# HOUSING: PRICE PREDICTION :Problem Statement: Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world’s economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company. A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price. For the same purpose, the company has collected a data set from the sale of houses in Australia. The data is provided in the CSV file below. The company is looking at prospective properties to buy houses to enter the market. You are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not. For this company wants to know: • Which variables are important to predict the price of variable? • How do these variables describe the price of the house? Business Goal: You are required to model the price of houses with the available independent variables. This model will then be used by the management to understand how exactly the prices vary with the variables. They can accordingly manipulate the strategy of the firm and concentrate on areas that will yield high returns. Further, the model will be a good way for the management to understand the pricing dynamics of a new market. House Price Prediction

## Importing Libraries

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
  
from sklearn.model\_selection import GridSearchCV  
from sklearn.linear\_model import LinearRegression, Ridge, Lasso  
from sklearn.metrics import mean\_squared\_error, r2\_score  
  
import warnings  
warnings.filterwarnings('ignore')

## Reading the data

df=pd.read\_csv(r'C:\users\rksan\Downloads\train.csv')  
df.head()

Id MSSubClass MSZoning LotFrontage LotArea Street Alley LotShape \  
0 127 120 RL NaN 4928 Pave NaN IR1   
1 889 20 RL 95.0 15865 Pave NaN IR1   
2 793 60 RL 92.0 9920 Pave NaN IR1   
3 110 20 RL 105.0 11751 Pave NaN IR1   
4 422 20 RL NaN 16635 Pave NaN IR1   
  
 LandContour Utilities ... PoolArea PoolQC Fence MiscFeature MiscVal \  
0 Lvl AllPub ... 0 NaN NaN NaN 0   
1 Lvl AllPub ... 0 NaN NaN NaN 0   
2 Lvl AllPub ... 0 NaN NaN NaN 0   
3 Lvl AllPub ... 0 NaN MnPrv NaN 0   
4 Lvl AllPub ... 0 NaN NaN NaN 0   
  
 MoSold YrSold SaleType SaleCondition SalePrice   
0 2 2007 WD Normal 128000   
1 10 2007 WD Normal 268000   
2 6 2007 WD Normal 269790   
3 1 2010 COD Normal 190000   
4 6 2009 WD Normal 215000   
  
[5 rows x 81 columns]

## Performing basic checks on data set

df.shape

(1168, 81)

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1168 entries, 0 to 1167  
Data columns (total 81 columns):  
Id 1168 non-null int64  
MSSubClass 1168 non-null int64  
MSZoning 1168 non-null object  
LotFrontage 954 non-null float64  
LotArea 1168 non-null int64  
Street 1168 non-null object  
Alley 77 non-null object  
LotShape 1168 non-null object  
LandContour 1168 non-null object  
Utilities 1168 non-null object  
LotConfig 1168 non-null object  
LandSlope 1168 non-null object  
Neighborhood 1168 non-null object  
Condition1 1168 non-null object  
Condition2 1168 non-null object  
BldgType 1168 non-null object  
HouseStyle 1168 non-null object  
OverallQual 1168 non-null int64  
OverallCond 1168 non-null int64  
YearBuilt 1168 non-null int64  
YearRemodAdd 1168 non-null int64  
RoofStyle 1168 non-null object  
RoofMatl 1168 non-null object  
Exterior1st 1168 non-null object  
Exterior2nd 1168 non-null object  
MasVnrType 1161 non-null object  
MasVnrArea 1161 non-null float64  
ExterQual 1168 non-null object  
ExterCond 1168 non-null object  
Foundation 1168 non-null object  
BsmtQual 1138 non-null object  
BsmtCond 1138 non-null object  
BsmtExposure 1137 non-null object  
BsmtFinType1 1138 non-null object  
BsmtFinSF1 1168 non-null int64  
BsmtFinType2 1137 non-null object  
BsmtFinSF2 1168 non-null int64  
BsmtUnfSF 1168 non-null int64  
TotalBsmtSF 1168 non-null int64  
Heating 1168 non-null object  
HeatingQC 1168 non-null object  
CentralAir 1168 non-null object  
Electrical 1168 non-null object  
1stFlrSF 1168 non-null int64  
2ndFlrSF 1168 non-null int64  
LowQualFinSF 1168 non-null int64  
GrLivArea 1168 non-null int64  
BsmtFullBath 1168 non-null int64  
BsmtHalfBath 1168 non-null int64  
FullBath 1168 non-null int64  
HalfBath 1168 non-null int64  
BedroomAbvGr 1168 non-null int64  
KitchenAbvGr 1168 non-null int64  
KitchenQual 1168 non-null object  
TotRmsAbvGrd 1168 non-null int64  
Functional 1168 non-null object  
Fireplaces 1168 non-null int64  
FireplaceQu 617 non-null object  
GarageType 1104 non-null object  
GarageYrBlt 1104 non-null float64  
GarageFinish 1104 non-null object  
GarageCars 1168 non-null int64  
GarageArea 1168 non-null int64  
GarageQual 1104 non-null object  
GarageCond 1104 non-null object  
PavedDrive 1168 non-null object  
WoodDeckSF 1168 non-null int64  
OpenPorchSF 1168 non-null int64  
EnclosedPorch 1168 non-null int64  
3SsnPorch 1168 non-null int64  
ScreenPorch 1168 non-null int64  
PoolArea 1168 non-null int64  
PoolQC 7 non-null object  
Fence 237 non-null object  
MiscFeature 44 non-null object  
MiscVal 1168 non-null int64  
MoSold 1168 non-null int64  
YrSold 1168 non-null int64  
SaleType 1168 non-null object  
SaleCondition 1168 non-null object  
SalePrice 1168 non-null int64  
dtypes: float64(3), int64(35), object(43)  
memory usage: 739.2+ KB

df.describe()

Id MSSubClass LotFrontage LotArea OverallQual \  
count 1168.000000 1168.000000 954.00000 1168.000000 1168.000000   
mean 724.136130 56.767979 70.98847 10484.749144 6.104452   
std 416.159877 41.940650 24.82875 8957.442311 1.390153   
min 1.000000 20.000000 21.00000 1300.000000 1.000000   
25% 360.500000 20.000000 60.00000 7621.500000 5.000000   
50% 714.500000 50.000000 70.00000 9522.500000 6.000000   
75% 1079.500000 70.000000 80.00000 11515.500000 7.000000   
max 1460.000000 190.000000 313.00000 164660.000000 10.000000   
  
 OverallCond YearBuilt YearRemodAdd MasVnrArea BsmtFinSF1 ... \  
count 1168.000000 1168.000000 1168.000000 1161.000000 1168.000000 ...   
mean 5.595890 1970.930651 1984.758562 102.310078 444.726027 ...   
std 1.124343 30.145255 20.785185 182.595606 462.664785 ...   
min 1.000000 1875.000000 1950.000000 0.000000 0.000000 ...   
25% 5.000000 1954.000000 1966.000000 0.000000 0.000000 ...   
50% 5.000000 1972.000000 1993.000000 0.000000 385.500000 ...   
75% 6.000000 2000.000000 2004.000000 160.000000 714.500000 ...   
max 9.000000 2010.000000 2010.000000 1600.000000 5644.000000 ...   
  
 WoodDeckSF OpenPorchSF EnclosedPorch 3SsnPorch ScreenPorch \  
count 1168.000000 1168.000000 1168.000000 1168.000000 1168.000000   
mean 96.206336 46.559932 23.015411 3.639555 15.051370   
std 126.158988 66.381023 63.191089 29.088867 55.080816   
min 0.000000 0.000000 0.000000 0.000000 0.000000   
25% 0.000000 0.000000 0.000000 0.000000 0.000000   
50% 0.000000 24.000000 0.000000 0.000000 0.000000   
75% 171.000000 70.000000 0.000000 0.000000 0.000000   
max 857.000000 547.000000 552.000000 508.000000 480.000000   
  
 PoolArea MiscVal MoSold YrSold SalePrice   
count 1168.000000 1168.000000 1168.000000 1168.000000 1168.000000   
mean 3.448630 47.315068 6.344178 2007.804795 181477.005993   
std 44.896939 543.264432 2.686352 1.329738 79105.586863   
min 0.000000 0.000000 1.000000 2006.000000 34900.000000   
25% 0.000000 0.000000 5.000000 2007.000000 130375.000000   
50% 0.000000 0.000000 6.000000 2008.000000 163995.000000   
75% 0.000000 0.000000 8.000000 2009.000000 215000.000000   
max 738.000000 15500.000000 12.000000 2010.000000 755000.000000   
  
