NAME OF THE PROJECT

Housing Price Prediction

Submitted by:

Arshi Maitra

Acknowledgement:

This project has been completed with the help of training documents and live classes recordings from Data Trained Education. Few helps on coding have also been taken from few data science websites like Toward Data Science, Geek for Geeks, Stack Overflow.

INTRODUCTION

House price prediction can help the developer determine the selling price of a house and can help the customer to arrange the right time to purchase a house. There are three factors that influence the price of a house which include physical conditions, concept and location. Traditional house price prediction is based on cost and sale price comparison lacking of an accepted standard and a certification process. Therefore, the availability of a house price prediction model helps fill up an important information gap and improve the efficiency of the real estate market. 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.

Below is the dataset available from the sale of houses in Australia as 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.

The main goal of the project will be to determine the following:

- To determine the price of houses in Australia
- To understand the factors that are responsible for the price of a house

Analytical Problem Framing

Importing of necessary libraries:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest, f_classif, f_regression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from prettytable import PrettyTable
import warnings
warnings.filterwarnings('ignore') #Importing the necessary libraries
```

Data preprocessing/Data Cleaning:

Both the train and test set has been loaded

```
In [2]: Train_Data=pd.read_csv("HousingTrain.csv") #Checking the training dataset

In [3]: Test_Data=pd.read_csv("Housing Test.csv") #Checking the test dataset

In [4]: Train_Data.head()
```

Out[4]:

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

5 rows × 81 columns

Going through the columns of both the data:

```
In [5]: Train_Data.columns #Checking the total columns

Out[5]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
               'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
               'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
               'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
               'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
               'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
               'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
               'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
               'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
               'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
               'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
               'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
               'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
               'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
               'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
               'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
               'SaleCondition', 'SalePrice'],
              dtype='object')
```

```
In [6]: Test_Data.columns

Out[6]: Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
               'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
               'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
               'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
               'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
               'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
               'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
               'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
               'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
               'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
               'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
               'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
               'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
               'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
               'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
               'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
               'SaleCondition'],
              dtype='object')
```

Checking the null values from Train and Test data:

```
In [8]: Train_Data.isnull().sum() #Checking the null values of the train data

Out[8]: Id                 0
        MSSubClass         0
        MSZoning           0
        LotFrontage      214
        LotArea            0
        Street             0
        Alley           1091
        LotShape           0
        LandContour        0
        Utilities          0
        LotConfig          0
        LandSlope          0
        Neighborhood       0
        Condition1         0
        Condition2         0
        BldgType           0
        HouseStyle         0
        OverallQual        0
```

```
LowQualFinSF        0
GrLivArea           0
BsmtFullBath        0
BsmtHalfBath        0
FullBath            0
HalfBath            0
BedroomAbvGr        0
KitchenAbvGr        0
KitchenQual         0
TotRmsAbvGrd        0
Functional          0
Fireplaces          0
FireplaceQu       551
GarageType         64
GarageYrBlt        64
GarageFinish       64
GarageCars          0
GarageArea          0
GarageQual         64
GarageCond         64
PavedDrive          0
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
PoolQC           1161
Fence             931
```

```
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         7
MasVnrArea         7
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          30
BsmtCond          30
BsmtExposure      31
BsmtFinType1      30
BsmtFinSF1         0
BsmtFinType2      31
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         0
1stFlrSF           0
2ndFlrSF           0
```

```
MiscFeature      1124
MiscVal             0
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice           0
dtype: int64
```

Dropping the null values as well as filling up the null values:

```
In [9]: # For the train data- before imputing the null values, it can be seen that for few columns there are more 90% of null values.
        # Hence dropping those columns

In [10]: Train_Data.drop(['Alley','PoolQC','Fence','MiscFeature'],axis=1,inplace=True)
```

Data Imputation:

