HOUSING PRICE PREDICTION

Submitted by:

Vishal Pandey

INTRODUCTION

Business Problem:

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia.

The company is looking at prospective properties to buy houses to enter the market.

We need to build a model to predict the actual value of the prospective properties and decide whether to invest in them or not. The company wants to know which properties are important to predict the price of houses and How do they describe the price of the house.

Background of domain:

- 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.
- The Australian property market comprises the trade of land and its permanent fixtures located within Australia.
- The average Australian property price grew 0.5% per year from 1890 to 1990 after inflation however rose from 1990 to 2017 at a faster rate and may be showing signs of a contracting economic bubble.
- House prices in Australia receive considerable attention from the media and the Reserve Bank and some commentators have argued that there is an Australian property bubble.
- The residential housing market has seen drastic changes in prices in the past few decades. The property prices are soaring in major cities like Sydney, Melbourne, Adelaide, Perth, Brisbane and Hobart. The median house price in Sydney peaked to $780,000 in 2016.

- However, with stricter credit policy and reduced interest from foreign investors in residential property, prices have started falling in all the major cities.
- When compared with the soaring prices of 2017, the housing prices fell by 11.1% in Sydney and 7.2% in Melbourne in 2018.
- In the late 2000s, housing prices in Australia, relative to average incomes, were among the highest in the world. As at 2011, house prices were on average six times average household income, compared to four times in 1990. This prompted speculation that the country was experiencing a real estate bubble, like many other countries.
- Foreign investment has also been identified as a key driver of affordability issues, with recent years seeing particularly high capital inflows from Chinese investors.

MOTIVATION FOR PROBLEM UNDER TAKEN:

Based on data provided from our client database, A price of house is determined based on different factors. By building the model, we can assess which characteristics are highly likely to impact on positively on sales price of a property and those characteristics which affect the price negatively.

ANALYTICAL PROBLEM FRAMING

MATHEMATICAL MODELLING OF PROBLEM:

Mathematical modeling is simply the method of implementing statistical analysis to a dataset where a Statistical Model is a mathematical representation of observed data.

While analyzing the data, there are an array of statistical models we can choose to utilize.

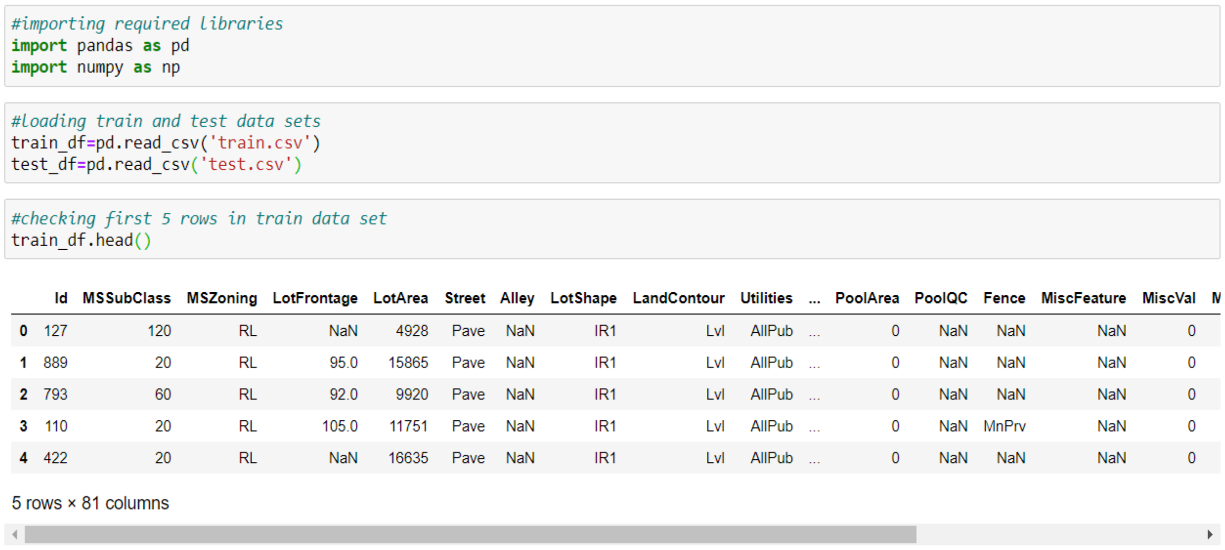
For the given project, we need to predict sales price of property based on given factors.

This is a regression problem. The classic regression model is linear regression but it is susceptible to over-fitting but it can be avoided using some dimensionality reduction techniques, regularization (L1 and L2) techniques and cross-validation.

DATA SOURCE AND FORMAT:

The data has been provided by client two different comma separated values(.csv) files.

1. The data will be loaded into pandas dataframe.



```python
#importing required libraries
import pandas as pd
import numpy as np
```

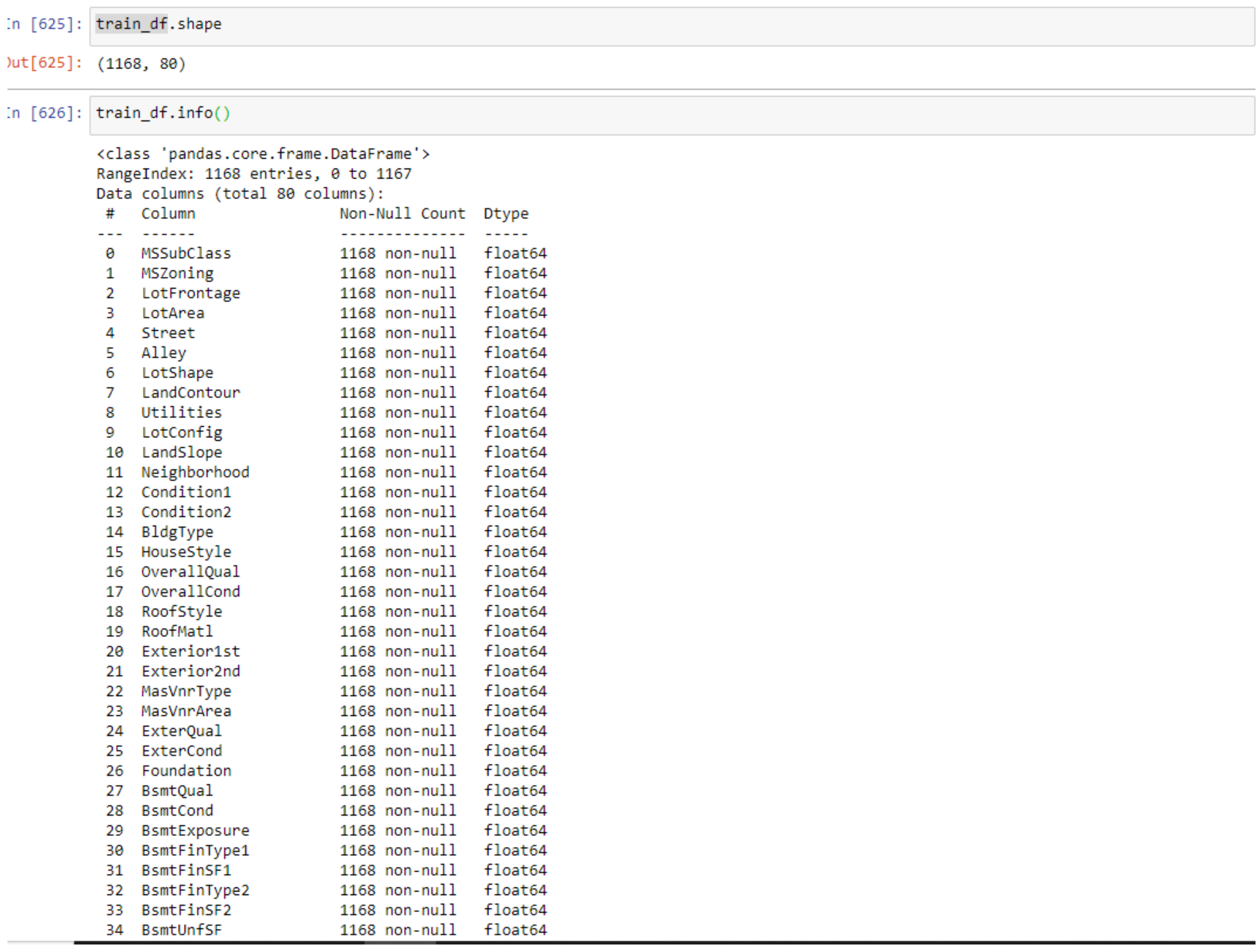
```python
#loading train and test data sets
train_df=pd.read_csv('train.csv')
test_df=pd.read_csv('test.csv')
```

```python
#checking first 5 rows in train data set
train_df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	M
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

2. Checking no. of rows and columns of the data frame and the data type of columns.



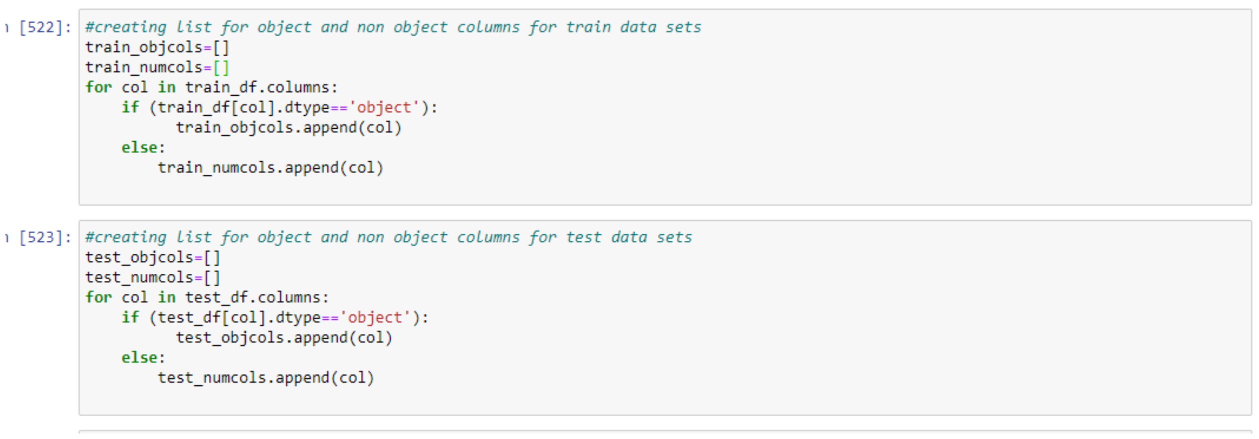
```
In [625]: train_df.shape