[8 rows x 38 columns]

#Percentage of null values in each columns  
(df.isnull().sum()/df.shape[0])\*100

Id 0.000000  
MSSubClass 0.000000  
MSZoning 0.000000  
LotFrontage 18.321918  
LotArea 0.000000  
 ...   
MoSold 0.000000  
YrSold 0.000000  
SaleType 0.000000  
SaleCondition 0.000000  
SalePrice 0.000000  
Length: 81, dtype: float64

## Performing EDA

def calc\_null(df):  
 return round((df.isnull().sum()/df.shape[0])\*100,3)

### 1. Dropping the columns having more than 45% of missing values

calc\_null(df)

Id 0.000  
MSSubClass 0.000  
MSZoning 0.000  
LotFrontage 18.322  
LotArea 0.000  
 ...   
MoSold 0.000  
YrSold 0.000  
SaleType 0.000  
SaleCondition 0.000  
SalePrice 0.000  
Length: 81, dtype: float64

df = df.loc[:,~(df.isnull().sum()/df.shape[0]>0.45)]

df.shape

(1168, 76)

# Dropping unique columns as well  
df=df.drop('Id',axis=1)

# Handling years column  
#df['Age']=2020-df.GarageYrBlt  
#df=df.drop('GarageYrBlt',axis=1)

df.head()

MSSubClass MSZoning LotFrontage LotArea Street LotShape LandContour \  
0 120 RL NaN 4928 Pave IR1 Lvl   
1 20 RL 95.0 15865 Pave IR1 Lvl   
2 60 RL 92.0 9920 Pave IR1 Lvl   
3 20 RL 105.0 11751 Pave IR1 Lvl   
4 20 RL NaN 16635 Pave IR1 Lvl   
  
 Utilities LotConfig LandSlope ... EnclosedPorch 3SsnPorch ScreenPorch \  
0 AllPub Inside Gtl ... 0 0 0   
1 AllPub Inside Mod ... 0 0 224   
2 AllPub CulDSac Gtl ... 0 0 0   
3 AllPub Inside Gtl ... 0 0 0   
4 AllPub FR2 Gtl ... 0 0 0   
  
 PoolArea MiscVal MoSold YrSold SaleType SaleCondition SalePrice   
0 0 0 2 2007 WD Normal 128000   
1 0 0 10 2007 WD Normal 268000   
2 0 0 6 2007 WD Normal 269790   
3 0 0 1 2010 COD Normal 190000   
4 0 0 6 2009 WD Normal 215000   
  
[5 rows x 75 columns]

#### 2. Checking the NA values in each column and treating it accordingly

calc\_null(df)

MSSubClass 0.000  
MSZoning 0.000  
LotFrontage 18.322  
LotArea 0.000  
Street 0.000  
 ...   
MoSold 0.000  
YrSold 0.000  
SaleType 0.000  
SaleCondition 0.000  
SalePrice 0.000  
Length: 75, dtype: float64

#Checking LotFrontage featrure  
df.LotFrontage.describe()

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

# Replacing the NaN value with mean for LotFrontage feature   
df.LotFrontage.fillna((df.LotFrontage.mean()), inplace = True)

# Checking BsmtQual  
df.BsmtQual.value\_counts()

TA 517  
Gd 498  
Ex 94  
Fa 29  
Name: BsmtQual, dtype: int64

# Replacing the NA with No Basement for features BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2  
df.BsmtQual.fillna("No Basement", inplace = True)  
df.BsmtCond.fillna("No Basement", inplace = True)  
df.BsmtExposure.fillna("No Basement", inplace = True)  
df.BsmtFinType1.fillna("No Basement", inplace = True)  
df.BsmtFinType2.fillna("No Basement", inplace = True)

# Dropping the records which are having NA for features 'MasVnrType', 'MasVnrArea', 'Electrical'  
df = df[pd.notnull(df['MasVnrType'])]  
df = df[pd.notnull(df['MasVnrArea'])]  
df = df[pd.notnull(df['Electrical'])]

# Replacing the NA with No Garage for features GarageType, GarageFinish, GarageQual, GarageCond  
df.GarageType.fillna("No Garage", inplace = True)  
df.GarageFinish.fillna("No Garage", inplace = True)  
df.GarageQual.fillna("No Garage", inplace = True)  
df.GarageCond.fillna("No Garage", inplace = True)

# Replacing the NA value with mean for LotFrontage feature   
df.GarageYrBlt.fillna((df.GarageYrBlt.mean()), inplace = True)

# Replacing the NA value with mean for Age feature   
#df.Age.fillna((df.Age.mean()), inplace = True)

calc\_null(df)

MSSubClass 0.0  
MSZoning 0.0  
LotFrontage 0.0  
LotArea 0.0  
Street 0.0  
 ...   
MoSold 0.0  
YrSold 0.0  
SaleType 0.0  
SaleCondition 0.0  
SalePrice 0.0  
Length: 75, dtype: float64

#### 3. Checking the continues variables

# Continues(numerical) variables in the dataset  
df\_numeric = df.select\_dtypes(include=['float64', 'int64'])  
df\_numeric.head()

MSSubClass LotFrontage LotArea OverallQual OverallCond YearBuilt \  
0 120 70.98847 4928 6 5 1976   
1 20 95.00000 15865 8 6 1970   
2 60 92.00000 9920 7 5 1996   
3 20 105.00000 11751 6 6 1977   
4 20 70.98847 16635 6 7 1977   
  
 YearRemodAdd MasVnrArea BsmtFinSF1 BsmtFinSF2 ... WoodDeckSF \  
0 1976 0.0 120 0 ... 0   
1 1970 0.0 351 823 ... 81   
2 1997 0.0 862 0 ... 180   
3 1977 480.0 705 0 ... 0   
4 2000 126.0 1246 0 ... 240   
  
 OpenPorchSF EnclosedPorch 3SsnPorch ScreenPorch PoolArea MiscVal \  
0 205 0 0 0 0 0   
1 207 0 0 224 0 0   
2 130 0 0 0 0 0   
3 122 0 0 0 0 0   
4 0 0 0 0 0 0   
  
 MoSold YrSold SalePrice   
0 2 2007 128000   
1 10 2007 268000   
2 6 2007 269790   
3 1 2010 190000   
4 6 2009 215000   
  
[5 rows x 37 columns]

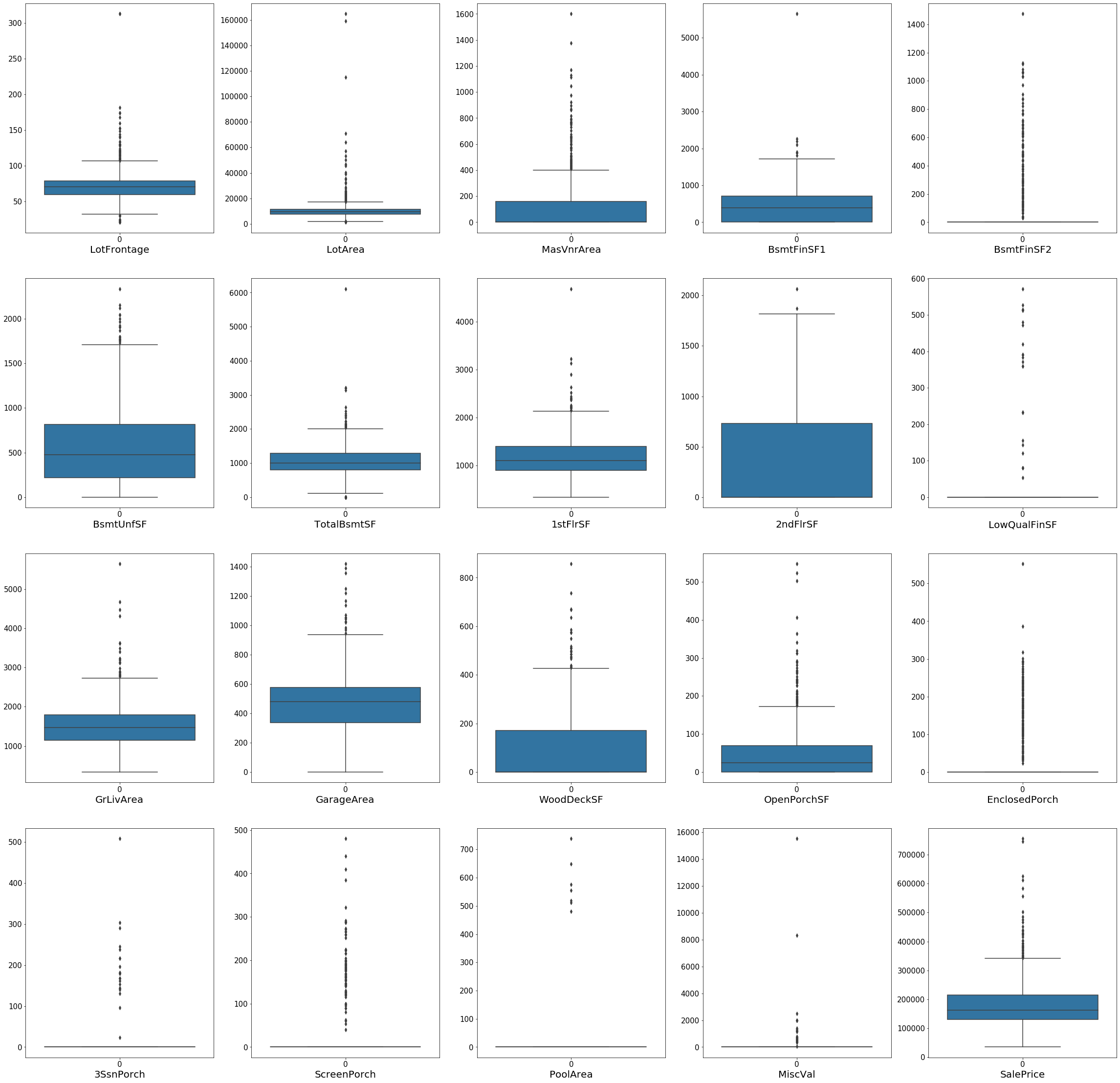
# From Data dictionary we see that there are few variables which are treated as numerical variables   
# but we can treat them as categorical variables. So we are droping those columns from df\_numeric DataFrame  
# dropping the columns we want to treat as categorical variables  
df\_numeric = df\_numeric.drop(['MSSubClass', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd','BsmtFullBath',  
 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr','KitchenAbvGr', 'TotRmsAbvGrd',  
 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'MoSold', 'YrSold'], axis=1)  
df\_numeric.head()

