NAME OF PROJECT

HOUSING PRICE PREDICTION

SUBMITTED BY

PANKAJ ADHIKARI

1. INTRODUCTION

Owning a House has always been a dream for people who have lived and work in big cities. With the explosion of urban population, demand for Housing has been steadily rising for the past few decades creating a void that needs to be fill. This in turn has provided business opportunity to the real estate developers for building houses and apartment that can meet the supply and demand. In order to attract the customers with various types of housing, developers needs to have the knowhow of housing market, their customers, location, cost of construction and so on. One of such Housing Company named Surprise Housing, which has their presence in US, have decided to bring their expertise to Australia and help in the growing demand. This will in turn also help them in expansion of the business in Australia. To do so they need the help of Data Analysis which will help them buy the house in prices below market value and sell them at higher prices. Hence, they have collected excessive data about Australian Housing market that can help them buy prospective properties and to make a decision on their purchase and sale value.

2. PROBLEM DEFINITION

The Dataset provided different parameters that determine the price of the House. These parameters can be used to build a Machine Learning Model to predict the Sale Price of the House giving the company to decide whether to buy the property or not. This Dataset provide the challenge of analysing and quantifying the parametrs that contribute directly to the Sale Price of the house.

3. DATA ANALYSIS

In this section we perform Exploratory Data Analysis (EDA) so as to gain insight on Data by modification, Visualisation.

IMPORTING LIBRARIES

We begin by importing important Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

UPLOADING DATA

Uploading the data set into a Data Frame and printing the first five rows

```
df=pd.read_csv('train.csv')
pd.options.display.max_columns=82
df.head()
```

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

SHAPE

This show that Data has 1168-rows and 81-columns

```
df.shape
```

```
(1168, 81)
```

INFO

This shows the datatype along with number of data available.

Output:

```
df.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
```

Observation:

- Columns like Alley, PoolQc, Fence, MiscFeature, FireplaceQu have very less data hence these columns has to be dropped.
- Columns like MasVnrType, MasVnrArea, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, GarageType, GarageYrBlt, GarageFinish, GarageQual, GarageCond, GarageFinish have missing data which needs to be filled with mode of data.

DATA DESCRIBE

Describing the important parameters of the data.

Output:

```
pd.options.display.max_columns=82
df.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1168.000000	1168.000000	954.00000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1161.000000	1168.000000	1168.000000
mean	724.136130	56.767979	70.98847	10484.749144	6.104452	5.595890	1970.930651	1984.758562	102.310078	444.726027	46.647260
std	416.159877	41.940650	24.82875	8957.442311	1.390153	1.124343	30.145255	20.785185	182.595606	462.664785	163.520016
min	1.000000	20.000000	21.00000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000
25%	360.500000	20.000000	60.00000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000
50%	714.500000	50.000000	70.00000	9522.500000	6.000000	5.000000	1972.000000	1993.000000	0.000000	385.500000	0.000000
75%	1079.500000	70.000000	80.00000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000
max	1460.000000	190.000000	313.00000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

BsmtUnfSF	TotalBsmtSF	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivArea	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	BedroomAbvGr
1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
569.721747	1061.095034	1169.860445	348.826199	6.380137	1525.066781	0.425514	0.055651	1.562500	0.388699	2.884418
449.375525	442.272249	391.161983	439.696370	50.892844	528.042957	0.521615	0.236699	0.551882	0.504929	0.817229
0.000000	0.000000	334.000000	0.000000	0.000000	334.000000	0.000000	0.000000	0.000000	0.000000	0.000000
216.000000	799.000000	892.000000	0.000000	0.000000	1143.250000	0.000000	0.000000	1.000000	0.000000	2.000000
474.000000	1005.500000	1096.500000	0.000000	0.000000	1468.500000	0.000000	0.000000	2.000000	0.000000	3.000000
816.000000	1291.500000	1392.000000	729.000000	0.000000	1795.000000	1.000000	0.000000	2.000000	1.000000	3.000000
2336.000000	6110.000000	4692.000000	2065.000000	572.000000	5642.000000	3.000000	2.000000	3.000000	2.000000	8.000000

KitchenAbvGr	TotRmsAbvGrd	Fireplaces	GarageYrBlt	GarageCars	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch
1168.000000	1168.000000	1168.000000	1104.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
1.045377	6.542808	0.617295	1978.193841	1.776541	476.860445	96.206336	46.559932	23.015411	3.639555	15.051370
0.216292	1.598484	0.650575	24.890704	0.745554	214.466769	126.158988	66.381023	63.191089	29.088867	55.080816
0.000000	2.000000	0.000000	1900.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	5.000000	0.000000	1961.000000	1.000000	338.000000	0.000000	0.000000	0.000000	0.000000	0.000000
1.000000	6.000000	1.000000	1980.000000	2.000000	480.000000	0.000000	24.000000	0.000000	0.000000	0.000000
1.000000	7.000000	1.000000	2002.000000	2.000000	576.000000	171.000000	70.000000	0.000000	0.000000	0.000000
3.000000	14.000000	3.000000	2010.000000	4.000000	1418.000000	857.000000	547.000000	552.000000	508.000000	480.000000

PoolArea	MiscVal	MoSold	YrSold	SalePrice
1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
3.448630	47.315068	6.344178	2007.804795	181477.005993
44.896939	543.264432	2.686352	1.329738	79105.586863
0.000000	0.000000	1.000000	2006.000000	34900.000000
0.000000	0.000000	5.000000	2007.000000	130375.000000
0.000000	0.000000	6.000000	2008.000000	163995.000000
0.000000	0.000000	8.000000	2009.000000	215000.000000
738.000000	15500.000000	12.000000	2010.000000	755000.000000

Observations:

1. Columns like OverallQual, OverallCond has very low standard deviation signifying data to be clustered around mean value.
2. Column BsmtFinSF2 has high standard deviation signifying data to be highly spread

DROPPING OF COLUMNS

- Columns like Alley, PoolQc, Fence, MiscFeature, FireplaceQu have very less data hence these columns are dropped as shown below.
- Id Column has been drop as the value is all unique and does not provide correct result in prediction.

```
df=df.drop(['Id','PoolQC','Fence','MiscFeature','Alley'],axis=1)
```

MISSING DATA

We Fill the missing data with mode of the data as shown below.

```
print('Mode of MasVnrType is:',df['MasVnrType'].mode())
print('Mode of MasVnrArea is:',df['MasVnrArea'].mode())
print('Mode of BsmtQual is:',df['BsmtQual'].mode())
print('Mode of BsmtCond is:',df['BsmtCond'].mode())
print('Mode of BsmtExposure is:',df['BsmtExposure'].mode())
print('Mode of BsmtFinType1 is:',df['BsmtFinType1'].mode())
print('Mode of BsmtFinType2 is:',df['BsmtFinType2'].mode())
print('Mode of GarageType is:',df['GarageType'].mode())
print('Mode of GarageYrBlt is:',df['GarageYrBlt'].mode())
print('Mode of GarageFinish is:',df['GarageFinish'].mode())
print('Mode of GarageQual is:',df['GarageQual'].mode())
print('Mode of GarageCond is:',df['GarageCond'].mode())
print('Mode of GarageFinish is:',df['GarageFinish'].mode())
```