```
In [13]: Train_Data['LotFrontage']=Train_Data['LotFrontage'].fillna(Train_Data['LotFrontage'].mean())
         Train_Data['BsmtQual']=Train_Data['BsmtQual'].fillna(Train_Data['BsmtQual'].mode()[0])
         Train_Data['BsmtCond']=Train_Data['BsmtQual'].fillna(Train_Data['BsmtCond'].mode()[0])
         Train_Data['BsmtExposure']=Train_Data['BsmtExposure'].fillna(Train_Data['BsmtExposure'].mode()[0])
         Train_Data['BsmtFinType1']=Train_Data['BsmtFinType1'].fillna(Train_Data['BsmtFinType1'].mode()[0])
         Train_Data['BsmtFinType2']=Train_Data['BsmtFinType2'].fillna(Train_Data['BsmtFinType2'].mode()[0])
         Train_Data['FireplaceQu']=Train_Data['FireplaceQu'].fillna(Train_Data['FireplaceQu'].mode()[0])
         Train_Data['GarageType']=Train_Data['GarageType'].fillna(Train_Data['GarageType'].mode()[0])
         Train_Data['GarageYrBlt']=Train_Data['GarageYrBlt'].fillna(Train_Data['GarageYrBlt'].mean())
         Train_Data['GarageFinish']=Train_Data['GarageFinish'].fillna(Train_Data['GarageFinish'].mode()[0])
         Train_Data['GarageCars']=Train_Data['GarageCars'].fillna(Train_Data['GarageCars'].mean())
         Train_Data['GarageArea']=Train_Data['GarageArea'].fillna(Train_Data['GarageArea'].mean())
         Train_Data['GarageQual']=Train_Data['GarageQual'].fillna(Train_Data['GarageQual'].mode()[0])
         Train_Data['GarageCond']=Train_Data['GarageCond'].fillna(Train_Data['GarageCond'].mode()[0])
         Train_Data['MasVnrType']=Train_Data['MasVnrType'].fillna(Train_Data['MasVnrType'].mode()[0])
         Train_Data['MasVnrArea']=Train_Data['MasVnrArea'].fillna(Train_Data['MasVnrArea'].mean())
```

Encoding the object data types:

```
In [22]: from sklearn.preprocessing import LabelEncoder

In [23]: enc= LabelEncoder()

In [24]: columns=['MSZoning','Street','LotShape','LandContour','Utilities','LotConfig','LandSlope','Neighborhood','Condition1','Condition2
         X[columns] = X[columns].apply(enc.fit_transform)
```

Selecting K Best feature selection method:

As we had 81 columns hence it was difficult to train the model with all the 80 features. Therefore K Best feature selection method has been utilised in order to get the best 25 features in comparison to 80.

```
In [26]: Features= SelectKBest(score_func=f_regression, k=25)
         fit=Features.fit(X,Y)
         scores=pd.DataFrame(fit.scores_)
         columns=pd.DataFrame(X.columns)
```

Checking the scores:

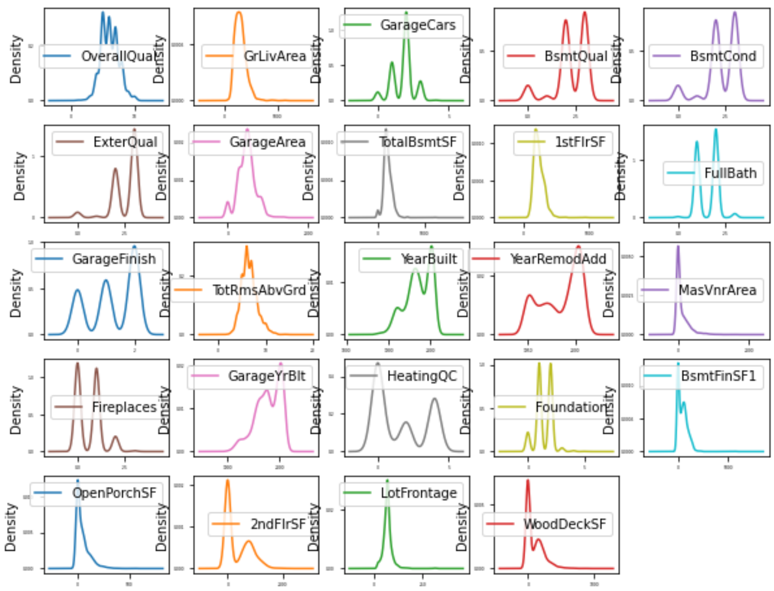
```
In [27]: Total_Score=pd.concat([columns,scores],axis=1)
         Total_Score.columns=['Column','Score']
         print(Total_Score.nlargest(25,'Score'))
```