LotFrontage LotArea MasVnrArea BsmtFinSF1 BsmtFinSF2 BsmtUnfSF \  
0 70.98847 4928 0.0 120 0 958   
1 95.00000 15865 0.0 351 823 1043   
2 92.00000 9920 0.0 862 0 255   
3 105.00000 11751 480.0 705 0 1139   
4 70.98847 16635 126.0 1246 0 356   
  
 TotalBsmtSF 1stFlrSF 2ndFlrSF LowQualFinSF GrLivArea GarageArea \  
0 1078 958 0 0 958 440   
1 2217 2217 0 0 2217 621   
2 1117 1127 886 0 2013 455   
3 1844 1844 0 0 1844 546   
4 1602 1602 0 0 1602 529   
  
 WoodDeckSF OpenPorchSF EnclosedPorch 3SsnPorch ScreenPorch PoolArea \  
0 0 205 0 0 0 0   
1 81 207 0 0 224 0   
2 180 130 0 0 0 0   
3 0 122 0 0 0 0   
4 240 0 0 0 0 0   
  
 MiscVal SalePrice   
0 0 128000   
1 0 268000   
2 0 269790   
3 0 190000   
4 0 215000

numerical\_col=df\_numeric.columns  
numerical\_col

Index(['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',  
 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',  
 'GrLivArea', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch',  
 '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'SalePrice'],  
 dtype='object')

plt.figure(figsize=(40,50))  
for i in range(len(numerical\_col)):  
 plt.subplot(5,5,i+1)  
 sns.boxplot(data = df[numerical\_col[i]])  
 plt.xlabel(numerical\_col[i],fontdict={'fontsize':20})  
 plt.xticks(fontsize=15)  
 plt.yticks(fontsize=15)  
  
plt.show()

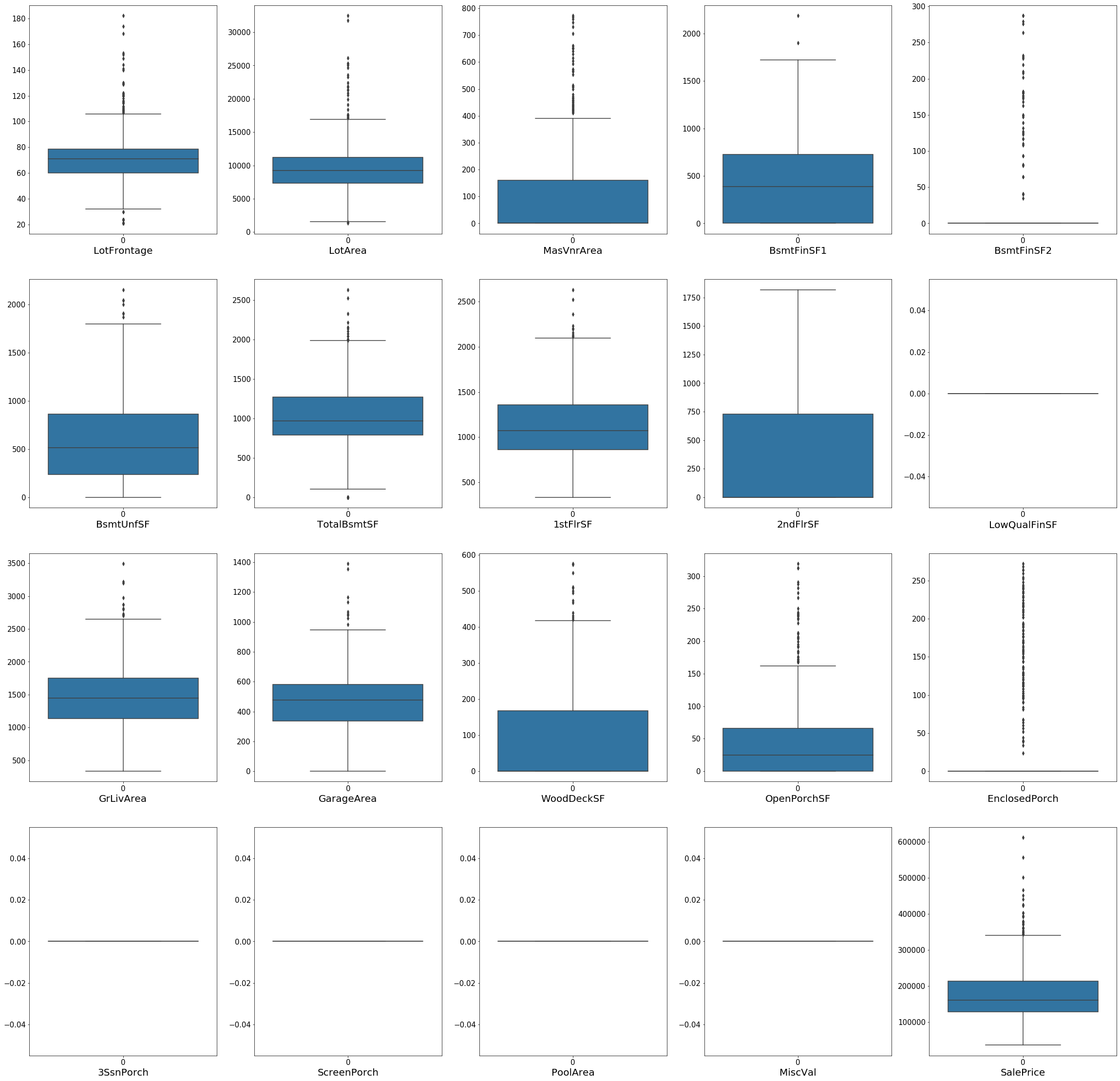


**So, we can see that there are outliers in out dataset.**

### Outlier Treatment

for i in range(len(numerical\_col)):  
 Q1=df[numerical\_col[i]].quantile(0.0)  
 Q3=df[numerical\_col[i]].quantile(0.90)  
 IQR=Q3-Q1  
 df=df[(df[numerical\_col[i]] >= Q1 - 1.5\*IQR) &   
 (df[numerical\_col[i]] <= Q3 + 1.5\*IQR)]

plt.figure(figsize=(40,50))  
for i in range(len(numerical\_col)):  
 plt.subplot(5,5,i+1)  
 sns.boxplot(data = df[numerical\_col[i]])  
 plt.xlabel(numerical\_col[i],fontdict={'fontsize':20})  
 plt.xticks(fontsize=15)  
 plt.yticks(fontsize=15)  
  
plt.show()