Out[625]: (1168, 80)

In [626]: train_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 80 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   MSSubClass      1168 non-null   float64
 1   MSZoning        1168 non-null   float64
 2   LotFrontage     1168 non-null   float64
 3   LotArea         1168 non-null   float64
 4   Street          1168 non-null   float64
 5   Alley           1168 non-null   float64
 6   LotShape        1168 non-null   float64
 7   LandContour     1168 non-null   float64
 8   Utilities       1168 non-null   float64
 9   LotConfig       1168 non-null   float64
 10  LandSlope       1168 non-null   float64
 11  Neighborhood    1168 non-null   float64
 12  Condition1      1168 non-null   float64
 13  Condition2      1168 non-null   float64
 14  BldgType        1168 non-null   float64
 15  HouseStyle      1168 non-null   float64
 16  OverallQual     1168 non-null   float64
 17  OverallCond     1168 non-null   float64
 18  RoofStyle       1168 non-null   float64
 19  RoofMatl        1168 non-null   float64
 20  Exterior1st     1168 non-null   float64
 21  Exterior2nd     1168 non-null   float64
 22  MasVnrType      1168 non-null   float64
 23  MasVnrArea      1168 non-null   float64
 24  ExterQual       1168 non-null   float64
 25  ExterCond       1168 non-null   float64
 26  Foundation      1168 non-null   float64
 27  BsmtQual        1168 non-null   float64
 28  BsmtCond        1168 non-null   float64
 29  BsmtExposure    1168 non-null   float64
 30  BsmtFinType1    1168 non-null   float64
 31  BsmtFinSF1      1168 non-null   float64
 32  BsmtFinType2    1168 non-null   float64
 33  BsmtFinSF2      1168 non-null   float64
 34  BsmtUnfSF       1168 non-null   float64
```

This data set has around 1168 rows and 80 columns.

DATA PRE PROCESSING:

Data preprocessing is a technique of converting raw data into useful format. Data cleaning is a part of preprocessing technique which involves filling missing values.

As there are large no. of columns, I dealt created separate list for object and numeric columns for both training and testing data frames.

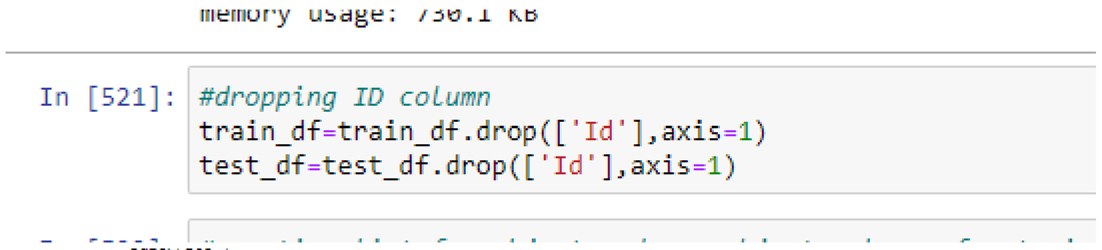


```
[522]: #creating list for object and non object columns for train data sets
       train_objcols=[]
       train_numcols=[]
       for col in train_df.columns:
           if (train_df[col].dtype=='object'):
                 train_objcols.append(col)
           else:
               train_numcols.append(col)
```

```
[523]: #creating list for object and non object columns for test data sets
       test_objcols=[]
       test_numcols=[]
       for col in test_df.columns:
           if (test_df[col].dtype=='object'):
                 test_objcols.append(col)
           else:
               test_numcols.append(col)
```

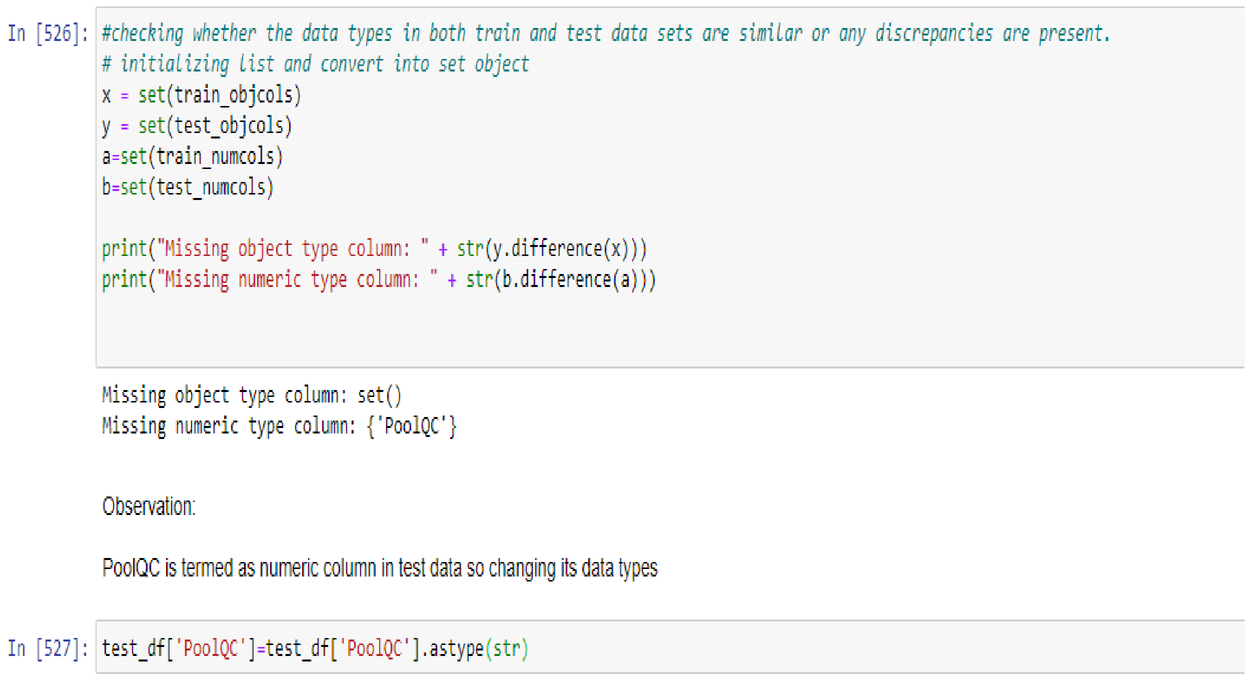
.

As the ID column doesn't impact the sales price, dropping it in both train and test data frames.



```
memory usage: 730.1 KB
```

```
In [521]: #dropping ID column
          train_df=train_df.drop(['Id'],axis=1)
          test_df=test_df.drop(['Id'],axis=1)
```



```
In [526]: #checking whether the data types in both train and test data sets are similar or any discrepancies are present.
          # initializing list and convert into set object
          x = set(train_objcols)
          y = set(test_objcols)
          a=set(train_numcols)
          b=set(test_numcols)

          print("Missing object type column: " + str(y.difference(x)))
          print("Missing numeric type column: " + str(b.difference(a)))
```

```
Missing object type column: set()
Missing numeric type column: {'PoolQC'}
```

Observation:

PoolQC is termed as numeric column in test data so changing its data types

```
In [527]: test_df['PoolQC']=test_df['PoolQC'].astype(str)
```

CHECKING FOR NULLS IN TRAIN DATA SET

In [528]:
```
#checking for nulls in object columns
for i in train_objcols:
    if(train_df[i].isnull().sum()!=0):
        print(i,":",train_df[i].isnull().sum())
    else:
        continue
```

```
Alley : 1091
MasVnrType : 7
BsmtQual : 30
BsmtCond : 30
BsmtExposure : 31
BsmtFinType1 : 30
BsmtFinType2 : 31
FireplaceQu : 551
GarageType : 64
GarageFinish : 64
GarageQual : 64
GarageCond : 64
PoolQC : 1161
Fence : 931
MiscFeature : 1124
```

In [529]:
```
#checking for nulls in numeric columns
for i in train_numcols:
    if(train_df[i].isnull().sum()!=0):
        print(i,":",train_df[i].isnull().sum())
    else:
        continue
```

```
LotFrontage : 214
MasVnrArea : 7
GarageYrBlt : 64
```

```
In [536]: #checking for nulls in test data frame
          for col in test_df.columns:
              if(test_df[col].isnull().sum()>0):
                  print(col,":",test_df[col].isnull().sum(),"  Type:",test_df[col].dtype)
              else:
                  continue
```

```
LotFrontage : 45   Type: float64
Alley : 278   Type: object
MasVnrType : 1   Type: object
MasVnrArea : 1   Type: float64
BsmtQual : 7   Type: object
BsmtCond : 7   Type: object
BsmtExposure : 7   Type: object
BsmtFinType1 : 7   Type: object
BsmtFinType2 : 7   Type: object
Electrical : 1   Type: object
FireplaceQu : 139   Type: object
GarageType : 17   Type: object
GarageYrBlt : 17   Type: float64
GarageFinish : 17   Type: object
GarageQual : 17   Type: object
GarageCond : 17   Type: object
Fence : 248   Type: object
MiscFeature : 282   Type: object
```

Treating nulls in training data set:

Checking the unique values of each column in training data set.