```
Mode of MasVnrType is: 0    None
dtype: object
Mode of MasVnrArea is: 0    0.0
dtype: float64
Mode of BsmtQual is: 0    TA
dtype: object
Mode of BsmtCond is: 0    TA
dtype: object
Mode of BsmtExposure is: 0    No
dtype: object
Mode of BsmtFinType1 is: 0    Unf
dtype: object
Mode of BsmtFinType2 is: 0    Unf
dtype: object
Mode of GarageType is: 0    Attchd
dtype: object
Mode of GarageYrBlt is: 0    2006.0
dtype: float64
Mode of GarageFinish is: 0    Unf
dtype: object
Mode of GarageQual is: 0    TA
dtype: object
Mode of GarageCond is: 0    TA
dtype: object
Mode of GarageFinish is: 0    Unf
dtype: object
```

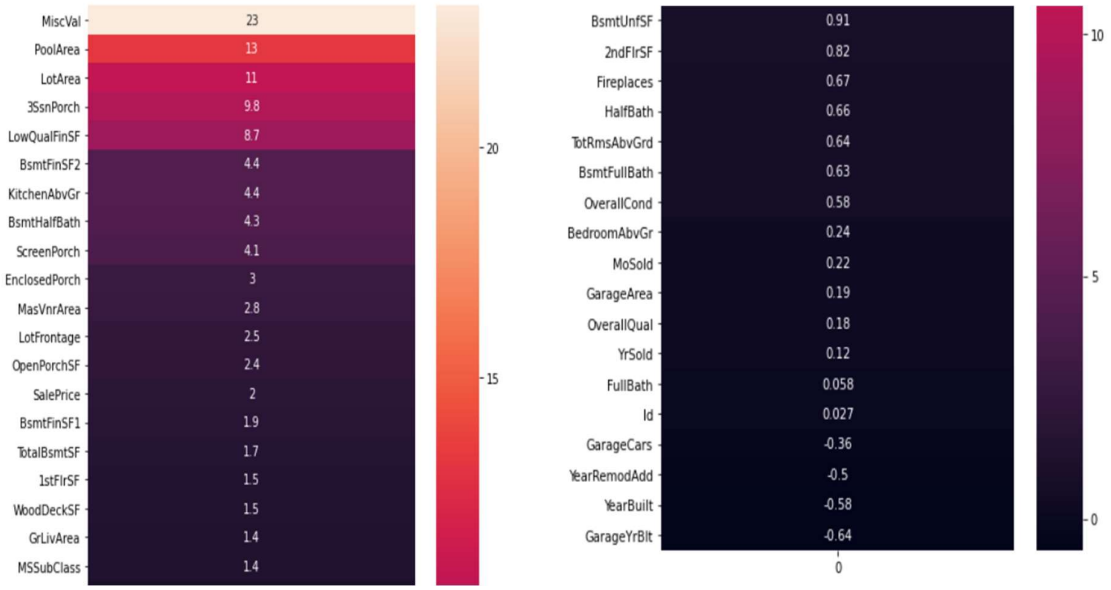
```
df['LotFrontage']=df['LotFrontage'].fillna(41)
df['MasVnrType']=df['MasVnrType'].fillna('None')
df['MasVnrArea']=df['MasVnrArea'].fillna(0.00)
df['BsmtQual']=df['BsmtQual'].fillna('TA')
df['BsmtCond']=df['BsmtCond'].fillna('TA')
df['BsmtExposure']=df['BsmtExposure'].fillna('No')
df['BsmtFinType1']=df['BsmtFinType1'].fillna('Unf')
df['BsmtFinType2']=df['BsmtFinType2'].fillna('Unf')
df['GarageType']=df['GarageType'].fillna('Attchd')
df['GarageYrBlt']=df['GarageYrBlt'].fillna(2006.0)
df['GarageFinish']=df['GarageFinish'].fillna('Unf')
df['GarageQual']=df['GarageQual'].fillna('TA')
df['GarageCond']=df['GarageCond'].fillna('TA')
df['GarageFinish']=df['GarageFinish'].fillna('Unf')
df['FireplaceQu']=df['FireplaceQu'].fillna('NA')
```

SKEWNESS OF COLUMNS

Calculating skewness of Columns and plotting in a Descending Heatmap.

