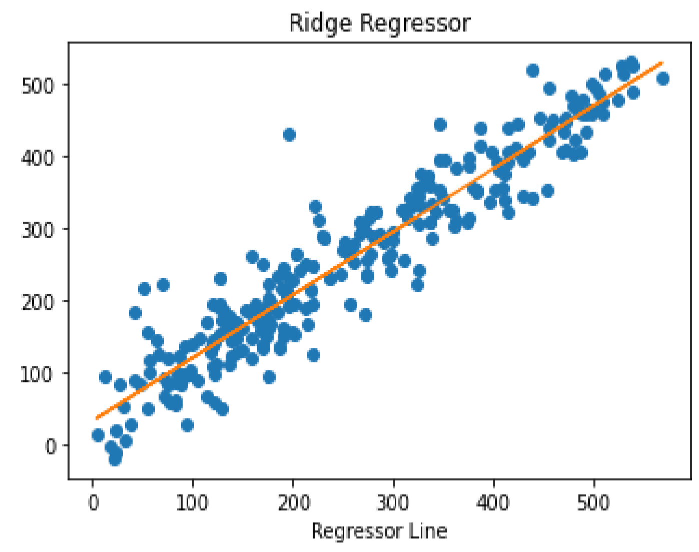
```
#Saving Model::

import pickle
filename='House_Price_Prediction.pkl'
pickle.dump(rid,open(filename,'wb'))
```



```
#Conclude::

res=pd.DataFrame()
res['Actual']=y_test
res['Predict']=rid.predict(x_test)

print(res)

x=np.array(y_test)
y=np.array(rid.predict(x_test))

plt.plot(x,y,'o')

m,b=np.polyfit(x,y,1)

plt.plot(x,m*x+b)
plt.xlabel('Regressor Line')
plt.title('Ridge Regressor')
```

```
      Actual     Predict
605      324  355.185332
1048     286  291.149277
673      250  269.580808
37       429  344.323152
154      376  387.829339
...      ...         ...
509      159  260.423690
1055     194  213.865152
285       27   83.280292
879      127  193.875863
412      401  352.752171
```

```
]: Text(0.5, 1.0, 'Ridge Regressor')
```



Just do the same process on the test data

For predicting price validation using ridge regressor

```
#Conclude::

res=pd.DataFrame()
res['Predict']=rid.predict(x1)
print(res)
```

```
        Predict
0    587.642002
1    320.620224
2    446.754156
3    263.140857
4    404.493146
..          ...
254  287.254938
255  433.114909
256  144.063368
257  248.606012
258  301.905382

[259 rows x 1 columns]
```