```
In [530]: #checking the unique entries of each column
          for i in range(len(train_objcols)):
              print(train_objcols[i],"\n","************************\n",train_df[train_objcols[i]].value_counts())
              print("###########################################################")
```

```
RRAn        20
PosN        17
RRAe         9
PosA         6
RRNn         4
RRNe         2
Name: Condition1, dtype: int64
###########################################################
Condition2
 ************************
 Norm      1154
Feedr        6
PosN         2
Artery       2
PosA         1
RRNn         1
RRAn         1
RRAe         1
Name: Condition2, dtype: int64
###########################################################
```

NOTE:

In data description,All the above object columns has 'NA' which means that the particular value is not present. when checked for the unique entries we can see that there is no NA. This indicates that the NA values might be mistakenly entred as NULL.

alley--if there is no alley access to property this might be null

bsmtqual,BsmtCond,BsmtExposure ,BsmtFinType1,BsmtFinType2 -- if there is no basement this might be chance are these columns are null.

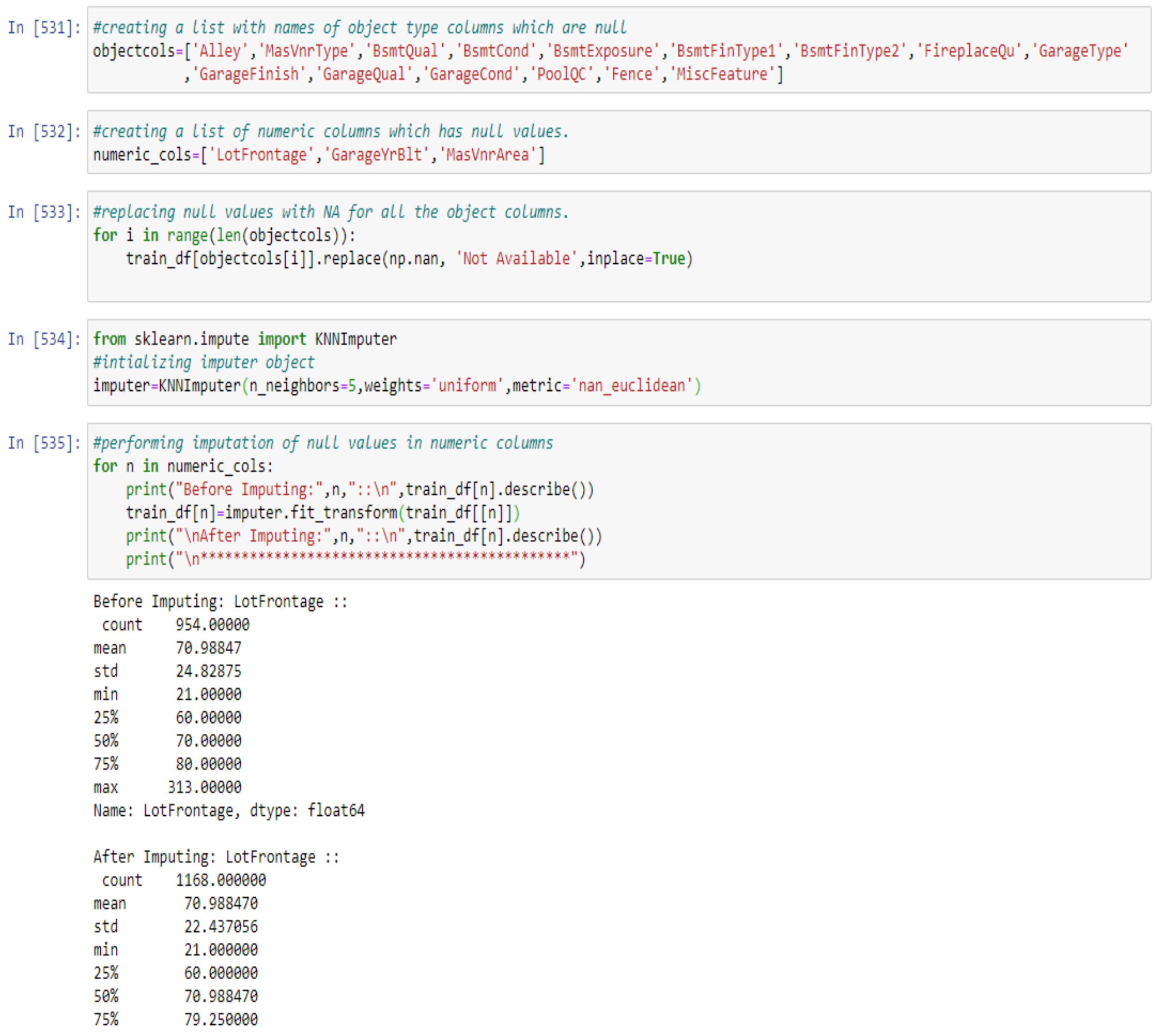
FireplaceQu -- if no fireplace is present,this column can be null.

GarageType ,GarageFinish,GarageQual-- if there is no garage these columns can be null

poolQC- If no pool in the building this column can be null

fence-If no fence to the building this column can be null

miscfeatures- If there are no miscellaneous features,this can be null



```
In [531]: #creating a list with names of object type columns which are null
          objectcols=['Alley','MasVnrType','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','FireplaceQu','GarageType'
                      ,'GarageFinish','GarageQual','GarageCond','PoolQC','Fence','MiscFeature']
```

```
In [532]: #creating a list of numeric columns which has null values.
          numeric_cols=['LotFrontage','GarageYrBlt','MasVnrArea']
```

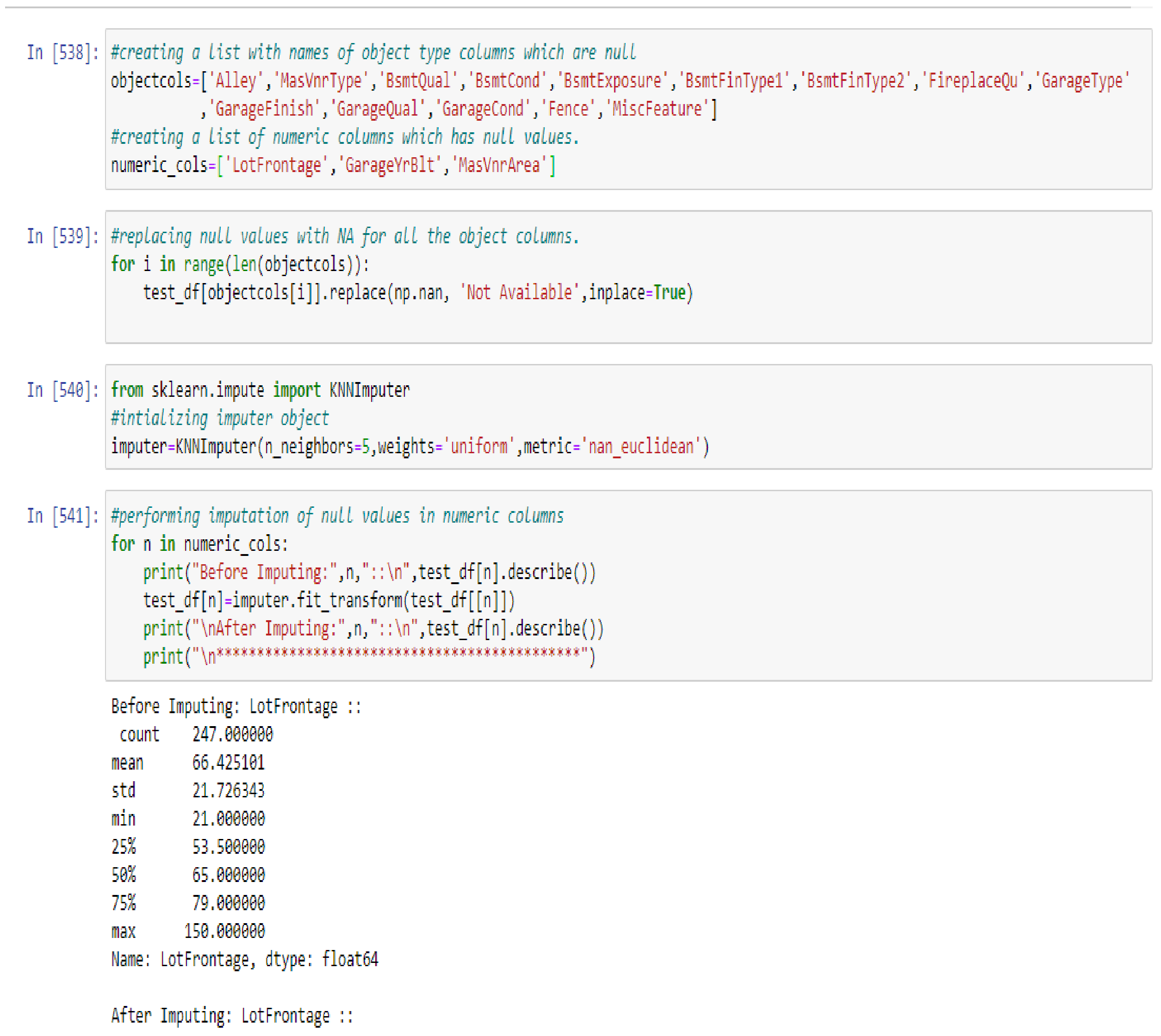
```
In [533]: #replacing null values with NA for all the object columns.
          for i in range(len(objectcols)):
              train_df[objectcols[i]].replace(np.nan, 'Not Available',inplace=True)
```

```
In [534]: from sklearn.impute import KNNImputer
          #intializing imputer object
          imputer=KNNImputer(n_neighbors=5,weights='uniform',metric='nan_euclidean')
```

```
In [535]: #performing imputation of null values in numeric columns
          for n in numeric_cols:
              print("Before Imputing:",n,"::\n",train_df[n].describe())
              train_df[n]=imputer.fit_transform(train_df[[n]])
              print("\nAfter Imputing:",n,"::\n",train_df[n].describe())
              print("\n*********************************************")
```

```
Before Imputing: LotFrontage ::
 count    954.00000
mean      70.98847
std       24.82875
min       21.00000
25%       60.00000
50%       70.00000
75%       80.00000
max      313.00000
Name: LotFrontage, dtype: float64

After Imputing: LotFrontage ::
 count    1168.000000
mean       70.988470
std        22.437056
min        21.000000
25%        60.000000
50%        70.988470
75%        79.250000
```

Treating null values in test data frame



```
In [538]: #creating a list with names of object type columns which are null
          objectcols=['Alley','MasVnrType','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2','FireplaceQu','GarageType'
                     ,'GarageFinish','GarageQual','GarageCond','Fence','MiscFeature']
          #creating a list of numeric columns which has null values.
          numeric_cols=['LotFrontage','GarageYrBlt','MasVnrArea']
```