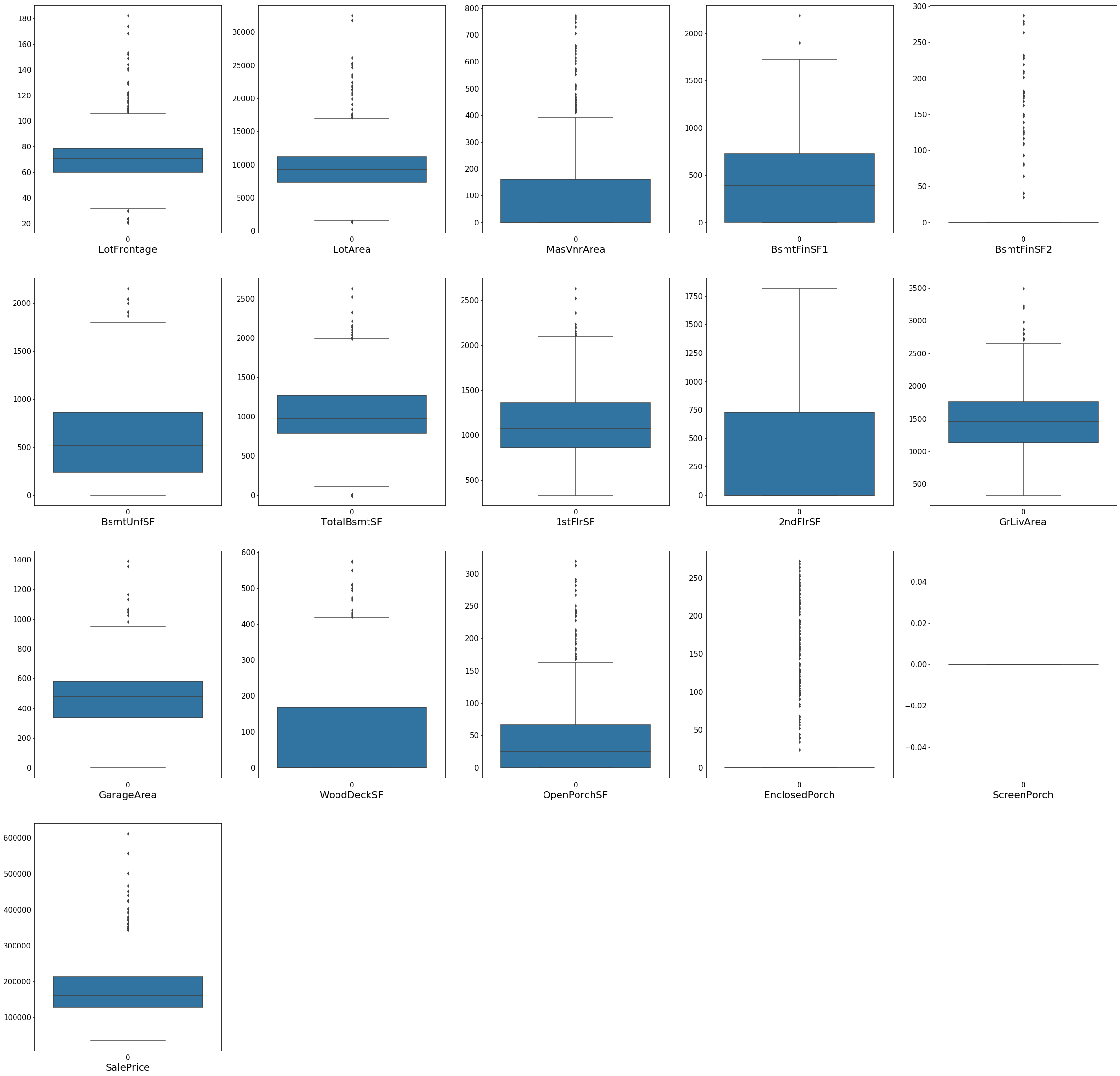
Observation: All Values for these 'LowQualFinSF','3SsnPorch','PoolArea','MiscVal' 4 variables are 0. So, will drop all these columns

df=df.drop(['LowQualFinSF','3SsnPorch','PoolArea','MiscVal'],axis=1)  
df\_numeric=df\_numeric.drop(['LowQualFinSF','3SsnPorch','PoolArea','MiscVal'],axis=1)

numerical\_col=df\_numeric.columns  
numerical\_col

Index(['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',  
 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',  
 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch',  
 'ScreenPorch', 'SalePrice'],  
 dtype='object')

plt.figure(figsize=(40,50))  
for i in range(len(numerical\_col)):  
 plt.subplot(5,5,i+1)  
 sns.boxplot(data = df[numerical\_col[i]])  
 plt.xlabel(numerical\_col[i],fontdict={'fontsize':20})  
 plt.xticks(fontsize=15)  
 plt.yticks(fontsize=15)  
  
plt.show()

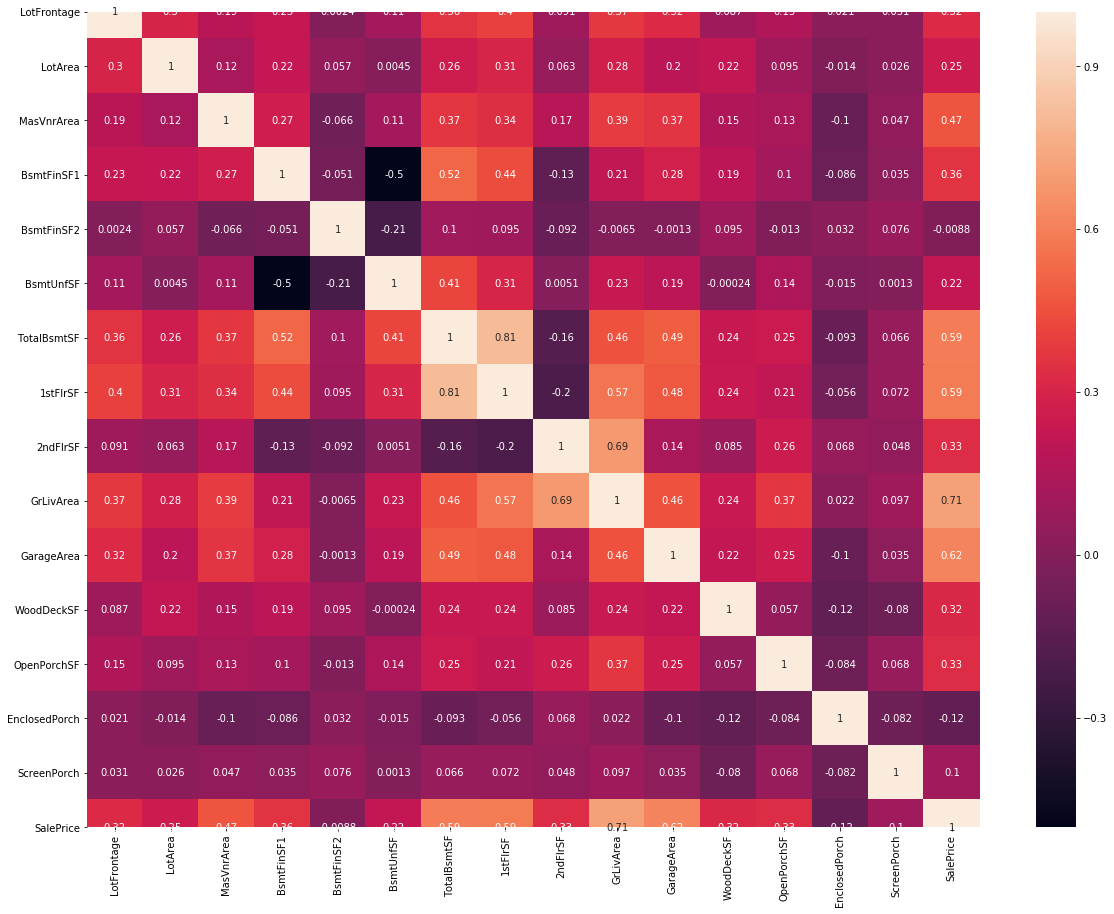


### Correlation

# Checking the correlation between numerical variables  
corr = df\_numeric.corr()  
corr

LotFrontage LotArea MasVnrArea BsmtFinSF1 BsmtFinSF2 \  
LotFrontage 1.000000 0.299527 0.189278 0.230480 0.002408   
LotArea 0.299527 1.000000 0.121448 0.220294 0.057240   
MasVnrArea 0.189278 0.121448 1.000000 0.267066 -0.065723   
BsmtFinSF1 0.230480 0.220294 0.267066 1.000000 -0.051336   
BsmtFinSF2 0.002408 0.057240 -0.065723 -0.051336 1.000000   
BsmtUnfSF 0.111278 0.004469 0.109850 -0.501103 -0.213945   
TotalBsmtSF 0.355325 0.256259 0.366833 0.516661 0.099884   
1stFlrSF 0.404127 0.307850 0.339938 0.442641 0.095433   
2ndFlrSF 0.091039 0.062575 0.173358 -0.126381 -0.092173   
GrLivArea 0.374241 0.278716 0.387891 0.214123 -0.006460   
GarageArea 0.322229 0.195753 0.365849 0.284618 -0.001338   
WoodDeckSF 0.087090 0.219025 0.151978 0.193530 0.094631   
OpenPorchSF 0.148923 0.095231 0.131850 0.103665 -0.012724   
EnclosedPorch 0.021346 -0.013749 -0.102321 -0.086273 0.032360   
ScreenPorch 0.030728 0.025720 0.046509 0.034703 0.076126   
SalePrice 0.321906 0.250421 0.466386 0.359530 -0.008848   
  