```
          Column        Score
16   OverallQual  1925.310146
45     GrLivArea  1167.278131
60    GarageCars   760.625799
29      BsmtQual   754.737009
30      BsmtCond   754.737009
26     ExterQual   746.729944
61    GarageArea   724.282299
37   TotalBsmtSF   639.162482
52   KitchenQual   630.659595
42      1stFlrSF   615.032166
48      FullBath   518.998338
59  GarageFinish   472.788235
53  TotRmsAbvGrd   451.573798
18     YearBuilt   419.564317
19  YearRemodAdd   405.199690
25    MasVnrArea   319.254378
55    Fireplaces   312.273172
58   GarageYrBlt   309.521298
39     HeatingQC   230.953343
28    Foundation   189.817339
33    BsmtFinSF1   176.819524
66   OpenPorchSF   151.901231
43      2ndFlrSF   142.869824
3    LotFrontage   136.550260
65    WoodDeckSF   128.843420
```

Therefore now received the best 25 features for the target variable.

Checking Data Skewness :

```
In [30]: NewX.plot(kind='kde',subplots=True,layout=(5,5),sharex=False,legend=True,fontsize=3,figsize=(10,8))
         plt.show() # ploting the data and observing high skewness
```



Skewness Score:

```
In [31]: NewX.skew().sort_values(ascending=False) #checking the skewness

Out[31]: MasVnrArea       2.834658
         LotFrontage      2.710383
         OpenPorchSF      2.410840
         BsmtFinSF1       1.871606
         TotalBsmtSF      1.744591
         1stFlrSF         1.513707
         WoodDeckSF       1.504929
         GrLivArea        1.449952
         2ndFlrSF         0.823479
         Fireplaces       0.671966
         TotRmsAbvGrd     0.644657
         HeatingQC        0.449933
         GarageArea       0.189665
         OverallQual      0.175082
         FullBath         0.057809
         Foundation      -0.002761
         GarageCars      -0.358556
         GarageFinish    -0.450190
         YearRemodAdd    -0.495864
         YearBuilt       -0.579204
         GarageYrBlt     -0.662934
         BsmtCond        -1.343781
         BsmtQual        -1.343781
         ExterQual       -1.810843
         dtype: float64
```

Importing Power Transformation.

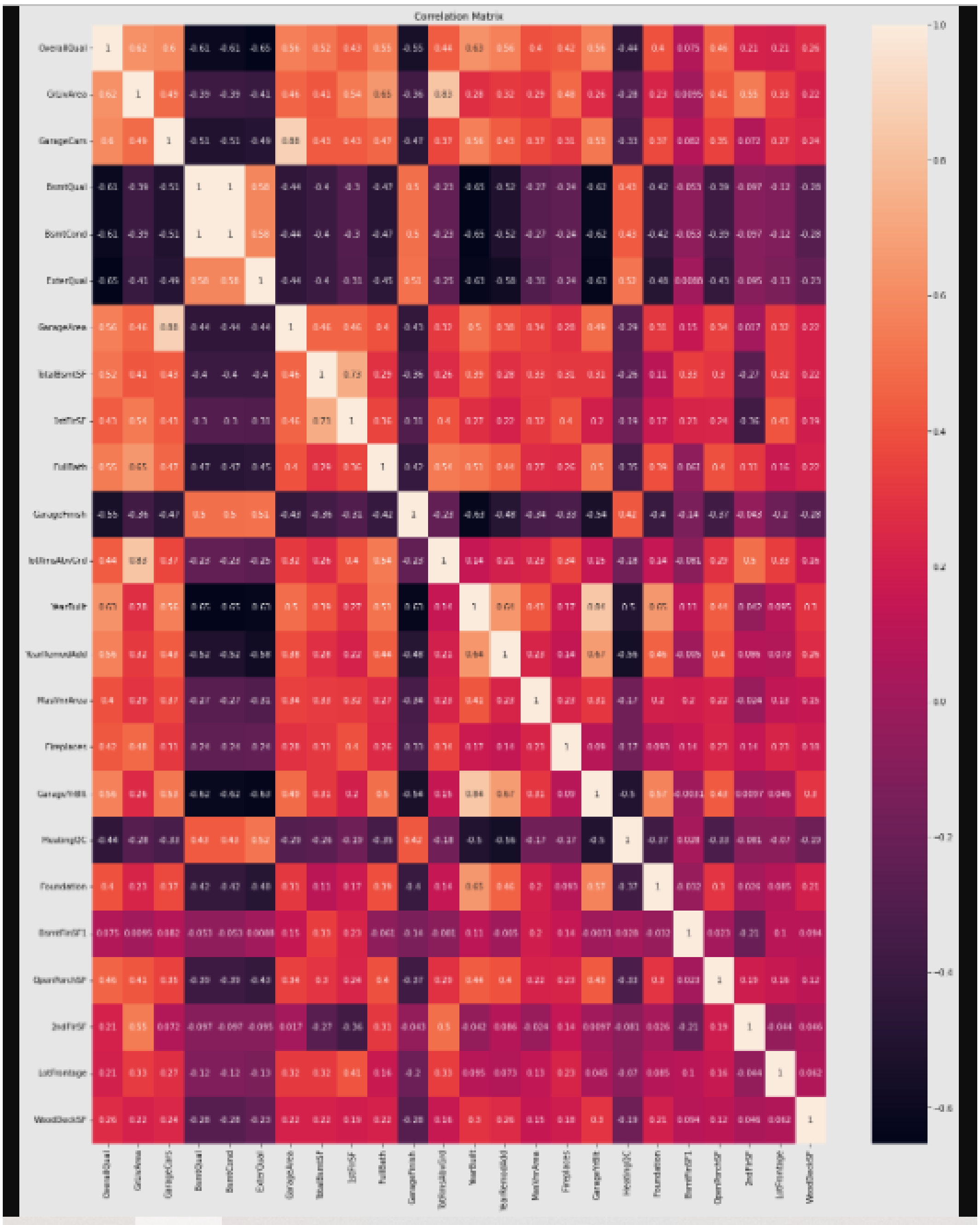
As the data is skewed it will affect the prediction score. Therefore data has to be transformed before taking any further action:

```
In [32]: from sklearn.preprocessing import power_transform

In [33]: New_X=power_transform(NewX)

In [34]: pd.DataFrame(New_X,columns=NewX.columns).skew().sort_values(ascending=False) # transforming the data to reduce skewness

Out[34]: MasVnrArea       0.416370
         TotalBsmtSF      0.286779
         2ndFlrSF         0.280208
         LotFrontage      0.161368
         HeatingQC        0.156511
         WoodDeckSF       0.113026
         Fireplaces       0.084950
         OverallQual      0.021658
         Foundation       0.004296
         TotRmsAbvGrd     0.002332
         GrLivArea       -0.000054
         1stFlrSF        -0.002391
         OpenPorchSF     -0.002749
         GarageCars      -0.022970
         FullBath        -0.045944
         YearBuilt       -0.126641
         GarageYrBlt     -0.132523
         YearRemodAdd    -0.225131
         GarageArea      -0.320370
         GarageFinish    -0.335248
         BsmtFinSF1      -0.404528
         BsmtCond        -0.413999
         BsmtQual        -0.413999

         ExterQual       -0.605112
         dtype: float64
```

Checking Multicollinearity:

i) Heat Map:

```
In [37]: corr_mat=X.corr()
         plt.figure(figsize=[20,25])
         sns.heatmap(corr_mat,annot=True)
         plt.title("Correlation Matrix")
         plt.show() #Checking correlation
```



From the heat map it can be observed that basement quality and basement condition has 100 % multicollinearity issue. Therefore it needs to be dropped, lets check via another metric that is VIF.

ii) VIF:

```
In [39]: vif_data = pd.DataFrame()
         vif_data["feature"] = X.columns
         vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                                   for i in range(len(X.columns))]