```
In [539]: #replacing null values with NA for all the object columns.
          for i in range(len(objectcols)):
              test_df[objectcols[i]].replace(np.nan, 'Not Available',inplace=True)
```

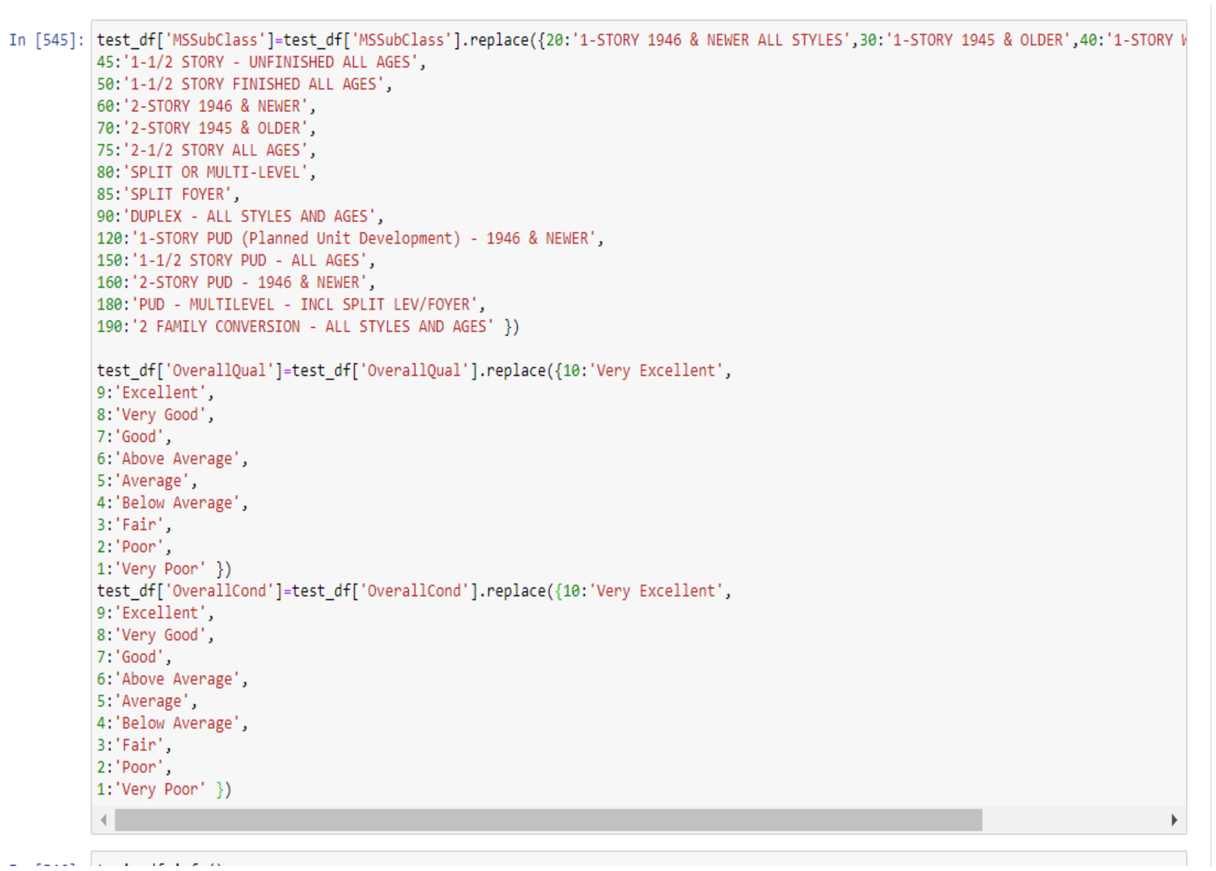
```
In [540]: from sklearn.impute import KNNImputer
          #intializing imputer object
          imputer=KNNImputer(n_neighbors=5,weights='uniform',metric='nan_euclidean')
```

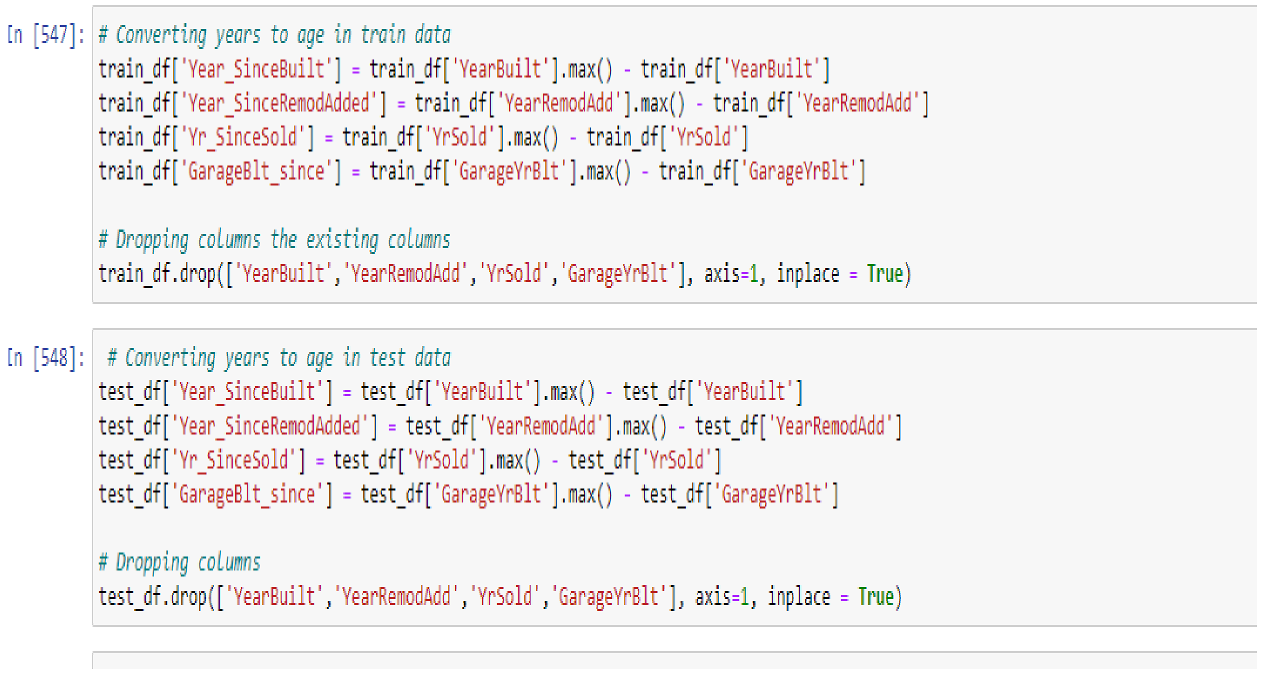
```
In [541]: #performing imputation of null values in numeric columns
          for n in numeric_cols:
              print("Before Imputing:",n,"::\n",test_df[n].describe())
              test_df[n]=imputer.fit_transform(test_df[[n]])
              print("\nAfter Imputing:",n,"::\n",test_df[n].describe())
              print("\n*********************************************")
```

```
Before Imputing: LotFrontage ::
 count    247.000000
mean      66.425101
std       21.726343
min       21.000000
25%       53.500000
50%       65.000000
75%       79.000000
max      150.000000
Name: LotFrontage, dtype: float64

After Imputing: LotFrontage ::
```

Mssubclass,over all quality and over all condition columns are converted into object columns with their values to gain better insights.

```
3]: train_df['MSSubClass']=train_df['MSSubClass'].replace({20:'1-STORY 1946 & NEWER ALL STYLES',30:'1-STORY 1945 & OLDER',40:'1-STORY
    45:'1-1/2 STORY - UNFINISHED ALL AGES',
    50:'1-1/2 STORY FINISHED ALL AGES',
    60:'2-STORY 1946 & NEWER',
    70:'2-STORY 1945 & OLDER',
    75:'2-1/2 STORY ALL AGES',
    80:'SPLIT OR MULTI-LEVEL',
    85:'SPLIT FOYER',
    90:'DUPLEX - ALL STYLES AND AGES',
    120:'1-STORY PUD (Planned Unit Development) - 1946 & NEWER',
    150:'1-1/2 STORY PUD - ALL AGES',
    160:'2-STORY PUD - 1946 & NEWER',
    180:'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER',
    190:'2 FAMILY CONVERSION - ALL STYLES AND AGES' })

    train_df['OverallQual']=train_df['OverallQual'].replace({10:'Very Excellent',
    9:'Excellent',
    8:'Very Good',
    7:'Good',
    6:'Above Average',
    5:'Average',
    4:'Below Average',
    3:'Fair',
    2:'Poor',
    1:'Very Poor' })
    train_df['OverallCond']=train_df['OverallCond'].replace({10:'Very Excellent',
    9:'Excellent',
    8:'Very Good',
    7:'Good',
    6:'Above Average',
    5:'Average',
    4:'Below Average',
    3:'Fair',
    2:'Poor',
    1:'Very Poor' })
```


```
In [545]: test_df['MSSubClass']=test_df['MSSubClass'].replace({20:'1-STORY 1946 & NEWER ALL STYLES',30:'1-STORY 1945 & OLDER',40:'1-STORY W
          45:'1-1/2 STORY - UNFINISHED ALL AGES',
          50:'1-1/2 STORY FINISHED ALL AGES',
          60:'2-STORY 1946 & NEWER',
          70:'2-STORY 1945 & OLDER',
          75:'2-1/2 STORY ALL AGES',
          80:'SPLIT OR MULTI-LEVEL',
          85:'SPLIT FOYER',
          90:'DUPLEX - ALL STYLES AND AGES',
          120:'1-STORY PUD (Planned Unit Development) - 1946 & NEWER',
          150:'1-1/2 STORY PUD - ALL AGES',
          160:'2-STORY PUD - 1946 & NEWER',
          180:'PUD - MULTILEVEL - INCL SPLIT LEV/FOYER',
          190:'2 FAMILY CONVERSION - ALL STYLES AND AGES' })

          test_df['OverallQual']=test_df['OverallQual'].replace({10:'Very Excellent',
          9:'Excellent',
          8:'Very Good',
          7:'Good',
          6:'Above Average',
          5:'Average',
          4:'Below Average',
          3:'Fair',
          2:'Poor',
          1:'Very Poor' })
          test_df['OverallCond']=test_df['OverallCond'].replace({10:'Very Excellent',
          9:'Excellent',
          8:'Very Good',
          7:'Good',
          6:'Above Average',
          5:'Average',
          4:'Below Average',
          3:'Fair',
          2:'Poor',
          1:'Very Poor' })
```