```
skew=df.skew().sort_values(ascending=False)
skew1=pd.DataFrame(skew)
plt.figure(figsize=(7,15))
sns.heatmap(skew1,annot=True)
plt.show()
```

Output:



COORELATION HEAT MAP

Due to high number of columns we divide the heat map into four group for better visualization: df.iloc[:41,:41], df.iloc[:41,41:], df.iloc[41:,:41], df.iloc[41:,41:]

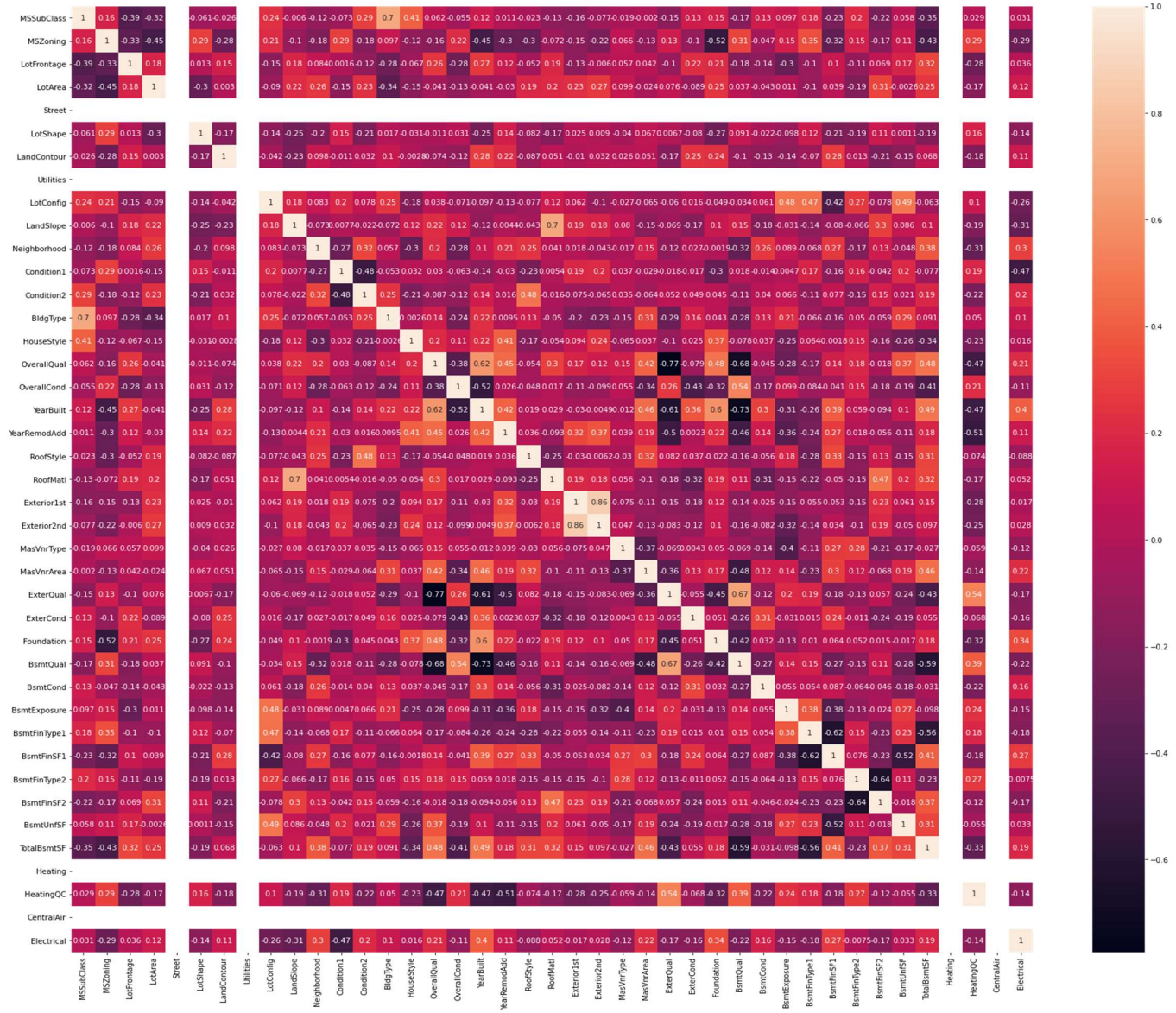


Figure : Heatmap-1

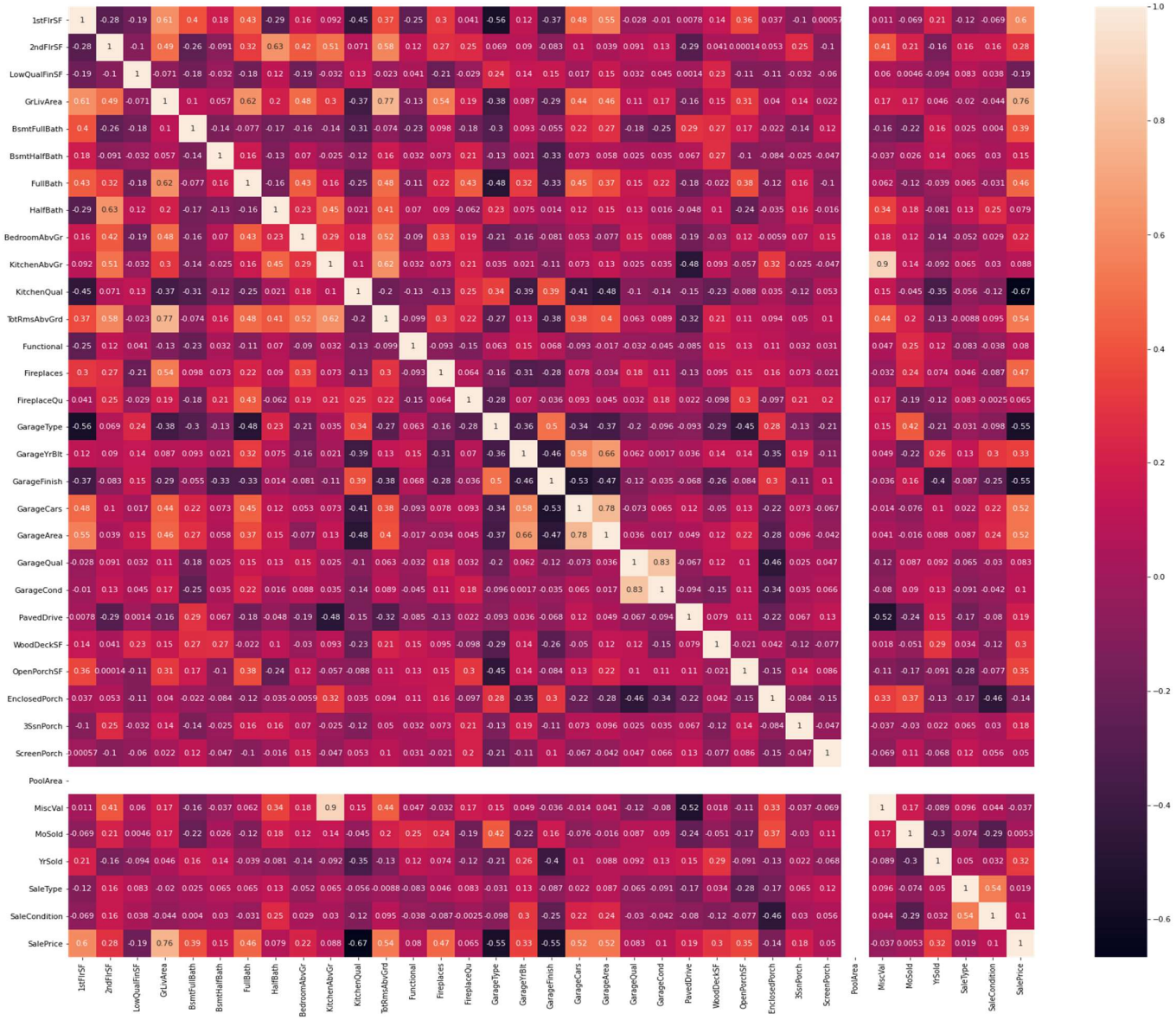


Figure : Heatmap-2

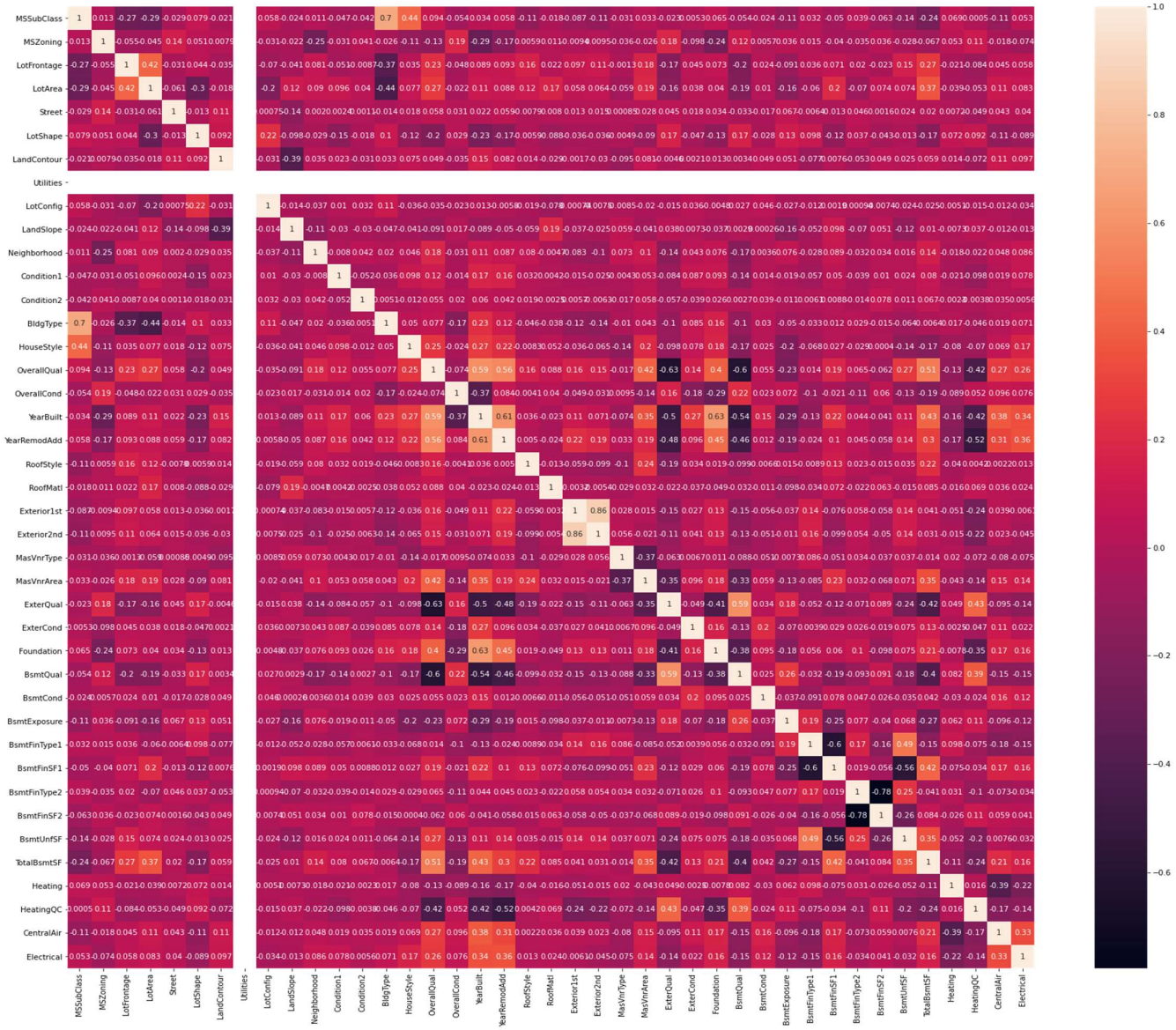


Figure : Heatmap-3

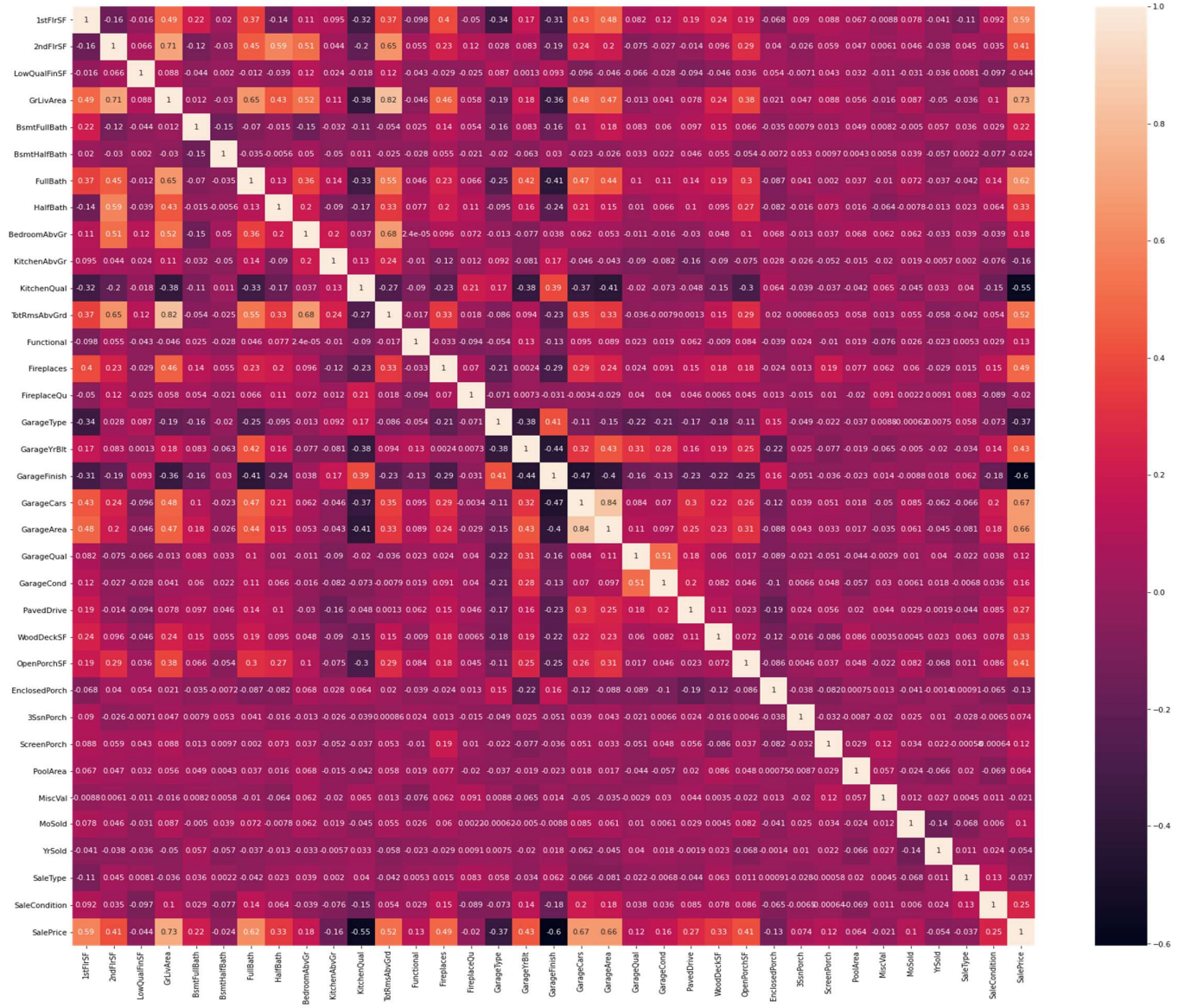


Figure : Heatmap-4

Observation:

- In Heatmap-2 Columns KitchenAbvGr and MiscVal has high Correlation of 0.9
- In Heatmap-4 Columns has high Correlation of 0.82
- Columns SalePrice and GarageFinish has low Correlation of -0.6
- In Heatmap-3 Columns BsmtFinSF2 and BsmtFinType2 has low Correlation of -0.78

PLOTS AND GRAPHS

PLOT-1

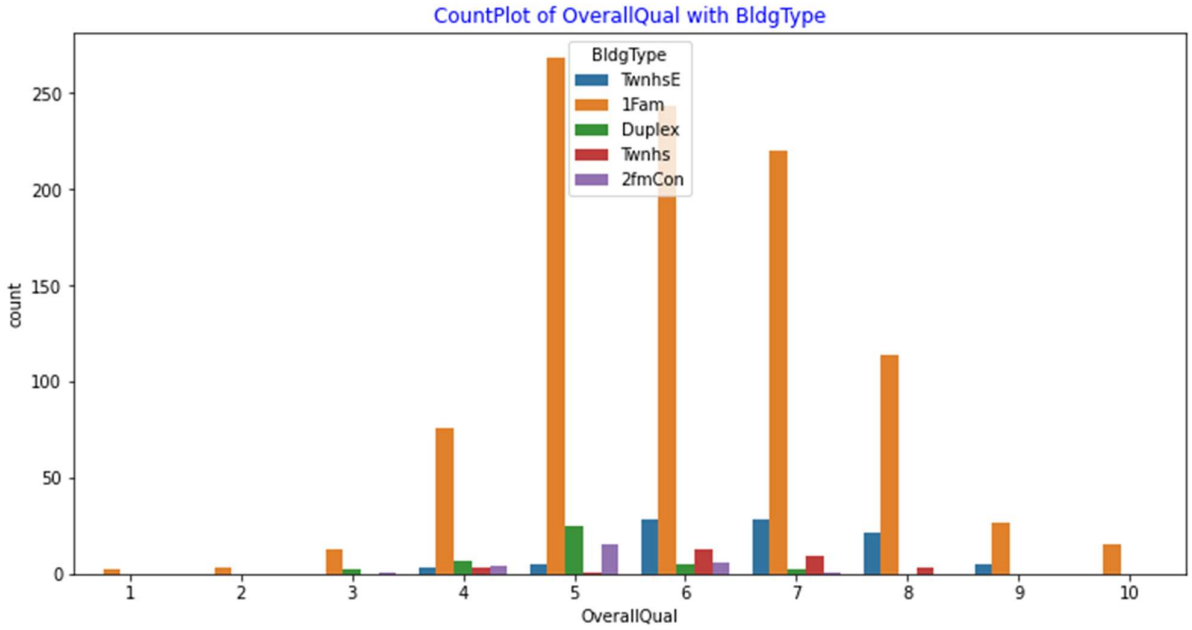


Figure : CountPlot of OverallQual with BldgType

OBSERVATION:

- 1Fam BldgType have high count
- OverallQual of 5 have the highest Count
- OverallQual of 1,2 and 10 have only 1Fam BldgType

PLOT-2