 BsmtUnfSF TotalBsmtSF 1stFlrSF 2ndFlrSF GrLivArea \  
LotFrontage 0.111278 0.355325 0.404127 0.091039 0.374241   
LotArea 0.004469 0.256259 0.307850 0.062575 0.278716   
MasVnrArea 0.109850 0.366833 0.339938 0.173358 0.387891   
BsmtFinSF1 -0.501103 0.516661 0.442641 -0.126381 0.214123   
BsmtFinSF2 -0.213945 0.099884 0.095433 -0.092173 -0.006460   
BsmtUnfSF 1.000000 0.414777 0.307542 0.005122 0.233777   
TotalBsmtSF 0.414777 1.000000 0.811702 -0.161228 0.459679   
1stFlrSF 0.307542 0.811702 1.000000 -0.201437 0.565718   
2ndFlrSF 0.005122 -0.161228 -0.201437 1.000000 0.688189   
GrLivArea 0.233777 0.459679 0.565718 0.688189 1.000000   
GarageArea 0.191817 0.492590 0.477209 0.137045 0.459506   
WoodDeckSF -0.000243 0.237353 0.239357 0.084608 0.243488   
OpenPorchSF 0.140097 0.246396 0.209865 0.255144 0.369480   
EnclosedPorch -0.014515 -0.093009 -0.056295 0.067981 0.022244   
ScreenPorch 0.001317 0.065922 0.071933 0.048261 0.096752   
SalePrice 0.217286 0.594103 0.588649 0.334605 0.709033   
  
 GarageArea WoodDeckSF OpenPorchSF EnclosedPorch \  
LotFrontage 0.322229 0.087090 0.148923 0.021346   
LotArea 0.195753 0.219025 0.095231 -0.013749   
MasVnrArea 0.365849 0.151978 0.131850 -0.102321   
BsmtFinSF1 0.284618 0.193530 0.103665 -0.086273   
BsmtFinSF2 -0.001338 0.094631 -0.012724 0.032360   
BsmtUnfSF 0.191817 -0.000243 0.140097 -0.014515   
TotalBsmtSF 0.492590 0.237353 0.246396 -0.093009   
1stFlrSF 0.477209 0.239357 0.209865 -0.056295   
2ndFlrSF 0.137045 0.084608 0.255144 0.067981   
GrLivArea 0.459506 0.243488 0.369480 0.022244   
GarageArea 1.000000 0.216564 0.251677 -0.099662   
WoodDeckSF 0.216564 1.000000 0.056703 -0.124573   
OpenPorchSF 0.251677 0.056703 1.000000 -0.084270   
EnclosedPorch -0.099662 -0.124573 -0.084270 1.000000   
ScreenPorch 0.034697 -0.080049 0.068126 -0.082374   
SalePrice 0.617842 0.315624 0.334060 -0.115045   
  
 ScreenPorch SalePrice   
LotFrontage 0.030728 0.321906   
LotArea 0.025720 0.250421   
MasVnrArea 0.046509 0.466386   
BsmtFinSF1 0.034703 0.359530   
BsmtFinSF2 0.076126 -0.008848   
BsmtUnfSF 0.001317 0.217286   
TotalBsmtSF 0.065922 0.594103   
1stFlrSF 0.071933 0.588649   
2ndFlrSF 0.048261 0.334605   
GrLivArea 0.096752 0.709033   
GarageArea 0.034697 0.617842   
WoodDeckSF -0.080049 0.315624   
OpenPorchSF 0.068126 0.334060   
EnclosedPorch -0.082374 -0.115045   
ScreenPorch 1.000000 0.102212   
SalePrice 0.102212 1.000000

# Plotting the Heatmap  
plt.figure(figsize=(20,15))  
sns.heatmap(corr, annot=True)  
plt.show()



numerical\_col=df\_numeric.columns  
numerical\_col

Index(['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',  
 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea',  
 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch',  
 'ScreenPorch', 'SalePrice'],  
 dtype='object')

#### 4. Checking the categorical variables

# Converting numerical ciolumns to categorical columns  
df.MSSubClass = df.MSSubClass.astype('object')  
df.OverallQual = df.OverallQual.astype('object')  
df.OverallCond = df.OverallCond.astype('object')  
df.BsmtFullBath = df.BsmtFullBath.astype('object')  
df.BsmtHalfBath = df.BsmtHalfBath.astype('object')  
df.FullBath = df.FullBath.astype('object')  
df.HalfBath = df.HalfBath.astype('object')  
df.BedroomAbvGr = df.BedroomAbvGr.astype('object')  
df.KitchenAbvGr = df.KitchenAbvGr.astype('object')  
df.TotRmsAbvGrd = df.TotRmsAbvGrd.astype('object')  
df.Fireplaces = df.Fireplaces.astype('object')  
df.GarageCars = df.GarageCars.astype('object')

# Categorical variables in the dataset  
df\_cat = df.select\_dtypes(include=['object'])  
df\_cat.columns

Index(['MSSubClass', 'MSZoning', 'Street', 'LotShape', 'LandContour',  
 'Utilities', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1',  
 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual', 'OverallCond',  
 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',  
 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond',  
 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating', 'HeatingQC',  
 'CentralAir', 'Electrical', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',  
 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',  
 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageType',  
 'GarageFinish', 'GarageCars', 'GarageQual', 'GarageCond', 'PavedDrive',  
 'SaleType', 'SaleCondition'],  
 dtype='object')

for col in df\_cat.columns:  
 print()  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ",col," \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print(round((df[col].value\_counts()/df.shape[0])\*100,2))