```
In [547]: # Converting years to age in train data
          train_df['Year_SinceBuilt'] = train_df['YearBuilt'].max() - train_df['YearBuilt']
          train_df['Year_SinceRemodAdded'] = train_df['YearRemodAdd'].max() - train_df['YearRemodAdd']
          train_df['Yr_SinceSold'] = train_df['YrSold'].max() - train_df['YrSold']
          train_df['GarageBlt_since'] = train_df['GarageYrBlt'].max() - train_df['GarageYrBlt']

          # Dropping columns the existing columns
          train_df.drop(['YearBuilt','YearRemodAdd','YrSold','GarageYrBlt'], axis=1, inplace = True)

In [548]:  # Converting years to age in test data
          test_df['Year_SinceBuilt'] = test_df['YearBuilt'].max() - test_df['YearBuilt']
          test_df['Year_SinceRemodAdded'] = test_df['YearRemodAdd'].max() - test_df['YearRemodAdd']
          test_df['Yr_SinceSold'] = test_df['YrSold'].max() - test_df['YrSold']
          test_df['GarageBlt_since'] = test_df['GarageYrBlt'].max() - test_df['GarageYrBlt']

          # Dropping columns
          test_df.drop(['YearBuilt','YearRemodAdd','YrSold','GarageYrBlt'], axis=1, inplace = True)
```

Hardware and Softwares Used:

Software requirement: Anaconda, Jupyter notebook

Libraries and packages used: Numpy, Pandas, Sklearn,seaborn,Matplotlib,imblearn,scipy.

Model/s Development and Evaluation

Problem-solving approach:

The data set is imbalanced since it has large no. of records which contains data about different characteristics which are considered to predict the sale price of a property. This problem comes under regression category. There are different regression algorithms. The classic regression algorithm is linear regression but,the disadvantage with linear regression is that linear regression assumes a linear relationship between dependent and independent variables. That means it assumes that there is a straight-line relationship between them. It assumes independence between attributes. Linear Regression is susceptible to over-fitting .In order to avoid it we will be using regularization (L1 and L2) techniques and cross-validation.

Statistical methods used:

Outlier removal : Mostly outliers are removed by either z score or IQR(Inter Quartile Range).Tried both these approaches first but, the data loss is high in both these approaches. So applied capping technique which is also called as winsorization.

OUTLIER REMOVAL:

```
In [574]: from scipy.stats import zscore

          z=np.abs(zscore(train_df))

          print(np.where(z>3))
```

```
(array([   1,    1,    1, ..., 1166, 1166, 1166], dtype=int64), array([10, 19, 33, ..., 38, 60, 61], dtype=int64))
```

```
In [575]: df1=train_df[(z<3).all(axis=1)]
          print("with outliers::",train_df.shape)
          print("After removing outliers::",df1.shape)
```

```
with outliers:: (1168, 80)
After removing outliers:: (0, 80)
```

IQR METHOD

```
In [576]: from scipy import stats
          IQR = stats.iqr(train_df)
          IQR
```

```
Out[576]: 5.0
```

```
In [577]: Q1 = train_df.quantile(0.25)
          Q3 = train_df.quantile(0.75)
```

```
In [578]: df_out = train_df[~((train_df < (Q1 - 1.5 * IQR)) |(train_df > (Q3 + 1.5 * IQR))).any(axis=1)]
          print(df_out.shape)
```

```
(5, 80)
```

NOTE:

Both IQR and Z score causes huge data loss.

Another technique is replacing the outlier data with mean or median.But when we obserfve this data set there is a huge difference between minimum and maximum values.If we calculate mean or median it wont give appropriate values as it includes the outlier value(maximum ones).So not using this approach.

As we are not dropping the outliers, another approach is capping or winsorization of outliers.

using percentile capping. Values that are less than the value at 10th percentile are replaced by 10th percentile value , and values greater than 90th percentile are replaced by 90th percentile value.

```
In [579]: for i in train_df.columns:
                      FloorQ=train_df[i].quantile(0.10)
                      CeilQ=train_df[i].quantile(0.90)
                      train_df[i] = np.where(train_df[i] <FloorQ,FloorQ,train_df[i])
                      train_df[i] = np.where(train_df[i] >CeilQ,CeilQ,train_df[i])
                      #checking skewness of each column
                      print(i,"->",train_df[i].skew())
```

```
MSSubClass -> 0.2763231996830171
MSZoning -> 2.0830205639783603
LotFrontage -> 0.09140893728088621
LotArea -> 0.13022197176194603
Street -> 0
Alley -> 0
LotShape -> -0.6037752483890678
LandContour -> -2.589909046438337
Utilities -> 0
LotConfig -> -1.118821122735627
LandSlope -> 0
Neighborhood -> 0.12403998681014414
Condition1 -> 0
Condition2 -> 0
BldgType -> 2.138398085468056
HouseStyle -> 0.04328965807090152
OverallQual -> 0.7046020834654392
OverallCond -> 1.0826530707035535
RoofStyle -> 1.4825606082024294
RoofMatl -> 0
Exterior1st -> -0.29774855639553294
Exterior2nd -> -0.34941727928323446
MasVnrType -> -0.801491163029031
MasVnrArea -> 1.0877858966029823
ExterQual -> -0.46837325367913335
ExterCond -> -2.2766386679193933
Foundation -> 0.15130882870028628
BsmtQual -> -0.06329729637250263
BsmtCond -> -2.5312287480710256
BsmtExposure -> -1.166986828385308
BsmtFinType1 -> 0.10081103200737304
BsmtFinSF1 -> 0.34855628344455675
BsmtFinType2 -> -2.28142748215115
BsmtFinSF2 -> 2.484900034120171
BsmtUnfSF -> 0.46135568380888786
TotalBsmtSF -> 0.40456946921095255
Heating -> 0
HeatingQC -> 0.4499332817447189
CentralAir -> 0
Electrical -> 0
1stFlrSF -> 0.42138819500880054
2ndFlrSF -> 0.5349173193148721
LowQualFinSF -> 0
GrLivArea -> 0.22761105410057356
BsmtFullBath -> 0.3552244818481984
BsmtHalfBath -> 0
```

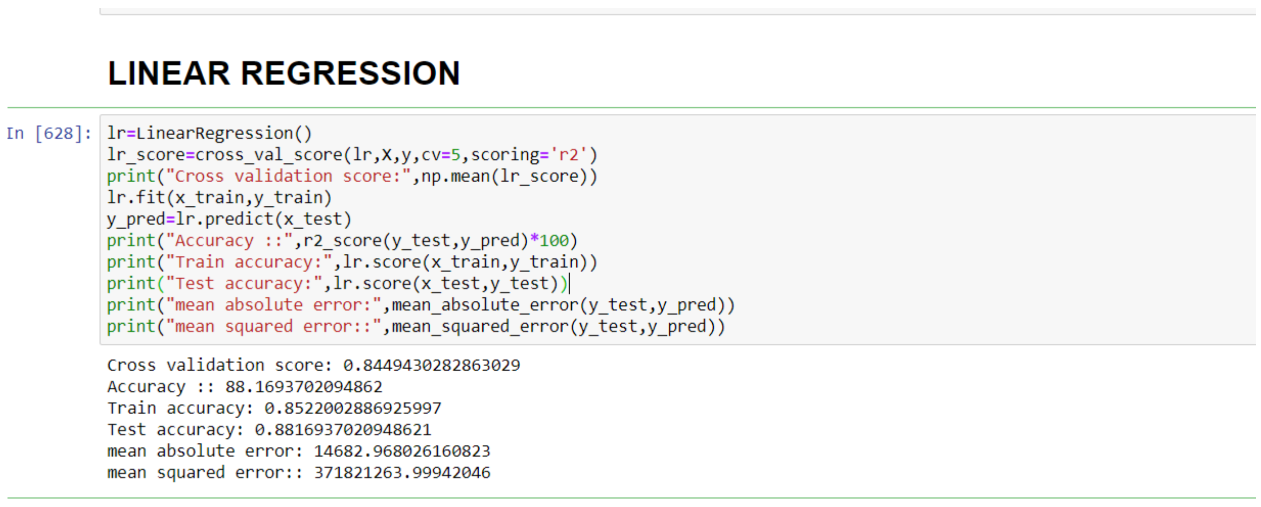
Testing of Identified Approaches (Algorithms):

List of algorithms used:

- Linear Regression
- Lasso regressiom
- Ridge regression

Run and Evaluate selected models:

Cross-validation is used to test the model's ability to predict new data that was not used in estimating it. Cross validation used in scenarios where we need to avoid over fitting.



LINEAR REGRESSION

```
In [628]: lr=LinearRegression()
          lr_score=cross_val_score(lr,X,y,cv=5,scoring='r2')
          print("Cross validation score:",np.mean(lr_score))
          lr.fit(x_train,y_train)
          y_pred=lr.predict(x_test)
          print("Accuracy ::",r2_score(y_test,y_pred)*100)
          print("Train accuracy:",lr.score(x_train,y_train))
          print("Test accuracy:",lr.score(x_test,y_test))
          print("mean absolute error:",mean_absolute_error(y_test,y_pred))
          print("mean squared error::",mean_squared_error(y_test,y_pred))
```

```
Cross validation score: 0.8449430282863029
Accuracy :: 88.1693702094862
Train accuracy: 0.8522002886925997
Test accuracy: 0.8816937020948621
mean absolute error: 14682.968026160823
mean squared error:: 371821263.99942046
```

LASSO REGRESSION

In [630]:
```
ls=Lasso()
ls_score=cross_val_score(ls,X,y,cv=5,scoring='r2')
print("Cross validation score:",np.mean(ls_score))
ls.fit(x_train,y_train)
y_pred=ls.predict(x_test)
print("Accuracy ::",r2_score(y_test,y_pred)*100)
print("mean absolute error:",mean_absolute_error(y_test,y_pred))
print("mean squared error::",mean_squared_error(y_test,y_pred))
print("Train accuracy:",ls.score(x_train,y_train))
print("Test accuracy:",ls.score(x_test,y_test))
```