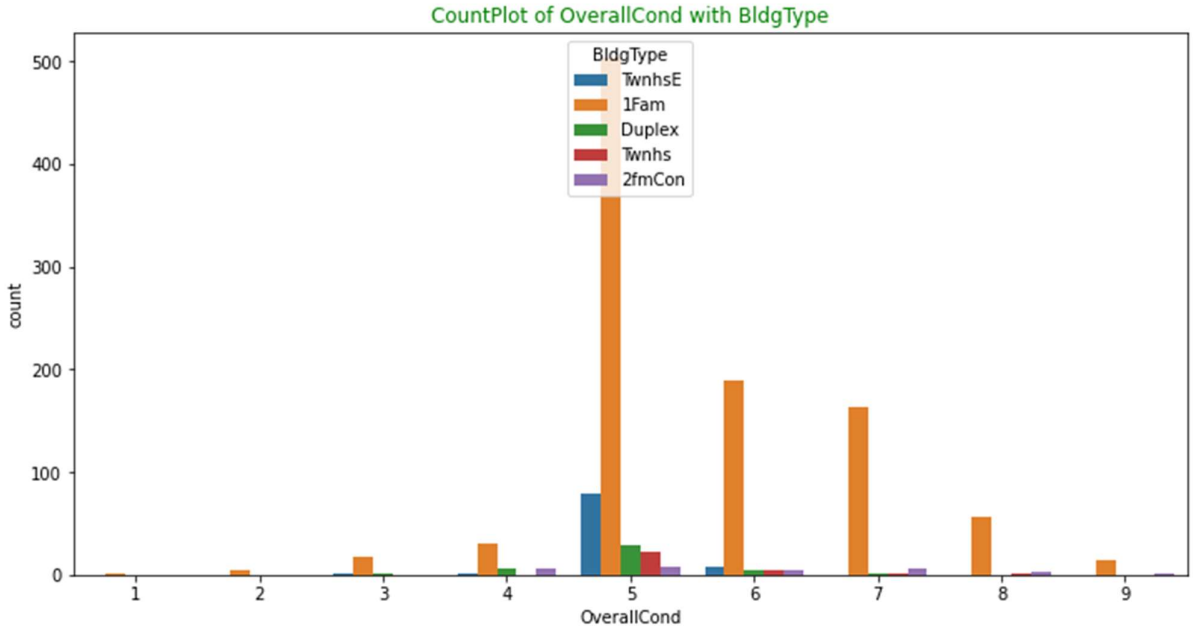


Figure : CountPlot of OverallCond with BldgType

OBSERVATIONS:

- OverallCond of 5 have the highest count
- 1Fam BldgType have the highest count
- OverallCond of 10 have the least count

PLOT-3

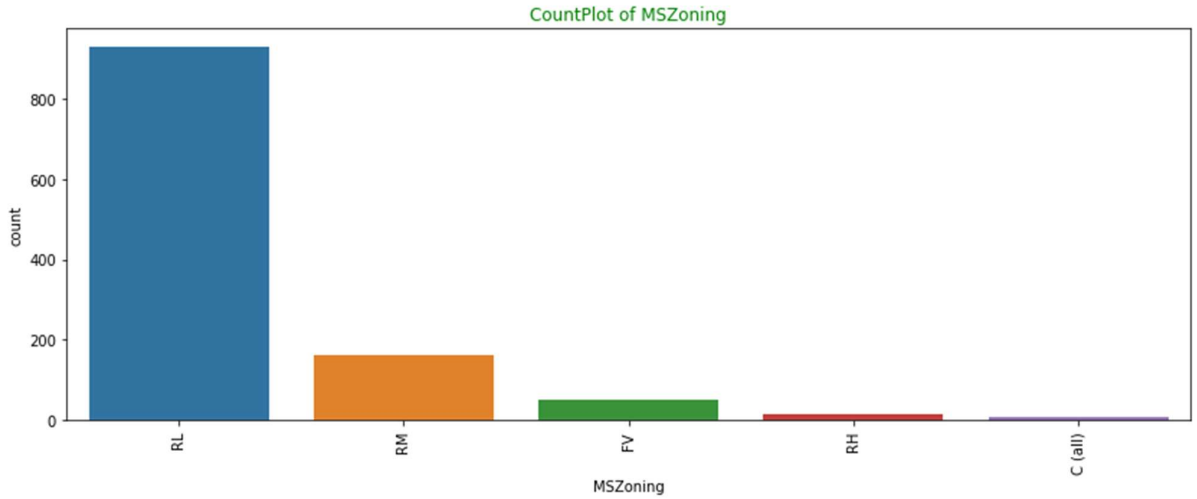


Figure : CountPlot of MSZoning

OBSERVATIONS:

- RL MSZoning have the highest count
- C type MSZoning have the lowest count.

PLOT-4

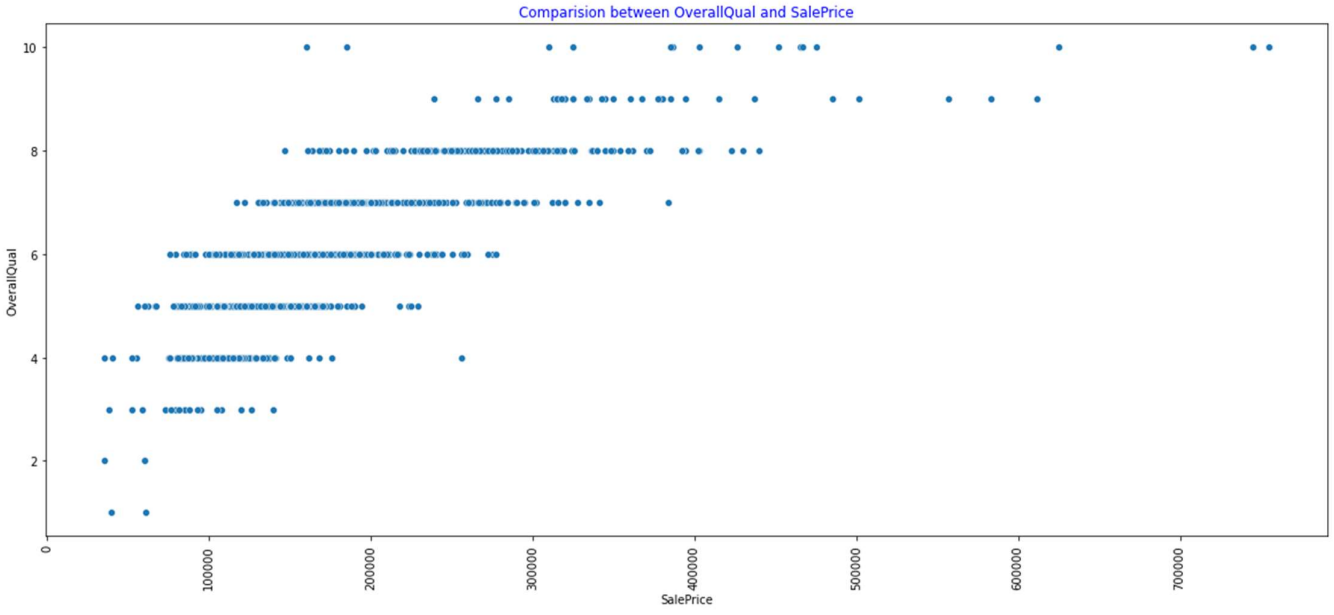


Figure : Comparision between OverallQual and SalePrice

OBSERVATION:

- Highest SalePrice is that of OverallQual 10.
- Lowest SalePrice is that of OverallQual 2.
- OverallQual of 1,2,and 3 is of low SalePrice
- OverallQual 10 has range from low to High SalePrice.

PLOT-5

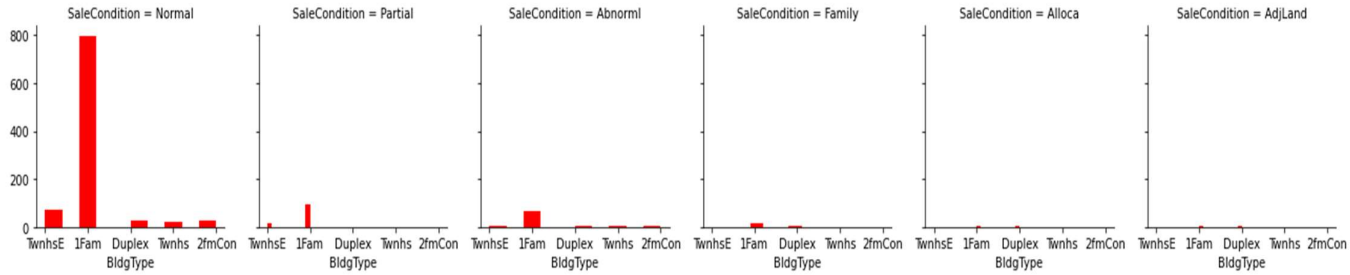


Figure : FacetGrid Plot of SaleCondition with Bldg Type

OBSERVATIONS:

- SaleCondition -Normal has the highest count

PLOT-6

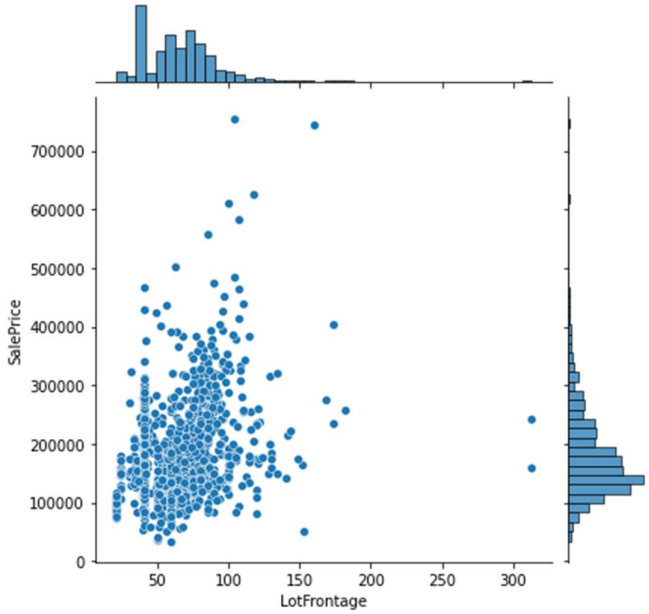


Figure : Joint Plot of LotFrontage and SalePrice

OBSERVATION:

- This Joint Plot shows the data is highly spread.
- LotFrontage is concentrated between 20 and 100

PLOT-7

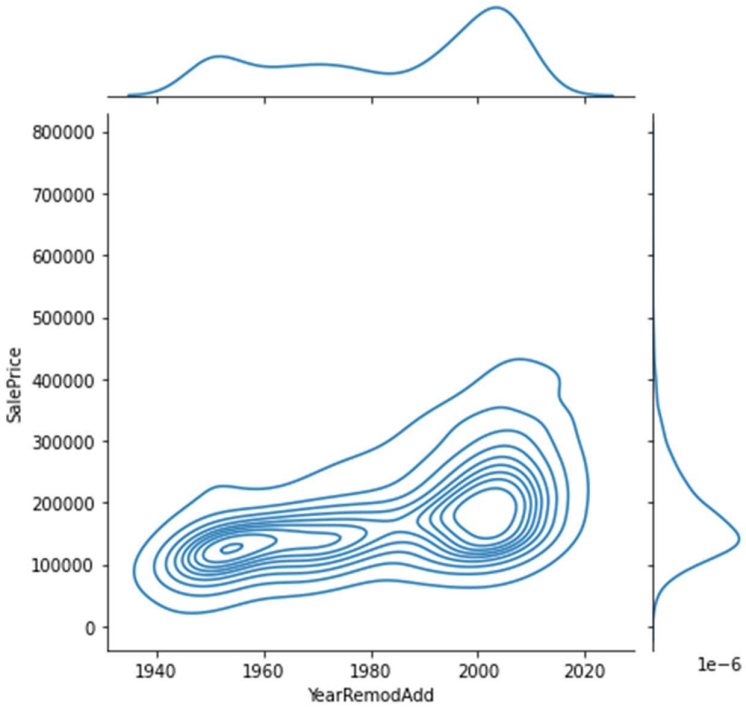


Figure : Joint Plot between YearRemodAdd and SalePrice