         vif_data
```

	feature	VIF
0	OverallQual	3.315234
1	GrLivArea	19.498638
2	GarageCars	5.351208
3	BsmtQual	inf
4	BsmtCond	inf
5	ExterQual	2.402698
6	GarageArea	4.981370
7	TotalBsmtSF	3.060318
8	1stFlrSF	12.212694
9	FullBath	2.358885
10	GarageFinish	1.946111
11	TotRmsAbvGrd	3.644552
12	YearBuilt	5.973074
13	YearRemodAdd	2.279980
14	MasVnrArea	1.385100
15	Fireplaces	1.529214
16	GarageYrBlt	4.344212
17	HeatingQC	1.652739
18	Foundation	2.007496
19	BsmtFinSF1	1.290198
20	OpenPorchSF	1.475894
21	2ndFlrSF	11.942552
22	LotFrontage	1.334319
23	WoodDeckSF	1.181661

As concluded from the heat map above, Basement Quality and basement condition has 100 % multicollinearity issue. Along with that first floor square feet, second floor squarefeet as well as Ground level above squarefeet also has multicollinearity issue.

It can be said be Ground level above squarefeet is the same data as first floor square feet, second floor squarefeet.

Therefore Basement Quality, first floor square feet and second floor squarefeet are dropped off in the train and test data.

```
In [40]: X.drop('BsmtQual',axis=1,inplace=True)
```

```
In [42]: X.drop(['1stFlrSF','2ndFlrSF'],axis=1,inplace=True)
```

Checking the final VIF Score:

```
In [48]: vif_data = pd.DataFrame()
         vif_data["feature"] = X.columns
         vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                                   for i in range(len(X.columns))]

         vif_data
```

0	OverallQual	3.261709
1	GrLivArea	5.866728
2	GarageCars	5.347618
3	BsmtCond	2.148281
4	ExterQual	2.401001
5	GarageArea	4.962258
6	TotalBsmtSF	1.890722
7	FullBath	2.341211
8	GarageFinish	1.941324
9	TotRmsAbvGrd	3.641823
10	YearBuilt	5.935050
11	YearRemodAdd	2.274966
12	MasVnrArea	1.372453
13	Fireplaces	1.498093
14	GarageYrBlt	4.324579
15	HeatingQC	1.651327
16	Foundation	1.944356
17	BsmtFinSF1	1.289871
18	OpenPorchSF	1.465648
19	LotFrontage	1.295308
20	WoodDeckSF	1.177207

Therefore, the issue of multicollinearity has now been resolved.

Detecting Outliers:

Zscore method has been applied to remove the outliers in the data.

```
In [51]: from scipy.stats import zscore

In [52]: (np.abs(zscore(X)<3)).all()

Out[52]: OverallQual      True
         GrLivArea       False
         GarageCars      False
         BsmtCond         True
         ExterQual        True
         GarageArea      False
         TotalBsmtSF     False
         FullBath         True
         GarageFinish     True
         TotRmsAbvGrd    False
         YearBuilt        True
         YearRemodAdd     True
         MasVnrArea       True
         Fireplaces       True
         GarageYrBlt      True
         HeatingQC        True
         Foundation      False
         BsmtFinSF1       True
         OpenPorchSF      True
         LotFrontage     False
         WoodDeckSF       True
         dtype: bool
```

Outliers has been removed and values without outliers have been stored in a different variable.

```
In [53]: # these are the index positions where outlier is present
         index = np.where(np.abs(zscore(X))>3)
         index

Out[53]: (array([  34,   48,   48,   48,   48,   52,   52,   54,   60,   86,   96,
                 119,  121,  124,  137,  141,  141,  159,  177,  191,  195,  195,
                 210,  211,  226,  231,  243,  249,  249,  249,  267,  305,  361,
                 361,  370,  420,  432,  483,  491,  498,  504,  510,  517,  537,
                 544,  558,  592,  592,  592,  592,  614,  644,  656,  691,  698,
                 706,  735,  747,  758,  760,  772,  800,  831,  834,  846,  846,
                 865,  884,  897,  899,  902,  908,  915,  935,  980,  980, 1025,
                1035, 1042, 1046, 1053, 1056, 1067, 1094, 1104, 1104, 1107, 1117,
                1120, 1126, 1144, 1147, 1148, 1164], dtype=int64),
          array([ 6,  0,  1,  6,  9,  0,  1,  6,  6,  6,  6,  7,  2,  6,  6,  1, 19,
                  6, 16, 19, 16, 19, 16,  7,  6, 19,  6,  0,  1,  7,  6,  6,  1,  6,
                  6, 19,  6,  7,  6, 16,  6,  0,  6, 19,  7, 19,  1,  5,  6, 19,  9,
                  7, 19,  1, 19, 19,  6, 19, 19, 19,  5, 19, 19,  6,  0,  7, 19,  6,
                  5,  6, 19,  6, 19,  6,  2,  5,  2,  6,  6, 19, 19,  6,  6,  5,  2,
                 16,  6,  6, 19,  6, 16, 19,  6,  6], dtype=int64))