```
Cross validation score: 0.8449479206638193
Accuracy :: 88.17570941630396
mean absolute error: 14678.429774135926
mean squared error:: 371622030.9971658
Train accuracy: 0.8522005697301187
Test accuracy: 0.8817570941630396
```

RIDGE REGRESSION

[631]:
```
rg=Ridge()
rg_score=cross_val_score(rg,X,y,cv=5,scoring='r2')
print("Cross validation score:",np.mean(rg_score))
rg.fit(x_train,y_train)
y_pred=rg.predict(x_test)
print("Accuracy ::",r2_score(y_test,y_pred)*100)
print("mean absolute error:",mean_absolute_error(y_test,y_pred))
print("mean squared error::",mean_squared_error(y_test,y_pred))
print("Train accuracy:",rg.score(x_train,y_train))
print("Test accuracy:",rg.score(x_test,y_test))
```

```
Cross validation score: 0.8450301816295296
Accuracy :: 88.19554886385342
mean absolute error: 14666.86419322152
mean squared error:: 370998503.0366506
Train accuracy: 0.8521965802636385
Test accuracy: 0.8819554886385341
```

Key Metrics for success in solving problem under consideration:

The following metrics are used :

1)R2_score: In multiregression models, R2 corresponds to the squared correlation between the observed outcome values and the predicted values by the model. The Higher the R-squared, the better the model.

2)Mean Absolute Error: It is the sum of the ‘absolute differences’ between the predicted and the actual values,

3)Mean Squared Error: It is the sum of squares of the errors, where errors are the differences between the ‘predicted’ and ‘actual’ values

HYPER PARAMETER TUNING:

Hyper parameter tuning is used to increase the performance of the algorithm.

HYPER PARAMETER TUNING

```
In [594]: from sklearn.model_selection import GridSearchCV
```

```
In [595]: params = {'alpha': [1,0.1,0.01,0.001,0.0001,0]}
          lasso = Lasso()

          # cross validation
          laso_grid = GridSearchCV(estimator = lasso,
                                   param_grid = params,
                                   scoring= 'r2',
                                   cv = 10,
                                   return_train_score=True,
                                   verbose = 1)

          laso_grid.fit(x_train, y_train)
```

```
          Fitting 10 folds for each of 6 candidates, totalling 60 fits

Out[595]: GridSearchCV(cv=10, estimator=Lasso(),
                       param_grid={'alpha': [1, 0.1, 0.01, 0.001, 0.0001, 0]},
                       return_train_score=True, scoring='r2', verbose=1)
```

```
In [596]: laso_grid.best_params_
```

```
Out[596]: {'alpha': 1}
```

```
In [597]: ls1=Lasso(alpha=1)
          ls1.fit(x_train,y_train)
          y_pred=ls1.predict(x_test)
          print("Accuracy ::",r2_score(y_test,y_pred)*100)
          print("mean absolute error:",mean_absolute_error(y_test,y_pred))
          print("mean squared error::",mean_squared_error(y_test,y_pred))
```

```
          Accuracy :: 88.17570941630396
          mean absolute error: 14678.429774135926
          mean squared error:: 371622030.9971658
```

Ridge Regression hyper parameter

In [603]:
```
param = {'alpha':[0.001,0.01,0.1,0.2,0.5,0.9,1.0, 5.0, 10.0,20.0]}

ridge = Ridge()

ridge_grid = GridSearchCV(estimator=ridge,
                        param_grid=param,
                        scoring='r2',
                        cv=10,
                        verbose=1,
                        return_train_score=True)

ridge_grid.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Out[603]:
```
GridSearchCV(cv=10, estimator=Ridge(),
             param_grid={'alpha': [0.001, 0.01, 0.1, 0.2, 0.5, 0.9, 1.0, 5.0,
                                   10.0, 20.0]},
             return_train_score=True, scoring='r2', verbose=1)
```

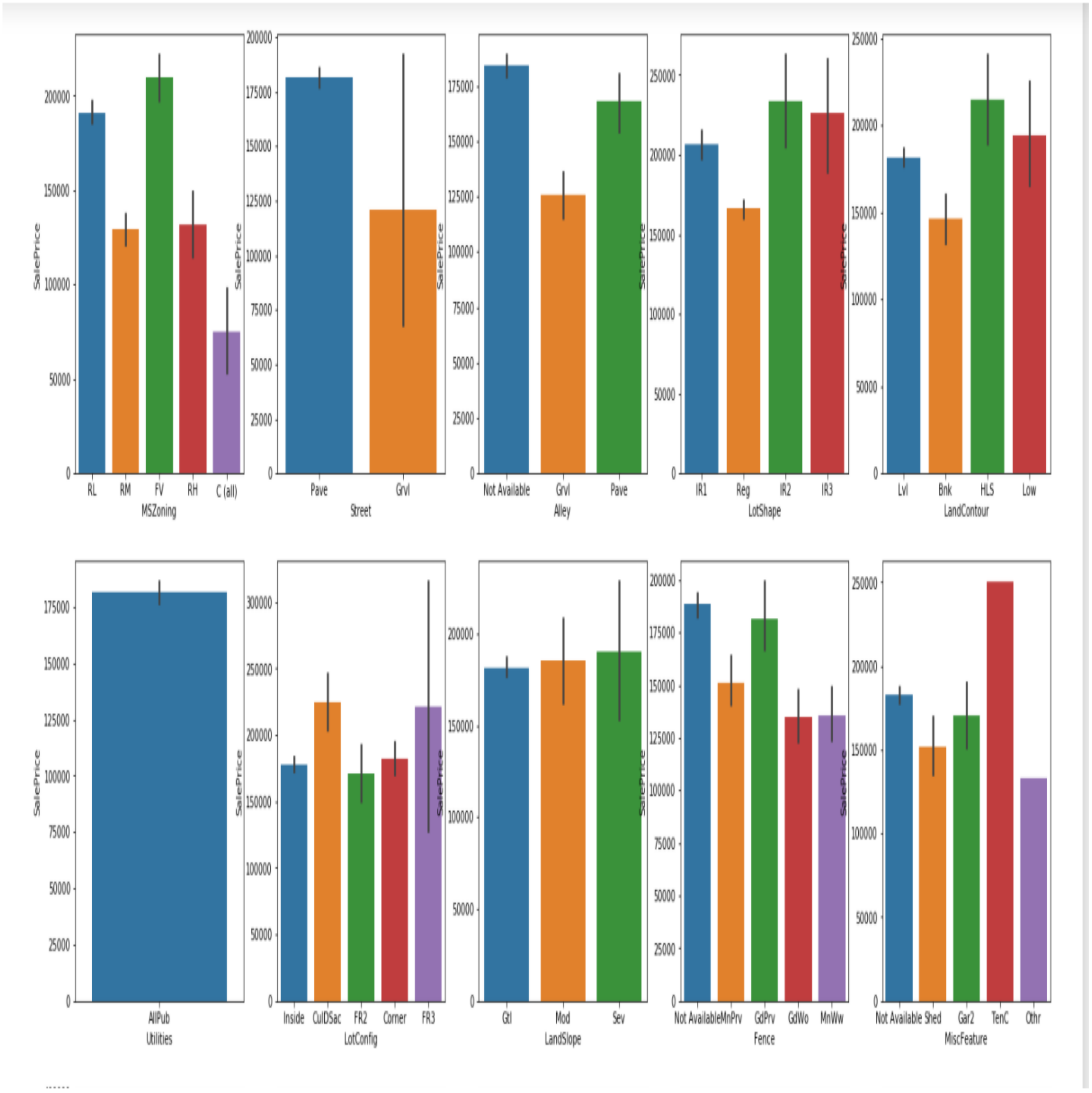
In [604]:
```
ridge_grid.best_params_
```