OBSERVATIONS:

- Data between YearRemodAdd and SalePrice is highly spread
- SalePrice is concentrated between 50k and 300k.

PLOT-8

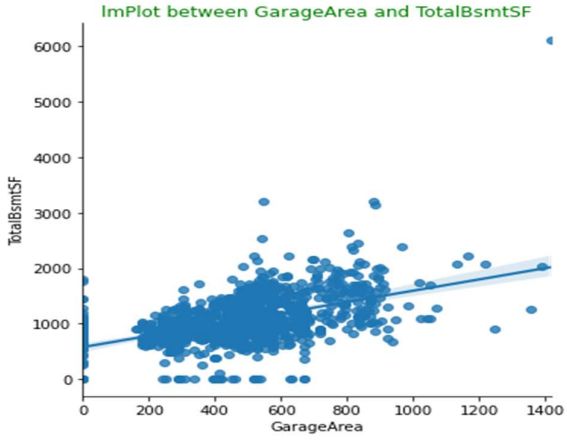


Figure : lmPlot Between GarageArea and TotalBsmtSF

OBSERVATIONS:

- High density of data between 200 and 900 Garage area.
- We have an outlier at 6000 TotalBsmtSF mark.
- Few of the data have zero TotalBsmtSF

PLOT-9

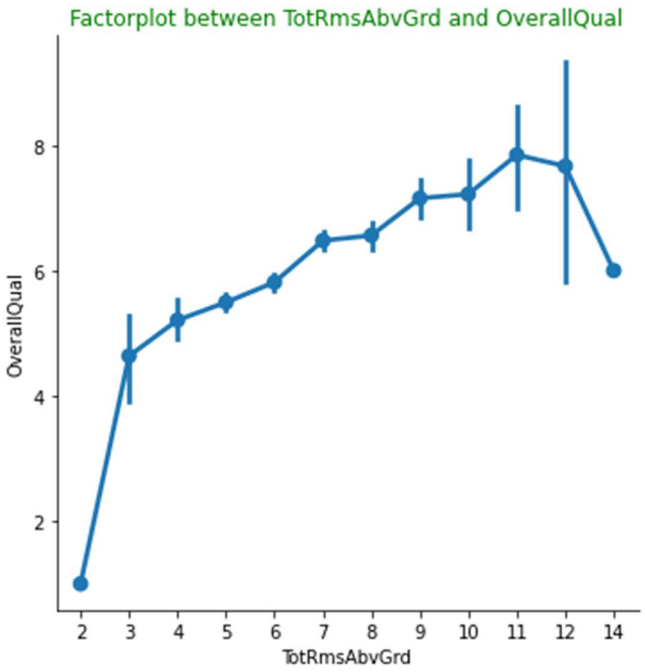


Figure : FactorPlot Between TotRmsAbvGrd and OverallQual

OBSERVATION

- There is steep rise in OverallQual between 2-3 TotRmsAbvGrd.
- TotRmsAbvGrd rises till 11 after that there is decline

PLOT-10

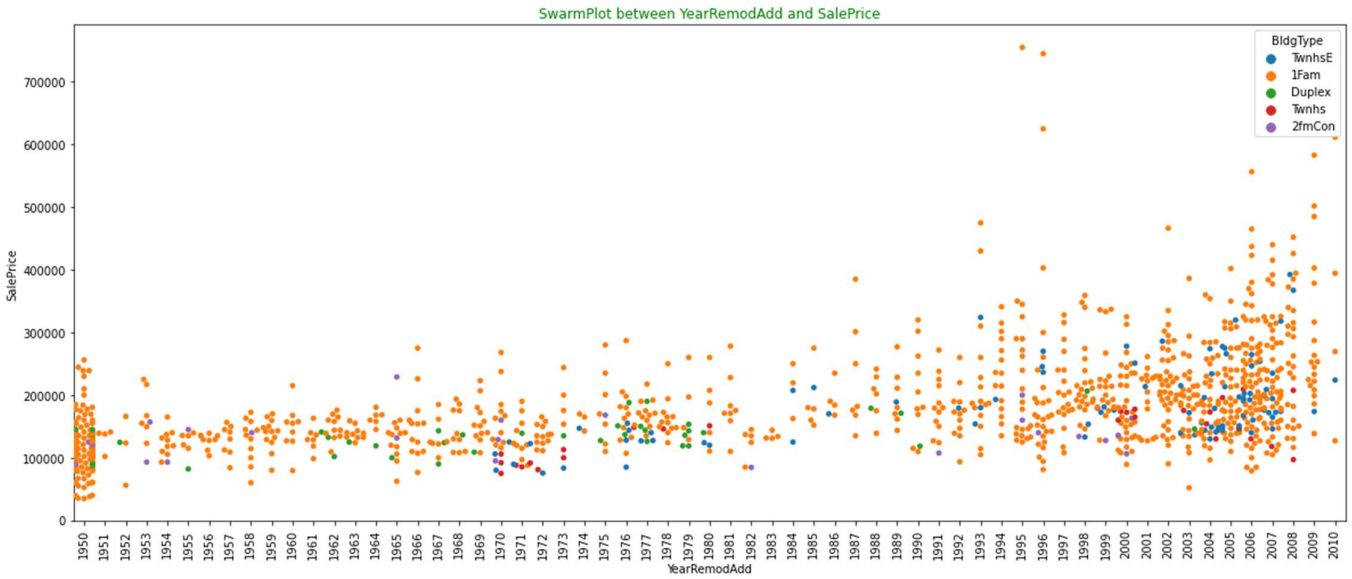


Figure : SwarmPlot between YearRemodAdd and SalePrice

OBSERVATION

- YearRemodAdd of 1950 has low SalePrice.
- YearRemodAdd between 2000 and 2008 has high data density.
- Twnhs BldgType has high YearRemodAdd between 1970 and 1973
- Prior to 1993 SalePrice is low.
- Number of High SalePrice is quite low.

LABEL ENCODING OF DATASET

Label Encoding is performed as the data have different datatype, this encoding will later be used in Machine Learning as shown below.