In [54]: New_X = X[(np.abs(zscore(X))<3).all(axis=1)]
         New_X

In [55]: Y_new=Y.drop(index[0],axis=0)
         Y_new  #removing the outliers from target variables

Out[55]: 0       128000
         1       268000
         2       269790
         3       190000
         4       215000
                  ...
         1162     58500
         1163    122000
         1165    148500
         1166     40000
         1167    183200
         Name: SalePrice, Length: 1089, dtype: int64
```

Scaling the data:

```
In [60]: X_Scaled=Scalar.fit_transform(New_X)
         X_Scaled

Out[60]: array([[-0.09859076, -1.3363421 ,  0.25995446, ...,  1.38217617,
                  0.10106507, -0.97594068],
                [ 1.39641172,  1.36740337,  0.25995446, ...,  1.38669106,
                  1.27311012,  0.76580805],
                [ 0.65945083,  1.05642844,  0.25995446, ...,  1.17074992,
                  1.13760085,  1.02781403],
                ...,
                [-0.09859076,  0.0126929 ,  0.25995446, ..., -1.0932726 ,
                 -3.2731638 ,  0.78974467],
                [-1.69445354, -0.31064138, -1.12809175, ...,  0.76598274,
                 -1.14710557, -0.97594068],
                [-0.09859076,  0.12790642,  0.25995446, ...,  0.91672675,
                  0.10106507,  0.83612318]])
```

Model/s Development and Evaluation

(Linear Regression, Decision Tree Regressor, Random Forest Regressor and Gradient Boosting Regressor)

Linear Regression: 83.66%

Linear Regression

```
In [63]: LR=LinearRegression()
```

```
In [64]: X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y_new,test_size=0.20,random_state=50)
         LR.fit(X_train,y_train)
         pred_test=LR.predict(X_test)


         print('R-Squared:',r2_score(y_test,pred_test)*100)

         R-Squared: 83.66504953665496
```

Decision Tree Regressor: 66.61%

Decision Tree Regressor

```
In [65]: DT=DecisionTreeRegressor()
```

```
In [66]: X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y_new,test_size=0.20,random_state=50)
         DT.fit(X_train,y_train)
         pred_test=DT.predict(X_test)


         print('R-Squared:',r2_score(y_test,pred_test)*100)

         R-Squared: 66.61352107885887
```

Random Forest Regressor::85.52%

Random Forest Regressor

```
In [67]: rf=RandomForestRegressor()
```

```
In [68]: X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y_new,test_size=0.20,random_state=50)
         rf.fit(X_train,y_train)
         pred_test=rf.predict(X_test)


         print('R-Squared:',r2_score(y_test,pred_test)*100)

         R-Squared: 85.52395463859415
```

Gradient Boosting Regressor:86.66%

Gradient Boosting ¶

```
In [71]: GB=GradientBoostingRegressor()
```

```
In [74]: X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y_new,test_size=0.20,random_state=50)
         GB.fit(X_train,y_train)
         pred_test=GB.predict(X_test)


         print('R-Squared:',r2_score(y_test,pred_test)*100)

         R-Squared: 86.66187602268887
```

After getting the r squared scores for all the models, it needs to be checked whether any one of them are overfitted, hence cross validation technique has been used.

Cross Validation for LR:

Cross Validation for LR

```
In [76]: for i in range(2,6):
             LR_Val=cross_val_score(LR,X_Scaled,Y_new,cv=i)
             print("The cross validation score for Linear Regressor",i,"is",LR_Val.mean())
```

```
The cross validation score for Linear Regressor 2 is 0.8099555120612187
The cross validation score for Linear Regressor 3 is 0.8165233974276257
The cross validation score for Linear Regressor 4 is 0.8165572438206878
The cross validation score for Linear Regressor 5 is 0.8132481644785304
```

Cross Validation for DT:

Cross Validation for DT

```
In [77]: for i in range(2,6):
             DT_Val=cross_val_score(DT,X_Scaled,Y_new,cv=i)
             print("The cross validation score for Decision Tree Regressor",i,"is",DT_Val.mean())
```