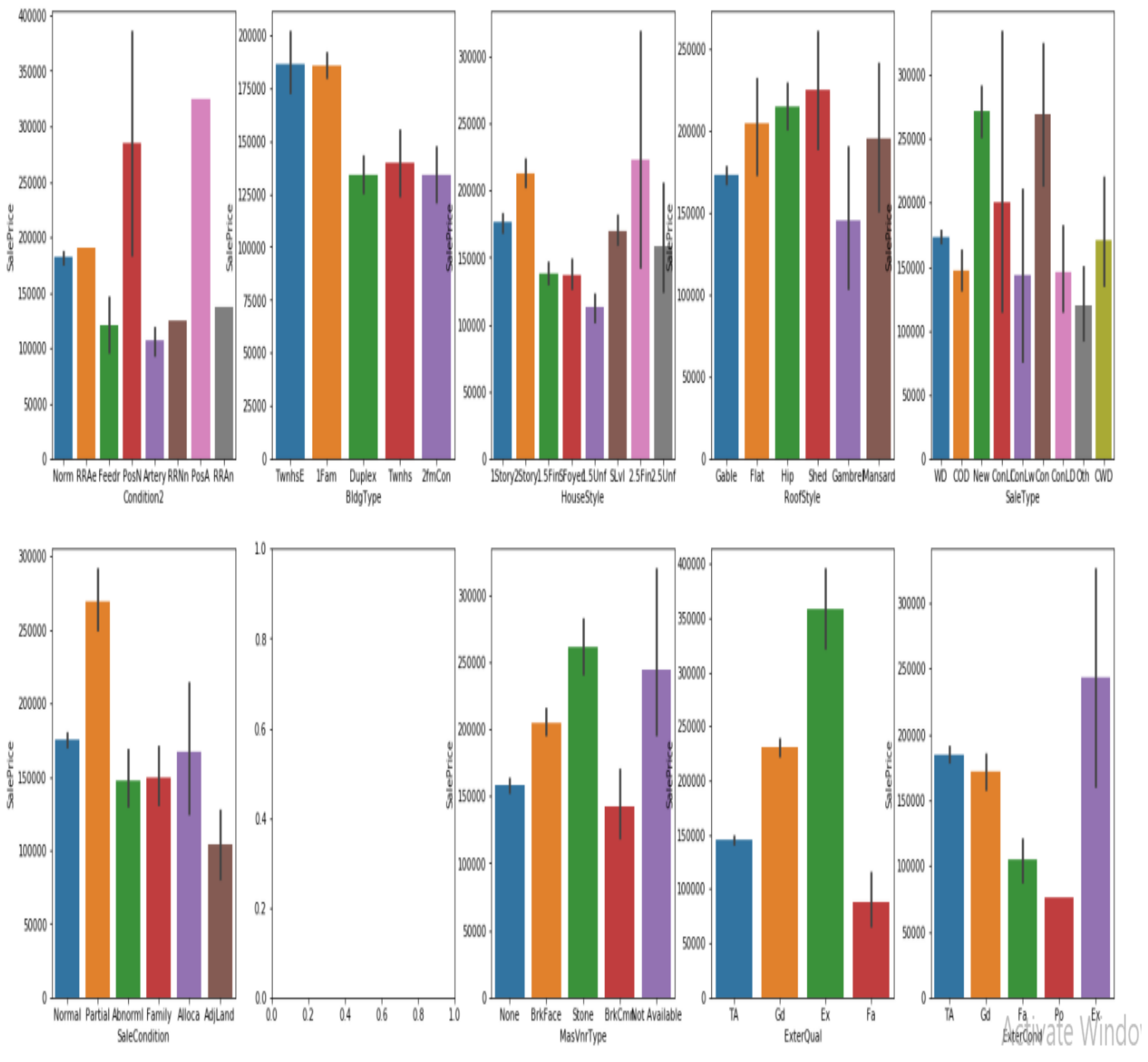
Out[604]:
```
{'alpha': 20.0}
```

In [605]:
```
rg1=Ridge(alpha=20)
rg1.fit(x_train,y_train)
y_pred=rg1.predict(x_test)
print("Accuracy ::",r2_score(y_test,y_pred)*100)
print("mean absolute error:",mean_absolute_error(y_test,y_pred))
print("mean squared error::",mean_squared_error(y_test,y_pred))
```

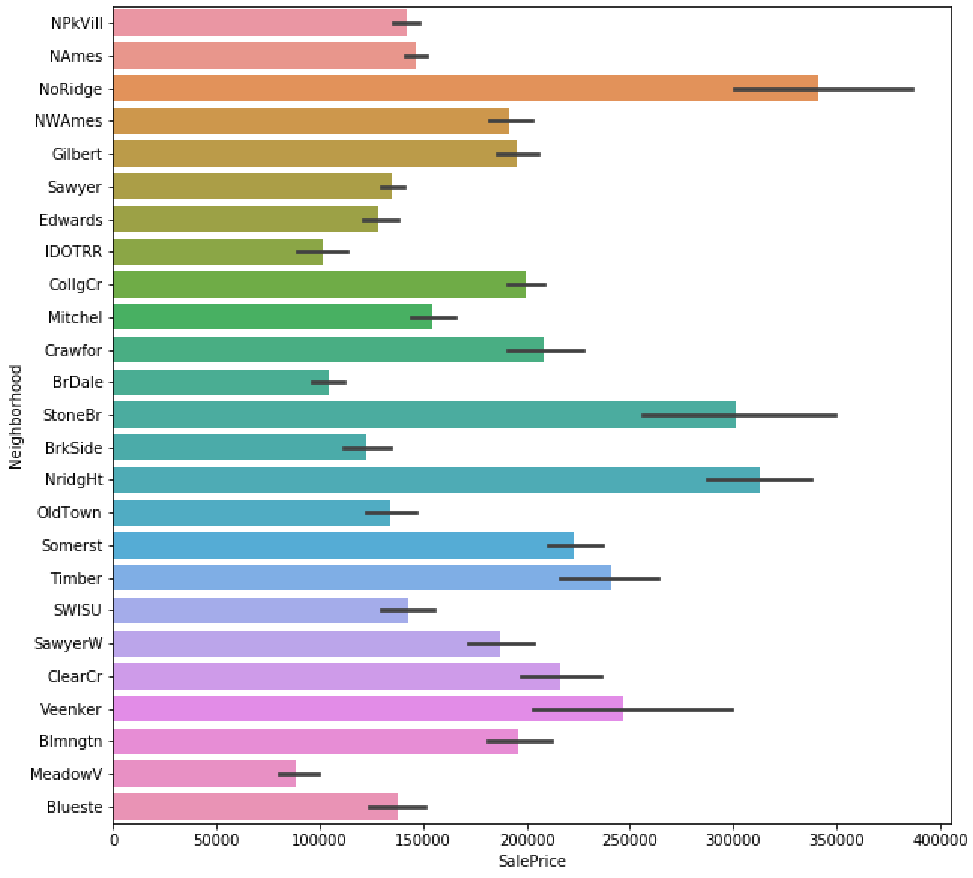
```
Accuracy :: 88.439238738436
mean absolute error: 14547.094319408572
mean squared error:: 363339648.11551946
```

Visualizations:

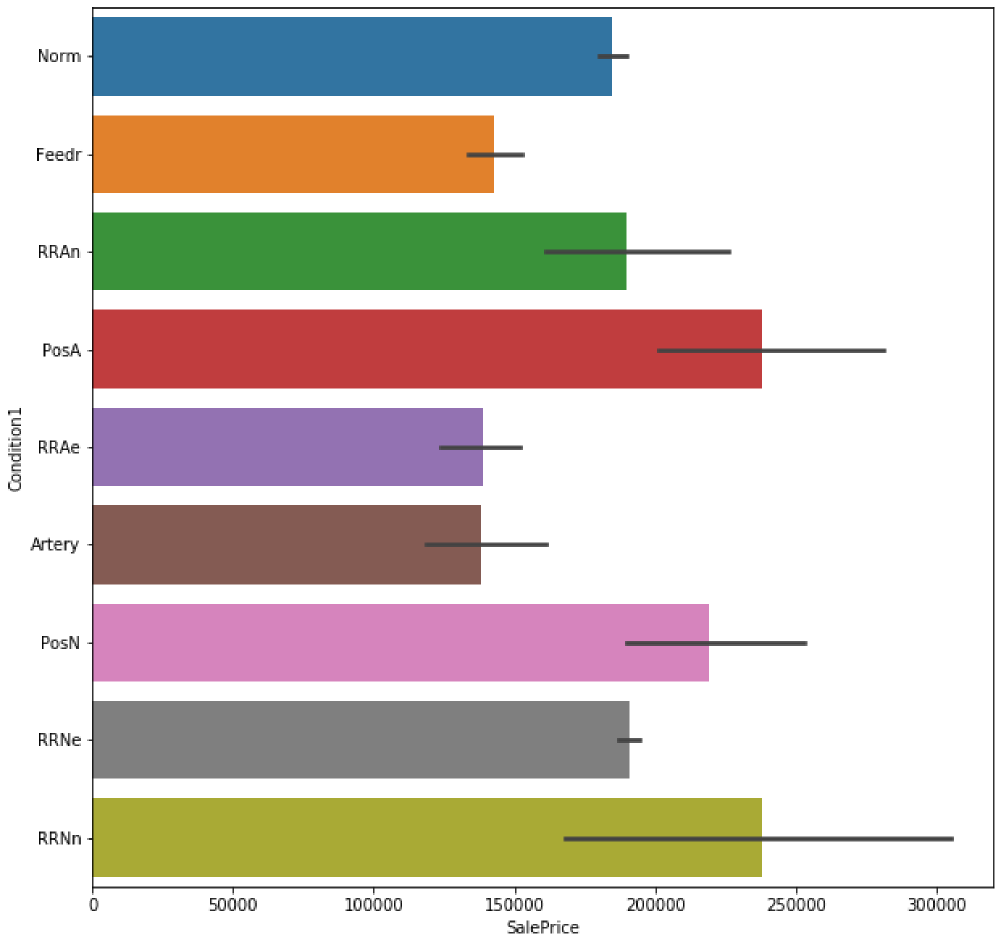



SalePrice
Norm RRAe Feedr PosN Artery RRNn PosA RRAn
Condition2
TwnhsE 1Fam Duplex Twnhs 2fmCon
BldgType
1Story 2Story 1.5Fin SFoyer 1.5Unf SLvl 2.5Fin 2.5Unf
HouseStyle
Gable Flat Hip Shed Gambrel Mansard
RoofStyle
WD COD New ConLI ConLw Con ConLD Oth CWD
SaleType
Normal Partial Abnorml Family Alloca AdjLand
SaleCondition
None BrkFace Stone BrkCmn Not Available
MasVnrType
TA Gd Ex Fa
ExterQual
TA Gd Fa Po Ex
ExterCond

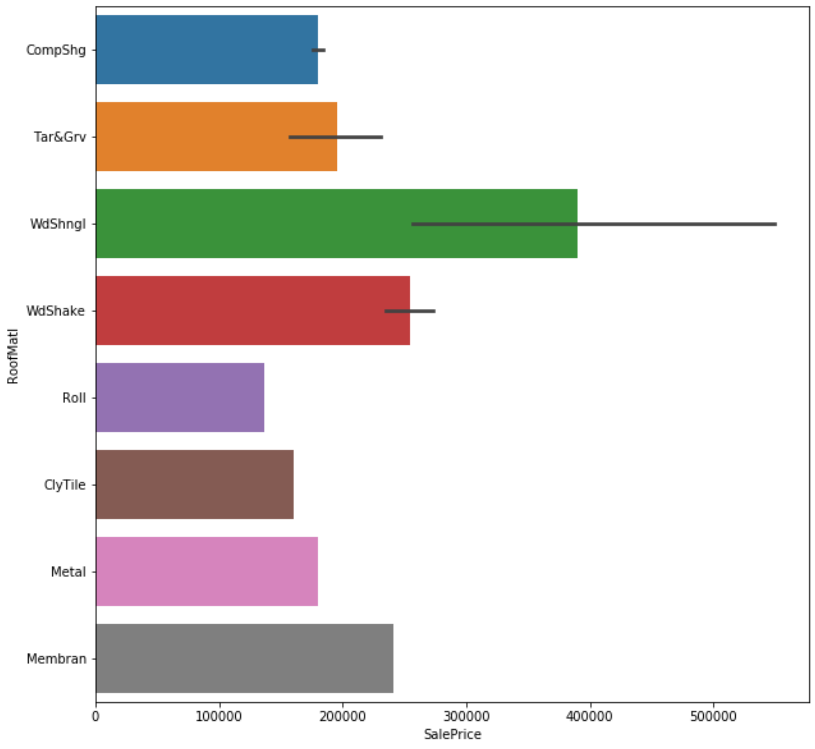
```
In [556]: plt.figure(figsize=(10,10))
          sns.barplot(y=train_df['Neighborhood'],x=train_df['SalePrice'])

Out[556]: <matplotlib.axes._subplots.AxesSubplot at 0x2cf44459548>
```



```
In [557]: plt.figure(figsize=(10,10))
          sns.barplot(y=train_df['Condition1'],x=train_df['SalePrice'])

Out[557]: <matplotlib.axes._subplots.AxesSubplot at 0x2cf448828c8>
```