```
from sklearn.preprocessing import LabelEncoder
lenc=LabelEncoder()
for i in df.columns:
    df[i]=lenc.fit_transform(df[i])
df.head()
```

Output

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	Bldg
0	11	3	11	80	1	0	3	0	4	0	13	2	2	
1	0	3	65	808	1	0	3	0	4	1	12	2	2	
2	5	3	62	449	1	0	3	0	1	0	15	2	2	
3	0	3	75	632	1	0	3	0	4	0	14	2	2	
4	0	3	11	821	1	0	3	0	2	0	14	2	2	

TRAIN TEST SPLIT

We split the dataset into training and testing (x and y respectively) so that it can be used in Machine Learning for prediction of the Target variable as shown below. The value of x train, x test, y train, y test is shown below.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
x=df.drop(['SalePrice'],axis=1)
print(x.shape)
y1=df['SalePrice'].to_numpy()
y=pd.DataFrame(y1)
```

Output

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(934, 75)
(234, 75)
(934, 1)
(234, 1)
```

MACHINE LEARNING MODEL:

We have seen that the target variable (SalePrice) is s continuous value. Hence to predict the continuous value we use Regression Model. The model used is as shown below.

LINEAR REGRESSION:

We import the library for Linear Regression and r2 score to determine the accuracy of the model.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

Here we determine the random state where we get the highest accuracy score.

```
lr=LinearRegression()
j=[]
r2=[]
for i in range(0,100):
    j.append(i)
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=i)
    lr.fit(x_train,y_train)
    pred_train=lr.predict(x_train)
    pred_test=lr.predict(x_test)
    r2.append(r2_score(y_test,pred_test)*100)
    print(f'At random state {i},the training accuracy is :- {r2_score(y_train,pred_train)}')
    print(f'At random state {i}, the testing accuarcy is :- {r2[i]}')
```

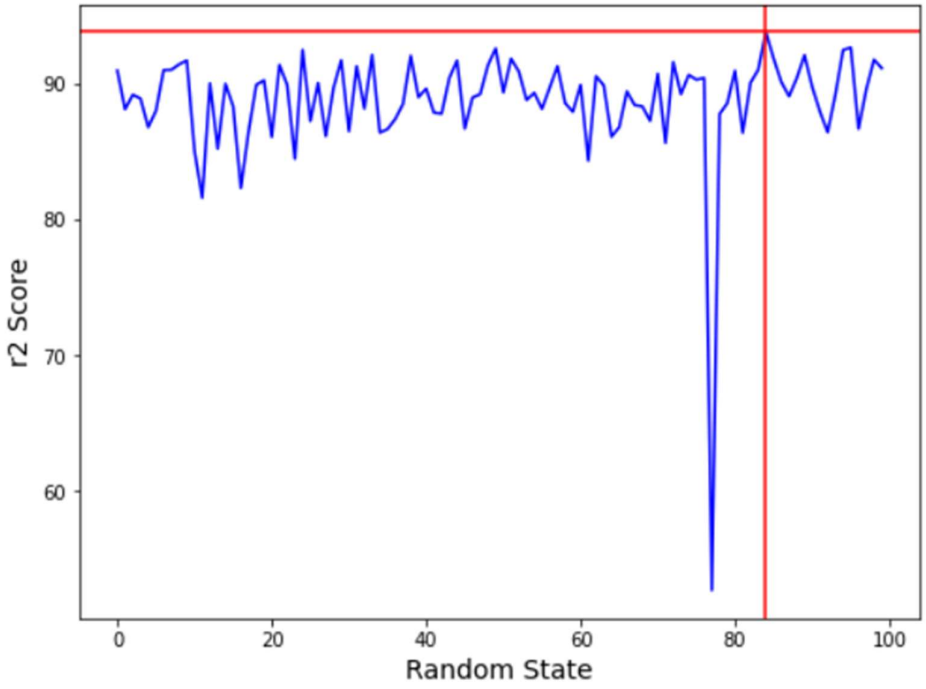
The below suggest that the random state of 84 gives the highest accuracy score of 93.806%, also shown in the graph below.

```
print(max(r2))
print(min(r2))
```

```
93.80622335673941
52.64727314164461
```

Below Graph showing r2 score with Random State

```
plt.figure(figsize=(8,6))
plt.plot(j,r2,color='b')
plt.xlabel('Random State',fontsize=14)
plt.ylabel('r2 Score',fontsize=14)
plt.axhline(y=max(r2),color='r')
plt.axvline(x=84,color='r')
plt.show()
```



We split the data into Train and Test using after Standard Scaling of x an y

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
sc_y = StandardScaler()
x = sc_x.fit_transform(x)
y = sc_y.fit_transform(y)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=84)
print('Shape of x_train',x_train.shape)
print('Shape of x_test',x_test.shape)
print('Shape of y_train',y_train.shape)
print('Shape of y_test',y_test.shape)
```

```
Shape of x_train (934, 75)
Shape of x_test (234, 75)
Shape of y_train (934, 1)
Shape of y_test (234, 1)
```

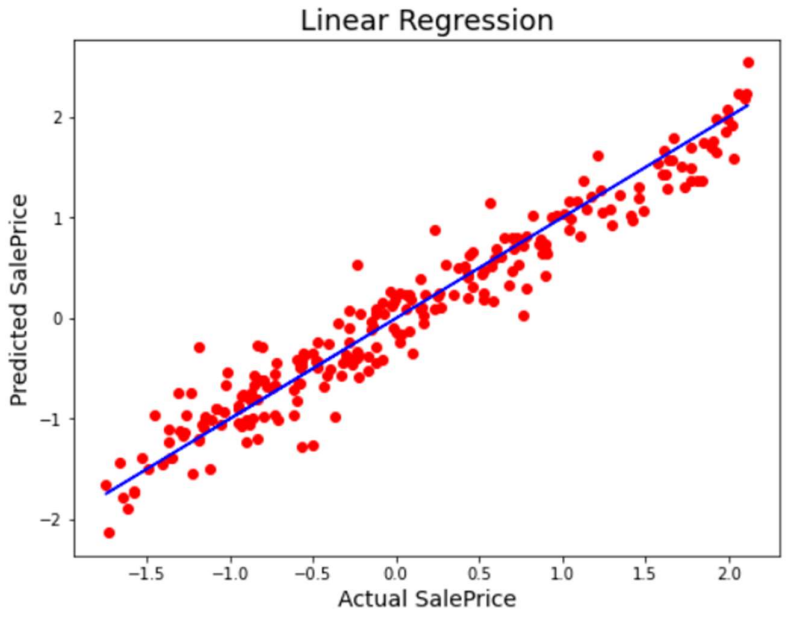
We calculate the Accuracy of the Model which comes to 93.80%

```
lr.fit(x_train,y_train)
pred_train=lr.predict(x_train)
pred_test=lr.predict(x_test)
print(f'The training accuracy is :- {r2_score(y_train,pred_train)}')
print(f'The testing accuarcy is :- {r2_score(y_test,pred_test)}')
```

```
The training accuracy is :- 0.9034057334124652
The testing accuarcy is :- 0.938068515822576
```