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MSSubClass \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
20 35.84  
60 21.57  
50 9.10  
120 6.07  
70 4.72  
30 4.72  
160 4.49  
90 4.27  
80 2.70  
190 2.25  
85 1.80  
45 1.12  
75 0.67  
180 0.56  
40 0.11  
Name: MSSubClass, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MSZoning \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
RL 78.20  
RM 15.06  
FV 4.72  
RH 1.57  
C (all) 0.45  
Name: MSZoning, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Street \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Pave 99.89  
Grvl 0.11  
Name: Street, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LotShape \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Reg 65.39  
IR1 32.36  
IR2 1.69  
IR3 0.56  
Name: LotShape, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LandContour \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Lvl 90.79  
Bnk 3.82  
HLS 3.71  
Low 1.69  
Name: LandContour, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Utilities \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
AllPub 100.0  
Name: Utilities, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LotConfig \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Inside 72.58  
Corner 18.65  
CulDSac 5.51  
FR2 3.03  
FR3 0.22  
Name: LotConfig, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LandSlope \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Gtl 95.96  
Mod 3.60  
Sev 0.45  
Name: LandSlope, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Neighborhood \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
NAmes 14.04  
CollgCr 11.80  
OldTown 7.53  
Edwards 7.08  
Somerst 6.40  
Gilbert 5.96  
NridgHt 5.39  
NWAmes 4.94  
Sawyer 4.83  
SawyerW 4.61  
BrkSide 4.38  
Crawfor 3.15  
NoRidge 3.03  
Mitchel 2.92  
IDOTRR 2.47  
Timber 2.36  
Blmngtn 1.69  
StoneBr 1.69  
SWISU 1.57  
BrDale 1.24  
ClearCr 0.90  
MeadowV 0.90  
NPkVill 0.56  
Veenker 0.45  
Blueste 0.11  
Name: Neighborhood, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Condition1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Norm 86.85  
Feedr 5.28  
Artery 3.03  
RRAn 1.69  
PosN 1.24  
RRAe 1.01  
PosA 0.45  
RRNn 0.34  
RRNe 0.11  
Name: Condition1, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Condition2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Norm 99.21  
Feedr 0.45  
Artery 0.22  
RRAn 0.11  
Name: Condition2, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BldgType \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1Fam 82.47  
TwnhsE 7.87  
Duplex 4.27  
Twnhs 3.03  
2fmCon 2.36  
Name: BldgType, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HouseStyle \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1Story 49.55  
2Story 32.70  
1.5Fin 9.44  
SFoyer 3.15  
SLvl 3.15  
1.5Unf 1.24  
2.5Unf 0.67  
2.5Fin 0.11  
Name: HouseStyle, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OverallQual \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 26.40  
7 23.60  
6 23.37  
8 12.70  
4 8.76  
9 2.70  
3 1.35  
10 0.56  
2 0.34  
1 0.22  
Name: OverallQual, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OverallCond \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
5 58.20  
6 16.63  
7 12.25  
8 5.39  
4 3.82  
3 2.02  
9 1.24  
2 0.34  
1 0.11  
Name: OverallCond, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RoofStyle \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Gable 80.90  
Hip 17.53  
Gambrel 0.90  
Mansard 0.45  
Shed 0.11  
Flat 0.11  
Name: RoofStyle, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RoofMatl \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
CompShg 99.21  
WdShake 0.34  
Tar&Grv 0.22  
Roll 0.11  
WdShngl 0.11  
Name: RoofMatl, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Exterior1st \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
VinylSd 38.31  
HdBoard 15.51  
MetalSd 15.28  
Wd Sdng 12.92  
Plywood 6.29  
BrkFace 3.03  
CemntBd 3.03  
WdShing 1.91  
AsbShng 1.80  
Stucco 1.57  
AsphShn 0.11  
BrkComm 0.11  
ImStucc 0.11  
Name: Exterior1st, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Exterior2nd \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
VinylSd 37.53  
MetalSd 14.94  
HdBoard 14.38  
Wd Sdng 11.91  
Plywood 8.54  
Wd Shng 3.03  
CmentBd 3.03  
Stucco 1.80  
AsbShng 1.69  
BrkFace 1.46  
ImStucc 0.67  
Brk Cmn 0.45  
AsphShn 0.34  
Stone 0.22  
Name: Exterior2nd, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MasVnrType \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
None 58.76  
BrkFace 31.35  
Stone 9.10  
BrkCmn 0.79  
Name: MasVnrType, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ExterQual \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
TA 60.11  
Gd 36.40  
Ex 2.70  
Fa 0.79  
Name: ExterQual, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ExterCond \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
TA 88.43  
Gd 9.10  
Fa 2.36  
Ex 0.11  
Name: ExterCond, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Foundation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
PConc 47.19  
CBlock 40.79  
BrkTil 9.55  
Slab 2.02  
Stone 0.34  
Wood 0.11  
Name: Foundation, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtQual \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Gd 45.06  
TA 41.80  
Ex 7.53  
No Basement 2.92  
Fa 2.70  
Name: BsmtQual, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtCond \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
TA 88.76  
Gd 4.83  
Fa 3.26  
No Basement 2.92  
Po 0.22  
Name: BsmtCond, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtExposure \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
No 65.96  
Av 16.29  
Mn 7.42  
Gd 7.30  
No Basement 3.03  
Name: BsmtExposure, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtFinType1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Unf 32.58  
GLQ 29.78  
ALQ 13.82  
BLQ 9.21  
Rec 8.43  
LwQ 3.26  
No Basement 2.92  
Name: BsmtFinType1, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtFinType2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Unf 92.02  
No Basement 2.92  
LwQ 2.13  
Rec 1.57  
BLQ 1.01  
ALQ 0.22  
GLQ 0.11  
Name: BsmtFinType2, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Heating \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
GasA 97.98  
GasW 0.90  
Grav 0.45  
Wall 0.45  
OthW 0.11  
Floor 0.11  
Name: Heating, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HeatingQC \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Ex 51.24  
TA 28.43  
Gd 16.85  
Fa 3.37  
Po 0.11  
Name: HeatingQC, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CentralAir \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Y 92.92  
N 7.08  
Name: CentralAir, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Electrical \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
SBrkr 91.01  
FuseA 6.63  
FuseF 2.02  
FuseP 0.22  
Mix 0.11  
Name: Electrical, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtFullBath \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 61.46  
1 37.42  
2 1.01  
3 0.11  
Name: BsmtFullBath, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BsmtHalfBath \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 95.96  
1 3.82  
2 0.22  
Name: BsmtHalfBath, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FullBath \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
2 54.27  
1 43.48  
3 1.57  
0 0.67  
Name: FullBath, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* HalfBath \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 62.36  
1 36.85  
2 0.79  
Name: HalfBath, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BedroomAbvGr \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
3 54.83  
2 25.17  
4 14.49  
1 3.37  
5 1.35  
6 0.56  
0 0.22  
Name: BedroomAbvGr, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KitchenAbvGr \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
1 94.61  
2 5.17  
3 0.11  
0 0.11  
Name: KitchenAbvGr, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KitchenQual \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
TA 48.76  
Gd 42.70  
Ex 6.07  
Fa 2.47  
Name: KitchenQual, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TotRmsAbvGrd \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
6 28.09  
7 23.37  
5 18.54  
8 12.47  
4 6.07  
9 5.96  
10 3.15  
3 1.12  
11 1.01  
12 0.11  
2 0.11  
Name: TotRmsAbvGrd, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functional \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Typ 94.04  
Min2 2.36  
Min1 1.69  
Maj1 0.79  
Mod 0.79  
Maj2 0.34  
Name: Functional, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Fireplaces \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
0 50.79  
1 43.15  
2 5.84  
3 0.22  
Name: Fireplaces, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GarageType \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Attchd 58.65  
Detchd 27.19  
No Garage 6.07  
BuiltIn 5.96  
Basment 1.12  
CarPort 0.56  
2Types 0.45  
Name: GarageType, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GarageFinish \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Unf 41.57  
RFn 28.99  
Fin 23.37  
No Garage 6.07  
Name: GarageFinish, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GarageCars \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
2 56.18  
1 24.16  
3 13.26  
0 6.07  
4 0.34  
Name: GarageCars, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GarageQual \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
TA 89.33  
No Garage 6.07  
Fa 3.48  
Gd 0.79  
Po 0.22  
Ex 0.11  
Name: GarageQual, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GarageCond \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
TA 90.11  
No Garage 6.07  
Fa 2.70  
Gd 0.56  
Po 0.45  
Ex 0.11  
Name: GarageCond, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* PavedDrive \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Y 91.24  
N 6.85  
P 1.91  
Name: PavedDrive, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SaleType \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
WD 85.62  
New 9.55  
COD 2.81  
ConLD 0.67  
ConLI 0.45  
Oth 0.34  
ConLw 0.22  
Con 0.22  
CWD 0.11  
Name: SaleType, dtype: float64  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SaleCondition \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Normal 80.79  
Partial 9.66  
Abnorml 6.29  
Family 1.80  
Alloca 1.01  
AdjLand 0.45  
Name: SaleCondition, dtype: float64

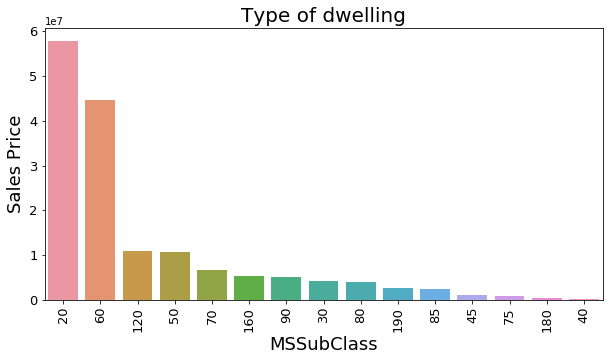
**Observation:**

* Street, Utilities, Condition2, RoofMatl, Hea?ting columns has approx. 98% or more records containing same values. So, will drop those columns.
* LandSlope, BsmtHalfBath, KitchenAbvGr contain same value for approx 95% records.