```
The cross validation score for Decision Tree Regressor 2 is 0.7001601206866955
The cross validation score for Decision Tree Regressor 3 is 0.7354424014889114
The cross validation score for Decision Tree Regressor 4 is 0.7002785021394589
The cross validation score for Decision Tree Regressor 5 is 0.7238830226633229
```

Cross Validation for RF:

Cross Validation for RF

```
In [79]: for i in range(2,6):
             RF_Val=cross_val_score(rf,X_Scaled,Y_new,cv=i)
             print("The cross validation score for",i,"is",RF_Val.mean()*100)
```

```
The cross validation score for 2 is 83.45543872118004
The cross validation score for 3 is 85.87326337278839
The cross validation score for 4 is 86.04839891223044
The cross validation score for 5 is 85.63141513051768
```

Cross Validation for GB:

Cross Validation for GB

```
In [80]: for i in range(2,6):
             GB_Val=cross_val_score(GB,X_Scaled,Y_new,cv=i)
             print("The cross validation score for",i,"is",GB_Val.mean()*100)
```

```
The cross validation score for 2 is 85.7902333179118
The cross validation score for 3 is 87.15819308502503
The cross validation score for 4 is 86.93685769382368
The cross validation score for 5 is 86.57363515177636
```

As none of the models are overfitted, and based on the r squared score and cross validation scores, Gradient Boosting Regressor model is best for this dataset.

Hypertuning Parameter:

As gradient boosting has been termed as the best model for this dataset. Lets try to tune the parameters to see if the score can be increased.

1) ***Trying first with best parameter by Grid Search CV method:***

```
In [82]: gb=GradientBoostingRegressor()

In [84]: Parameters={'criterion':['mse', 'mae'],'min_samples_split':[3,4],'min_samples_leaf':[2,3]}
         clf=GridSearchCV(gb,Parameters)
         clf.fit(X_train,y_train)

Out[84]: GridSearchCV(estimator=GradientBoostingRegressor(),
                      param_grid={'criterion': ['mse', 'mae'],
                                  'min_samples_leaf': [2, 3],
                                  'min_samples_split': [3, 4]})

In [85]: clf.best_params_ #taking the best parameters

Out[85]: {'criterion': 'mse', 'min_samples_leaf': 3, 'min_samples_split': 4}
```

Now reinitialising the best parameters and checking the score:

```
In [86]: gb=GradientBoostingRegressor(criterion='mse',min_samples_leaf=3,min_samples_split=4)

In [87]: X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y_new,test_size=0.20,random_state=50)
         gb.fit(X_train,y_train)
         pred_test=gb.predict(X_test)


         print('R-Squared:',r2_score(y_test,pred_test)*100)

         R-Squared: 86.93172876007162
```

Score has increased, but not much hence trying the next method to see the parameters can be tweaked further thereby increasing the score.

2) Trying different parameters other than Grid Search CV.

```
In [90]: gb=GradientBoostingRegressor(criterion='mae',min_samples_leaf=2,min_samples_split=3)
```

```
In [91]: X_train,X_test,y_train,y_test=train_test_split(X_Scaled,Y_new,test_size=0.20,random_state=50)
         gb.fit(X_train,y_train)
         pred_test=gb.predict(X_test)


         print('R-Squared:',r2_score(y_test,pred_test)*100)

         R-Squared: 88.24863201774133
```

<u>Score increased, thereby saving the model</u>

Conclusion

Key Findings:

Going through the data, it can be concluded that the following are the main 25 parameters that influence the price of houses in Australia. With the help of these details Surprise Housing will be able to manipulate the price and enter the Australian market.

1. Overall Qual
2. Above grade (ground) living area square feet
3. Garage Cars
4. Basement Quality
5. Basement Condition
6. External Quality
7. GarageArea
8. Total Basement Square feet
9. Kitchen Quality
10. First Floor Squarefeet
11. Full Bath
12. Garage Finish
13. Total rooms above grade (does not include bathrooms)
14. Year on which House wasBuilt
15. Year on which renovation was added
16. Masonry veneer area in square feet
17. Fire places
18. Year on which garage was built
19. Heating QC
20. Foundation
21. Type 1 finished square feet
22. Open Porch square feet
23. 2^{nd} Flr SF
24. Lot Frontage
25. WoodDeck SF