This Graph shows the Predicted SalePrice and Actual SalePrice

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



SUPPORT VECTOR REGRESSION:

In this model we first use scaling for x and y variables

```
# Feature Scaling
sc_x = StandardScaler()
sc_y = StandardScaler()
x = sc_x.fit_transform(x)
y = sc_y.fit_transform(y)
print(x.shape)
print(y.shape)
```

```
(1168, 75)
(1168, 1)
```

Finding the best kernel for the model.

```
from sklearn.svm import SVR
kernellist=['linear','poly','rbf']
for i in kernellist:
    sv=SVR(kernel=i)
    print(sv.fit(x,y))
    print(sv.score(x,y))
```

```
SVR(kernel='linear')
0.9082407051339781
SVR(kernel='poly')
0.9705504274363723
SVR()
0.9795444485986843
```

Using rbf as kernel for prediction of x

```
svr = SVR(kernel = 'rbf')
svr.fit(x,y)
y_pred = svr.predict(x)
y_pred
```

```
array([-0.83797212,  1.27623549,  1.19006618, ..., -0.46356383,
       -1.62102982,  0.19491034])
```

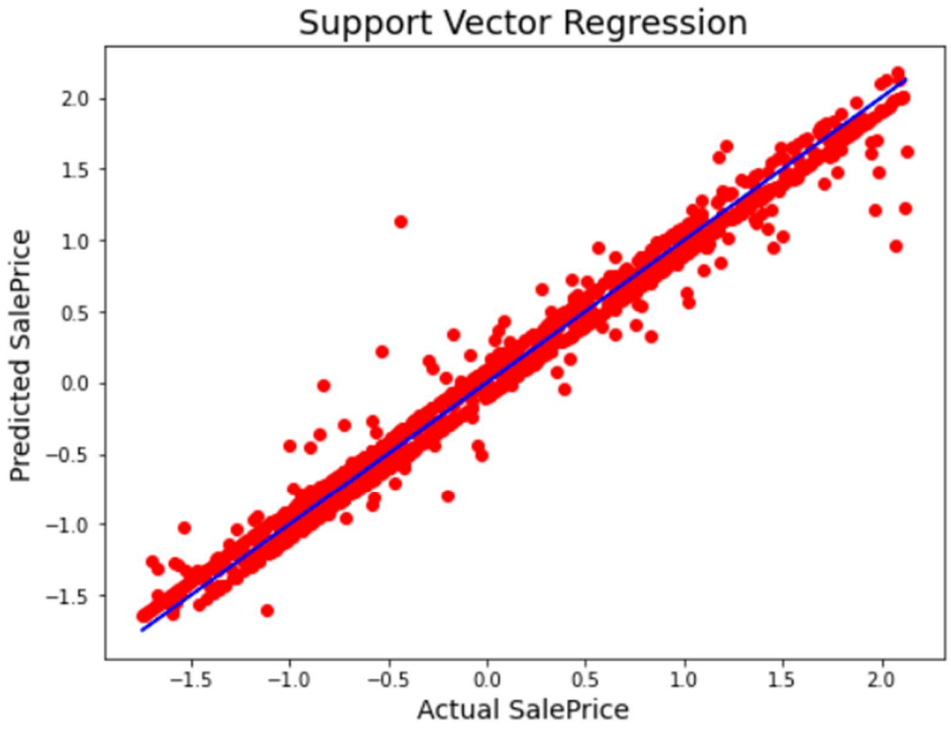
Accuracy score of the model

```
print(svr.fit(x,y))
print(svr.score(x,y))
```

```
SVR()
0.9795444485986843
```

Graphs showing Predicted SalePrice and Actual SalePrice

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
plt.scatter(x=y,y=y_pred,color='r')
plt.plot(y,y,color='b')
plt.xlabel('Actual SalePrice',fontsize=14)
plt.ylabel('Predicted SalePrice',fontsize=14)
plt.title('Support Vector Regression',fontsize=18)
plt.show()
```



DECISION TREE REGRESSION:

Splitting the Data for Testing ang Training

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print('Shape of x_train',x_train.shape)
print('Shape of x_test',x_test.shape)
print('Shape of y_train',y_train.shape)
print('Shape of y_test',y_test.shape)
```

```
Shape of x_train (934, 75)
Shape of x_test (234, 75)
Shape of y_train (934, 1)
Shape of y_test (234, 1)
```

Calculating the Predicted SalePrice

```
from sklearn.tree import DecisionTreeRegressor
dtc=DecisionTreeRegressor()
dtc.fit(x_train,y_train)
y_pred=dtc.predict(x)
print("Predicted SalePrice:",y_pred)
```

```
Predicted SalePrice: [-0.88656349  1.37648045  1.38983174 ... -0.41259264 -1.72101922
  0.29502582]
```

Accuracy of the Model

```
print(dtc.fit(x_train,y_train))
print('Accuracy Score is',dtc.score(x_train,y_train)*100,'%')
```

```
DecisionTreeRegressor()
Accuracy Score is 100.0 %
```

```
plt.figure(figsize=(8,6))
plt.scatter(x=y,y=y_pred,color='r')
plt.plot(y,y,color='b')
plt.xlabel('Actual Charges',fontsize=14)
plt.ylabel('Predicted Charges',fontsize=14)
plt.title('Decision Tree Regression',fontsize=18)
plt.show()
```

RANDOM FOREST REGRESSION:

Importing libraries for Random Forest regressor, Grid searchCV, Randomized Search CV.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from pprint import pprint
```

Deriving feature and label for the data

```
x=df.drop(['SalePrice'],axis=1)
print(x.shape)
y1=df['SalePrice'].to_numpy()
y=pd.DataFrame(y1)
print(y.shape)
print(y1.shape)
```

```
(1168, 75)
(1168, 1)
(1168,)
```

Spliting data into training and testing

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(934, 75)
(234, 75)
(934, 1)
(234, 1)
```

Parameter for Model

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(100, 200, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

Creating Random Grid for the Model

```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

Fit of Random Search Model

```
rfc=RandomForestRegressor(random_state=42)
rfr_random = RandomizedSearchCV(estimator = rfr, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
                                random_state=42, n_jobs = -1)
# Fit the random search model
rfr_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(random_state=42),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [100, 110, 120, 130, 140,
                                                      150, 160, 170, 180, 190,
                                                      200, None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

Best Parameter for the Model

```
rfr_random.best_params_
```

```
{'n_estimators': 400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': None,
 'bootstrap': False}
```

Creating The Parameter Grid Based on the Result of Random Search

```
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [False],
    'max_depth': [200,220,250,None],
    'max_features': ['sqrt'],
    'min_samples_leaf': [0.5,1,2],
    'min_samples_split': [2,4,6],
    'n_estimators': [300,400,500]
}
# Create a based model
rfr = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rfr, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
# Fit the grid search to the data
grid_search.fit(x_train,y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
```

```
{'bootstrap': False,
 'max_depth': 200,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 300}
```

Accuracy Score of the Model is 99.99%

```
: rfr2=RandomForestRegressor(bootstrap=False,
      max_depth=200,
      max_features='sqrt',
      min_samples_leaf=1,
      min_samples_split=2,
      n_estimators=300)
  print(rfr2.fit(x_train,y_train))
  print('Accuracy Score of Random Forest',rfr2.score(x_train,y_train)*100,'%')
```

```
RandomForestRegressor(bootstrap=False, max_depth=200, max_features='sqrt',
                      n_estimators=300)
Accuracy Score of Random Forest 99.99999938123389 %
```