# Dropping Street, Utilities, Condition2, RoofMatl, Heating  
df = df.drop(['Street', 'Utilities', 'Condition2', 'RoofMatl', 'Heating','LandSlope','BsmtHalfBath','KitchenAbvGr'], axis=1)  
df.shape

(890, 63)

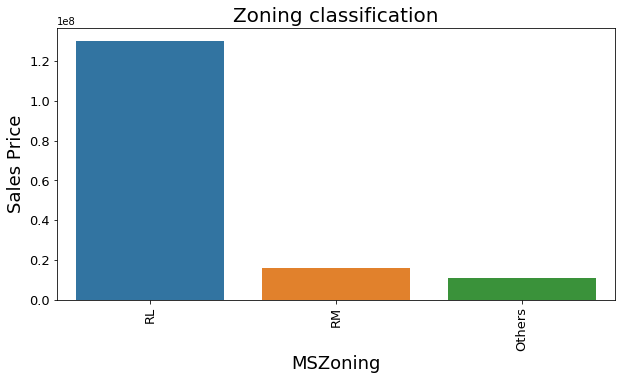
colName = 'MSSubClass'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('MSSubClass',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('Type of dwelling',fontsize=20)  
plt.show()



# Checking MSZoning columns  
frq = df.MSZoning.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<5).dropna()  
frq  
df.MSZoning = df.MSZoning.replace(frq.index,'Others')   
df.MSZoning.value\_counts()/df.shape[0]\*100

RL 78.202247  
RM 15.056180  
Others 6.741573  
Name: MSZoning, dtype: float64

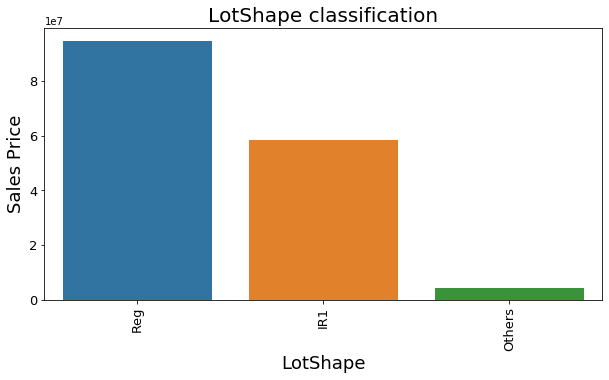
colName = 'MSZoning'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('MSZoning',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('Zoning classification',fontsize=20)  
plt.show()



# Checking LotShape columns  
frq = df.LotShape.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<5).dropna()  
frq  
df.LotShape = df.LotShape.replace(frq.index,'Others')   
df.LotShape.value\_counts()/df.shape[0]\*100

Reg 65.393258  
IR1 32.359551  
Others 2.247191  
Name: LotShape, dtype: float64

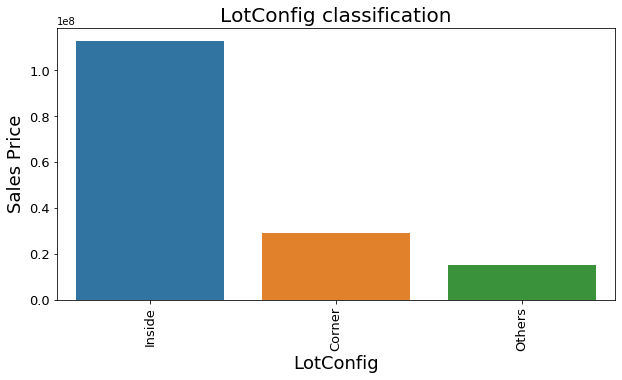
colName = 'LotShape'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('LotShape',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('LotShape classification',fontsize=20)  
plt.show()



# Checking LotConfig columns  
frq = df.LotConfig.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<10).dropna()  
frq  
df.LotConfig = df.LotConfig.replace(frq.index,'Others')   
df.LotConfig.value\_counts()/df.shape[0]\*100

Inside 72.584270  
Corner 18.651685  
Others 8.764045  
Name: LotConfig, dtype: float64

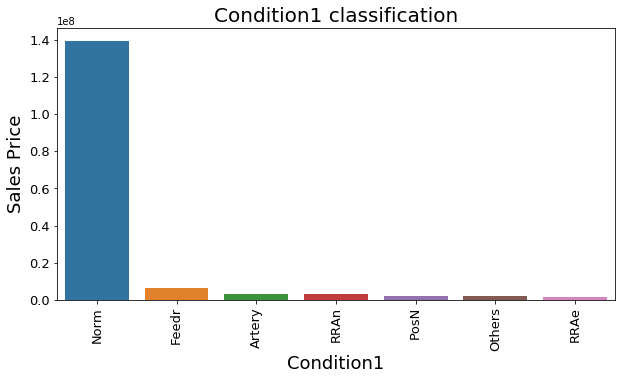
colName = 'LotConfig'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('LotConfig',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('LotConfig classification',fontsize=20)  
plt.show()



# Checking Condition1 columns  
frq = df.Condition1.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<1).dropna()  
frq  
df.Condition1 = df.Condition1.replace(frq.index,'Others')   
df.Condition1.value\_counts()/df.shape[0]\*100

Norm 86.853933  
Feedr 5.280899  
Artery 3.033708  
RRAn 1.685393  
PosN 1.235955  
RRAe 1.011236  
Others 0.898876  
Name: Condition1, dtype: float64

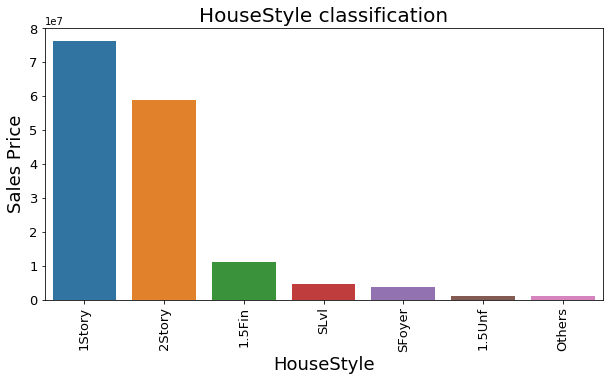
colName = 'Condition1'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('Condition1',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('Condition1 classification',fontsize=20)  
plt.show()



# Checking HouseStyle columns  
frq = df.HouseStyle.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<1).dropna()  
frq  
df.HouseStyle = df.HouseStyle.replace(frq.index,'Others')   
df.HouseStyle.value\_counts()/df.shape[0]\*100

1Story 49.550562  
2Story 32.696629  
1.5Fin 9.438202  
SFoyer 3.146067  
SLvl 3.146067  
1.5Unf 1.235955  
Others 0.786517  
Name: HouseStyle, dtype: float64

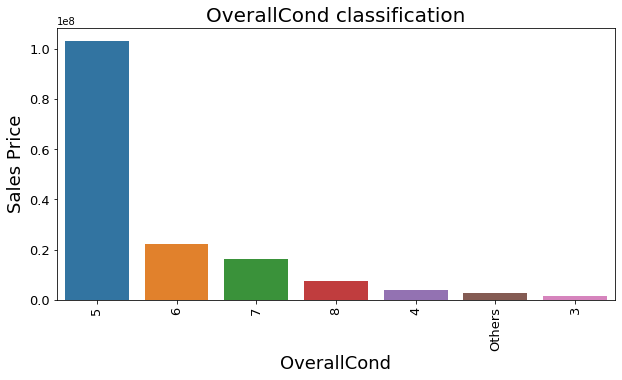
colName = 'HouseStyle'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('HouseStyle',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('HouseStyle classification',fontsize=20)  
plt.show()



# Checking OverallCond columns  
frq = df.OverallCond.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<2).dropna()  
frq  
df.OverallCond = df.OverallCond.replace(frq.index,'Others')   
df.OverallCond.value\_counts()/df.shape[0]\*100

5 58.202247  
6 16.629213  
7 12.247191  
8 5.393258  
4 3.820225  
3 2.022472  
Others 1.685393  
Name: OverallCond, dtype: float64

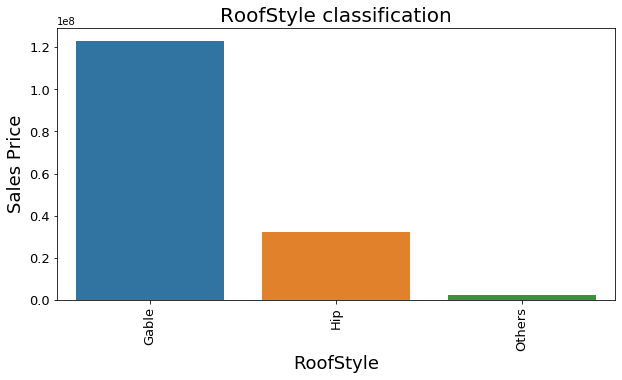
colName = 'OverallCond'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('OverallCond',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('OverallCond classification',fontsize=20)  
plt.show()