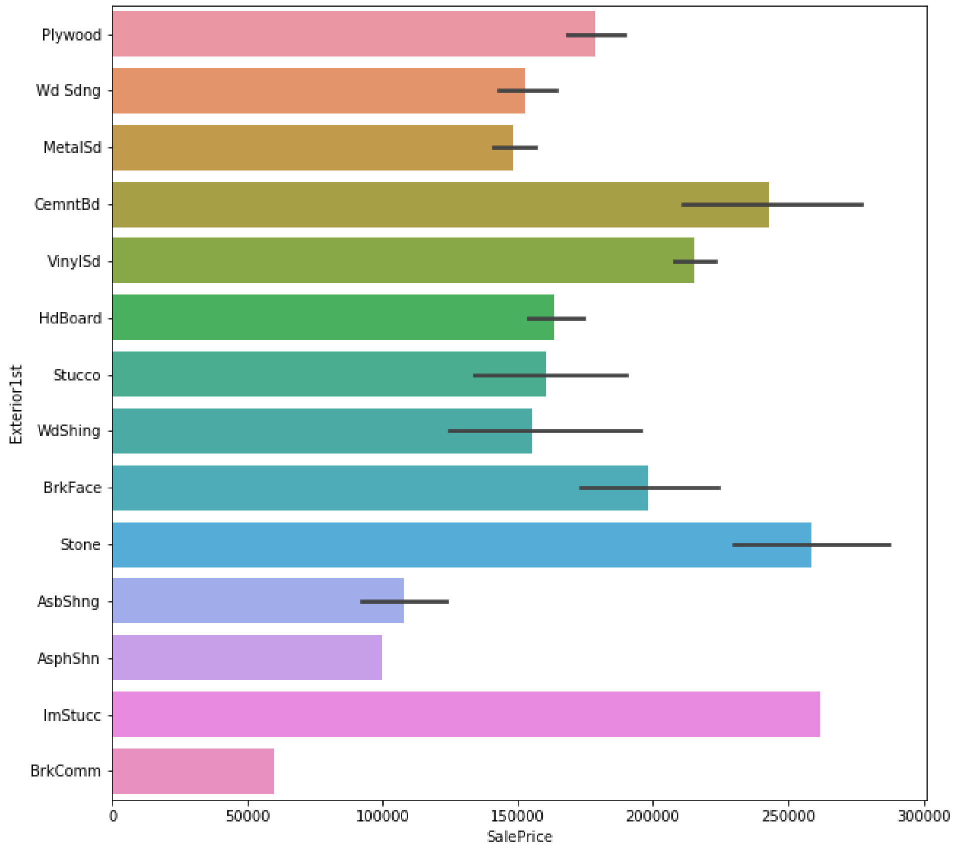


```
In [558]: plt.figure(figsize=(10,10))
          sns.barplot(y=train_df['RoofMatl'],x=train_df['SalePrice'])

Out[558]: <matplotlib.axes._subplots.AxesSubplot at 0x2cf4464bb08>
```



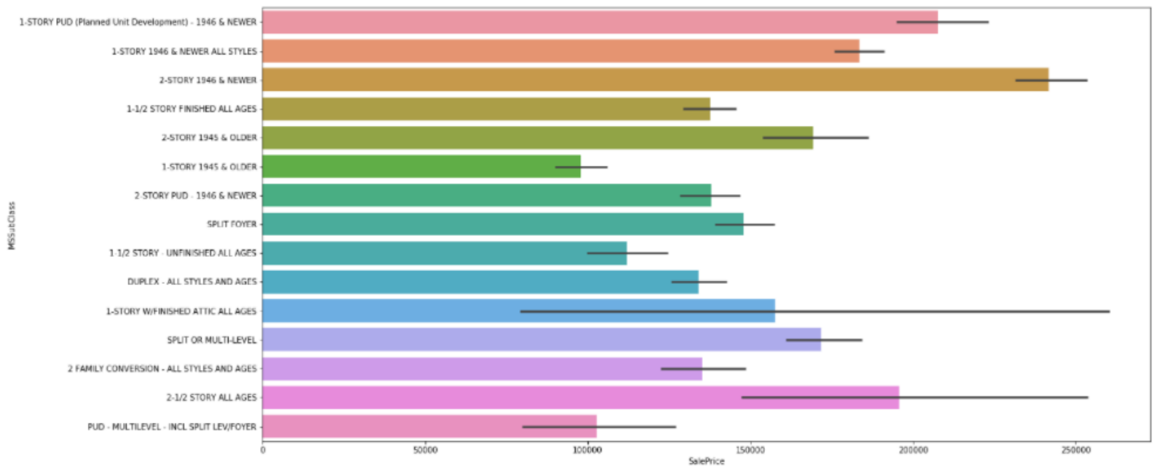
```
In [559]: plt.figure(figsize=(10,10))
          sns.barplot(y=train_df['Exterior1st'],x=train_df['SalePrice'])
```

```
Out[559]: <matplotlib.axes._subplots.AxesSubplot at 0x2cf4492e748>
```



```
In [561]: plt.figure(figsize=(20,10))
          sns.barplot(y=train_df['MSSubClass'],x=train_df['SalePrice'])
```

```
Out[561]: <matplotlib.axes._subplots.AxesSubplot at 0x2cf4587da88>
```



Interpretation of the Results:

Floating Village Residential,Residential Low Density has high sales price and commercial buildings has less price

- properties with paved streets has high sales prices compared to gravel roads
- properties with paved alleys has high sales prices.
- regular shaped properties has less price,moderately irregular properties has high prices
- hill side properties has high sales price
- the data we collected every building has all the utilities.
- properties with Cul-de-sac kind of lot configuration has high sale price
- properties with moderate to severe slope has high sale prices
- properties in Northridge,Northridge Heights,Stone Brook locations tops in sales prices
- properties within Within 200' of North-South Railroad has highest price followed by properties with Adjacency to postive off-site feature
- Townhouse Inside Unit,Single-family Detached type of properties has high sale prices
- Two and one-half story properties with 2nd level finished has high sale prices followed by 2 story buildings
- properties with shed type of roofs has high sale prices followed by hip and flat roofs.
- properties which just got constructed has high sale prices and properties with 15% Contract Down payment regular terms has also high sale prices.
- Homes that are partially constructed during last assessment has high sale prices.

- properties with stone type of Masonry veneers has high prices.
- properties with excellent exterior material quality and excellent exterior condition has higher sale prices.
- properties with poured concrete type of foundation has high sale prices.
- properties with basement height of 100+ inches has high sale price

- properties with good basement condition has higher sale price where as properties with poor basement condition has low price
- properties with good exposure to walkout or garden level walls has high sales price
- properties with Good Living Quarters has high sales prices
- properties with Gas forced warm air furnace and Gas hot water or steam heat has high sale prices.
- properties with ecellent heating conditions tend to have high prices
- properties with central air conditioning has high sales prices.
- properties with Standard Circuit Breakers & Romex has high sales values
- properties with excellent kitchen quality has high price.
- typical functioning homes has high sale price.
- properties with excellent fireplace quality has high price.
- properties with built in garages has high salesprice followed by attached garages.properties which do not have any garage has less sales price
- If the interior of garage is completely finished,those properties have high sale price
- properties with garages in good condition with excellent quality has high sale prices.

- properties with paved drive ways has high sales prices.
- properties with excellent pool quality has high prices
- properties with 2-STORY built in 1946 or NEWER has high sales prices.

CONCLUSION

- **Key Findings and Conclusions of the Study**:
  The following are top 10 characteristics that positively influence the saleprices:
    Above grade (ground) living area square feet,
    Total square feet of basement area
    Overall quality
    Number of fireplaces
    Second floor square feet

Type 2 finished square feet
Rating of basement finished area (if multiple types)
Size of garage in square feet
Lot size in square feet
Exterior covering on house (if more than one material)

The following are top 10 characteristics that negatively influence saleprices:
Interior finish of the garage
No.of Bedrooms above ground
Kitchen quality
Basement quality
Fireplace quality
Building type
Exterior quality
Years since built
Years since remodeling has been done.

- **Learning Outcomes of the Study in respect of Data Science**

One of the challenge i faced while data cleaning is outlier removal, in most of the scenarios Z-score will be used as outlier removal technique since it performs quite well with less data loss. In our data set, Z-score has almost every data deleted. Then I tried another famous technique called InterQuartileRange it also caused huge data loss.

Another technique is replacing the outlier data with mean or median. But when we observe this data set there is a huge difference between minimum and maximum values. If we calculate mean or median it won't give appropriate values as it includes the outlier value (maximum ones).So not using this approach.

As we are not dropping the outliers, another approach is capping or winsorization of outliers .Using percentile capping. Values that are less than the value at 10th percentile are replaced by 10th percentile value, and the value greater than 90th percentile are replaced by 90th percentile value.

Limitations of this work and Scope for Future Work:

This data set contains data of the year 2010 belonging to surprise house.

If we get data of recent years along with different popular cities ,we can predict on varied scenarios.