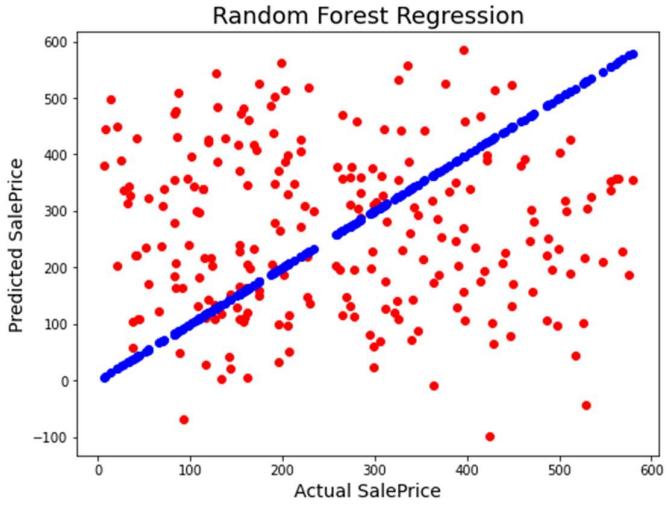
Calculating the Predicted SalePrice

```
rfr2.fit(x_train,y_train)
y_pred=dtc.predict(x_test)
print("Predicted SalePrice:",y_pred)
```

```
Predicted SalePrice: [2.07742327 2.07742327 2.05739633 1.93055906 2.05739633 2.07742327
 2.05739633 1.93055906 1.93055906 2.05739633 2.09077456 1.93055906
 1.93055906 2.07742327 1.93055906 2.05739633 1.30304835 2.07742327
 1.93055906 2.07742327 1.93055906 2.07742327 1.93055906 1.93055906
 2.07742327 2.05739633 1.93055906 1.93055906 2.07742327 2.07742327
 1.93055906 2.07742327 1.93055906 1.30304835 2.05739633 1.93055906
 1.93055906 1.93055906 1.93055906 1.93055906 1.30304835 1.93055906
 1.93055906 2.09077456 2.07742327 1.30304835 2.05739633 1.93055906
 2.07742327 1.93055906 1.93055906 1.30304835 2.05739633 1.93055906
 1.30304835 2.07742327 1.93055906 1.93055906 2.07742327 1.93055906
 2.07742327 2.07742327 1.93055906 1.30304835 1.93055906 2.05739633
 1.93055906 1.93055906 1.93055906 2.07742327 2.05739633 1.93055906
 2.07742327 2.07742327 1.93055906 1.93055906 1.93055906 2.07742327
 2.07742327 2.07742327 2.06407197 2.07742327 2.05739633 1.93055906
 2.07742327 1.5433716  2.07742327 1.93055906 1.93055906 2.07742327
 2.07742327 2.07742327 2.07742327 1.93055906 2.07742327 1.93055906
 2.07742327 2.07742327 2.07742327 1.93055906 1.30304835 1.93055906
 1.93055906 1.93055906 1.93055906 2.07742327 1.30304835 2.07742327
 1.93055906 1.93055906 2.07742327 1.93055906 2.07742327 2.07742327
 1.5433716  2.07742327 2.09077456 1.93055906 2.07742327 2.07742327
 1.93055906 2.05739633 1.93055906 2.07742327 2.07742327 1.93055906
 2.05739633 2.07742327 1.93055906 2.05739633 1.30304835 2.05739633
 1.30304835 2.07742327 2.05739633 2.07742327 1.93055906 1.93055906
 2.07742327 1.93055906 1.93055906 1.93055906 2.07742327 2.07742327
 2.05739633 2.05739633 2.05739633 2.07742327 1.93055906 1.93055906
 2.07742327 2.05739633 1.93055906 2.05739633 2.07742327 1.93055906
 1.93055906 2.05739633 2.07742327 1.93055906 1.93055906 2.05739633
 1.93055906 1.93055906 2.07742327 1.93055906 1.93055906 2.07742327
 1.30304835 1.30304835 2.05739633 2.07742327 1.93055906 2.07742327
 1.93055906 2.07742327 2.07742327 1.93055906 2.07742327 1.93055906
 2.05739633 1.93055906 1.93055906 2.05739633 2.07742327 1.93055906
 2.07742327 2.07742327 2.07742327 2.07742327 2.07742327 2.07742327
 2.05739633 2.07742327 1.93055906 2.07742327 1.93055906 1.93055906
 2.07742327 1.93055906 2.07742327 2.07742327 2.07742327 2.05739633
 1.93055906 2.07742327 1.93055906 2.05739633 1.93055906 1.93055906
 1.93055906 2.07742327 2.07742327 1.93055906 1.93055906 2.07742327
 2.05739633 1.93055906 2.07742327 2.07742327 2.05739633 1.30304835
 2.07742327 2.07742327 2.07742327 2.07742327 2.05739633 2.07742327
 2.07742327 2.07742327 1.93055906 1.93055906 2.07742327 2.07742327]
```

Graph of Predicted and Actual SalePrice

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test,y=pred_test,color='r')
plt.scatter(y_test,y_test,color='b')
plt.xlabel('Actual SalePrice',fontsize=14)
plt.ylabel('Predicted SalePrice',fontsize=14)
plt.title('Random Forest Regression',fontsize=18)
plt.show()
```



4. CONCLUSION

After analysing the result we found the following result.

- The Linear Regression Model gives us an accuracy of **93.80%**
- The Support Vector Regression after best fit gives us an accuracy of **97.95%.**
- The Decision Tree Regression Model gives an Overfitting result as every iteration of parameters provides us only **100%** accuracy.
- Random Forest Regression Model gives us varying accuracy as per the parameters and Grid Search CV. After applying best fit parameter for prediction, it gives us the accuracy of **99.99%.**

Hence **Random Forest Regression Model** is used for prediction of test dataset**.**

5. SAVING MODELS

We are saving the model in pickle file and load the test dataset as shown below.

```
import pickle
filename= 'picklerffile.pkl'
pickle.dump(rfr2, open(filename, 'wb'))
# Load the model from disk
loaded_model=pickle.load(open('picklerffile.pkl','rb'))
df_test=pd.read_csv('C:\\Users\\Pankaj\\Data Science\\FlipRobo Projects\\Housing Project\\test.csv')
df_test1=pd.DataFrame(df_test)
df_test1
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Co
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	Corner	Gtl	StoneBr	Norm	
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	StoneBr	Norm	
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl	Crawfor	Norm	
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	Somerst	Feedr	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
287	83	20	RL	78.0	10206	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Somerst	Norm	
288	1048	20	RL	57.0	9245	Pave	NaN	IR2	Lvl	AllPub	Inside	Gtl	CollgCr	Norm	
289	17	20	RL	NaN	11241	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	NAmes	Norm	
290	523	50	RM	50.0	5000	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl	BrkSide	Feedr	
291	1379	160	RM	21.0	1953	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	BrDale	Norm	

292 rows × 80 columns

We predict the test dataset as below.

```
predict_test=loaded_model.predict(df_test1)
print(predict_test)
```

```
[309.52333333 183.91       243.74       114.03       238.30333333
  44.34       133.07       262.05333333 205.87       146.61333333
  26.18333333  92.21333333  83.98       151.48333333 274.22333333
  80.23333333  70.68        82.62333333 160.68       189.74333333
 121.21       115.11       115.24666667  57.52        42.24333333
  73.86666667 167.03333333  94.58333333 154.15666667  52.51
 104.65666667 189.56333333 216.23       130.75        56.05333333
 167.54       186.64666667  75.18333333 116.85666667 120.68
  47.37       267.48       206.03       185.         113.52666667
  87.64        67.09666667  63.55666667 213.23       322.40666667
 108.22333333 171.76333333  41.12333333  25.73       236.57333333
  96.93333333 131.35666667 194.63666667  73.62666667 252.79666667
  59.10666667 173.99333333  72.83       134.99       195.23
  52.14666667 139.89       208.52333333 117.57666667 123.14
 255.27       158.92666667 132.76       117.44333333 113.72666667
 227.41       277.23       190.88       265.73666667 107.94
 209.52666667  92.15666667 109.01       122.69333333 187.09333333
 191.93        80.31333333 274.21       109.52666667 170.83
 230.29333333  94.86666667  85.97666667  82.37666667 197.85
 139.61333333 233.52666667 175.48333333 287.84        88.21333333
 245.05666667  67.91        72.14666667 168.44666667 177.41333333
 102.97333333 220.98       116.73333333 181.59666667 186.04666667
 200.45666667 148.06       145.91333333 244.47333333  76.99666667
  51.8         94.67333333 177.33       122.32333333  64.42333333
  63.26333333 194.98       205.28333333 100.91666667  82.59333333
 167.37666667  68.63666667 134.65333333  50.45        80.82
 106.51       212.61       116.71       143.60333333 126.59
 267.54666667 203.27        82.47666667 276.61        81.39333333
  96.29333333 298.44666667  53.54       321.51666667 122.88333333
 223.70333333 120.69666667  81.13666667  51.35       204.2
 174.83666667 101.49       195.61333333  74.65        53.46
 146.73333333 163.88666667 154.68666667  85.36       161.04
 204.23       108.92333333 182.33333333  72.28        78.01666667
 228.82       165.04       198.14333333  76.89666667 213.17666667
 139.98666667  86.97333333 110.61666667 269.72666667  95.61
 318.86333333  75.71        37.81       140.31       117.37666667
 211.79       130.19666667 254.03333333 162.33666667 343.56333333
 307.95       186.47666667  47.51333333 136.46666667 101.24
  64.21333333 181.37       204.19666667  54.25666667  90.21666667
  22.53666667 152.76       193.98333333  55.86333333 135.65666667
 147.75333333  65.73666667 248.65333333 247.41666667  71.15666667
  68.27333333 256.74333333  95.61333333 101.40333333 152.47333333
  60.25       114.65        86.51       145.76333333  52.64
 108.69333333 138.87666667  93.95        74.85666667 139.15333333
 208.25333333 200.25       275.05        91.1        195.12666667
 302.31333333 208.29        63.23333333 321.79333333 181.33666667
  46.89       141.23666667  40.56666667  82.69        72.87333333
 197.63       162.22       147.61       206.42       193.24333333
 199.22333333 102.42666667 234.60666667  37.82666667 110.06
 205.76666667 164.78666667 246.75666667 101.40333333 127.39666667
 134.46       238.78666667 101.54666667  60.81333333  66.49333333
 186.11333333  91.61333333 329.10666667 148.57333333  48.72333333
 187.02666667 166.10666667  54.3        113.36       184.32666667
 161.69333333  63.16       140.99333333 234.09333333 238.31
 117.35        83.41333333 337.50666667 195.04333333 268.79333333
 189.17333333 144.33666667 123.73333333 187.75666667 333.03
  88.48666667 323.21333333 103.65333333 268.68333333 124.56
  89.29666667 142.83333333 221.85       110.02333333 123.07333333
 120.83666667  54.31666667]
```

------ THE END ------