# Checking RoofStyle columns  
frq = df.RoofStyle.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<1).dropna()  
frq  
df.RoofStyle = df.RoofStyle.replace(frq.index,'Others')   
df.RoofStyle.value\_counts()/df.shape[0]\*100

Gable 80.898876  
Hip 17.528090  
Others 1.573034  
Name: RoofStyle, dtype: float64

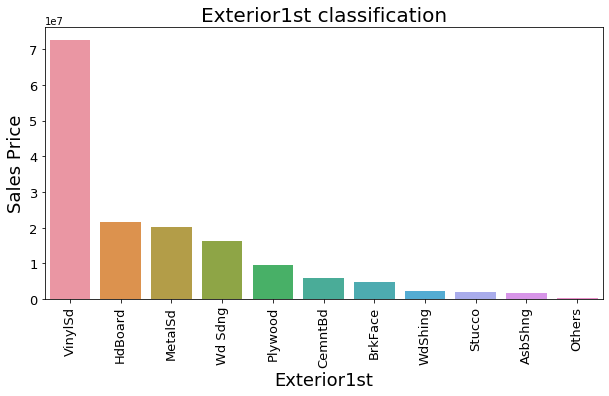
colName = 'RoofStyle'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('RoofStyle',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('RoofStyle classification',fontsize=20)  
plt.show()



# Checking Exterior1st columns  
frq = df.Exterior1st.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<0.15).dropna()  
frq  
df.Exterior1st = df.Exterior1st.replace(frq.index,'Others')   
df.Exterior1st.value\_counts()/df.shape[0]\*100

VinylSd 38.314607  
HdBoard 15.505618  
MetalSd 15.280899  
Wd Sdng 12.921348  
Plywood 6.292135  
BrkFace 3.033708  
CemntBd 3.033708  
WdShing 1.910112  
AsbShng 1.797753  
Stucco 1.573034  
Others 0.337079  
Name: Exterior1st, dtype: float64

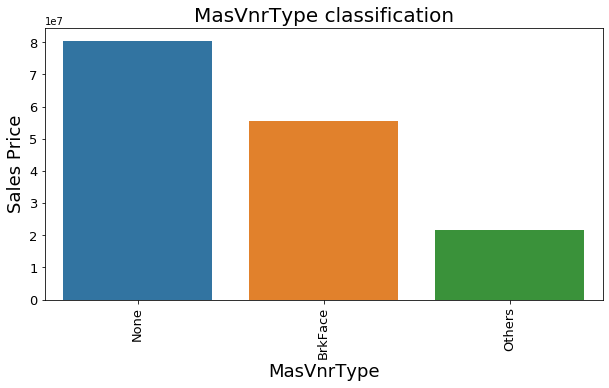
colName = 'Exterior1st'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('Exterior1st',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('Exterior1st classification',fontsize=20)  
plt.show()



# Checking MasVnrType columns  
frq = df.MasVnrType.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<10).dropna()  
frq  
df.MasVnrType = df.MasVnrType.replace(frq.index,'Others')   
df.MasVnrType.value\_counts()/df.shape[0]\*100

None 58.764045  
BrkFace 31.348315  
Others 9.887640  
Name: MasVnrType, dtype: float64

colName = 'MasVnrType'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('MasVnrType',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('MasVnrType classification',fontsize=20)  
plt.show()



# Checking ExterQual columns ExterCond  
frq = df.ExterQual.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<5).dropna()  
frq  
df.ExterQual = df.ExterQual.replace(frq.index,'Others')   
df.ExterQual.value\_counts()/df.shape[0]\*100

TA 60.112360  
Gd 36.404494  
Others 3.483146  
Name: ExterQual, dtype: float64

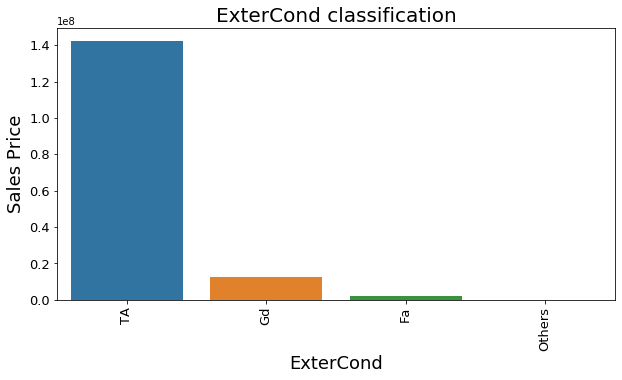
colName = 'ExterQual'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('ExterQual',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('ExterQual classification',fontsize=20)  
plt.show()



# Checking ExterCond columns Foundation  
frq = df.ExterCond.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<2).dropna()  
frq  
df.ExterCond = df.ExterCond.replace(frq.index,'Others')   
df.ExterCond.value\_counts()/df.shape[0]\*100

TA 88.426966  
Gd 9.101124  
Fa 2.359551  
Others 0.112360  
Name: ExterCond, dtype: float64

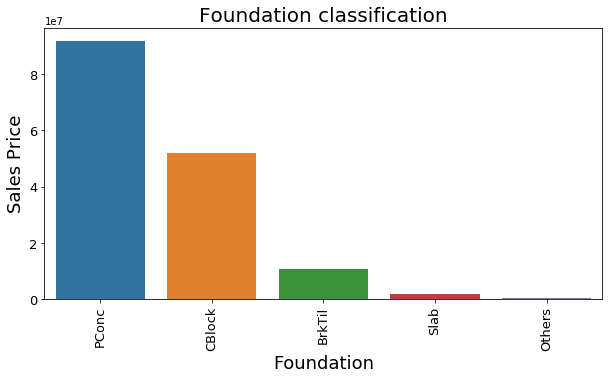
colName = 'ExterCond'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('ExterCond',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('ExterCond classification',fontsize=20)  
plt.show()



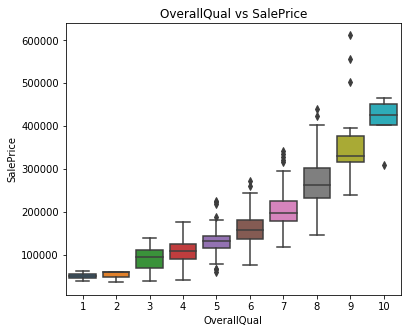
# Checking Foundation columns   
frq = df.Foundation.value\_counts(normalize=True,dropna=False)\*100  
frq = frq.where(frq<2).dropna()  
frq  
df.Foundation = df.Foundation.replace(frq.index,'Others')   
df.Foundation.value\_counts()/df.shape[0]\*100

PConc 47.191011  
CBlock 40.786517  
BrkTil 9.550562  
Slab 2.022472  
Others 0.449438  
Name: Foundation, dtype: float64

colName = 'Foundation'  
grouped\_df = df.groupby(colName)[[colName,'SalePrice']].sum().reset\_index().sort\_values('SalePrice',ascending=False)  
plt.figure(figsize=(10,5))  
plt\_obj=sns.barplot(x=grouped\_df[colName],y=grouped\_df.SalePrice,order=grouped\_df[colName])  
plt\_obj.set\_xticklabels(plt\_obj.get\_xticklabels(),rotation=90,fontsize=13)  
plt.yticks(fontsize=13)  
plt.xlabel('Foundation',fontsize = 18)  
plt.ylabel('Sales Price',fontsize = 18)  
plt.title('Foundation classification',fontsize=20)  
plt.show()



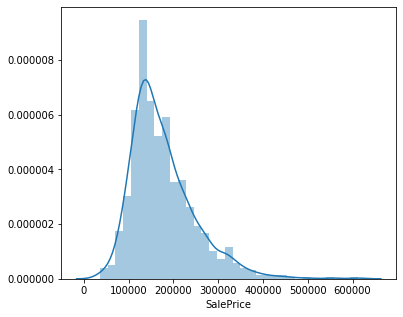
# Plotting the boxplot for OverallQual vs SalePrice  
plt.figure(figsize=(6,5))  
sns.boxplot(x="OverallQual", y="SalePrice", data=df)  
plt.title("OverallQual vs SalePrice")  
plt.show()



Observation: As OverallQual increases, the SalePrice also increases.

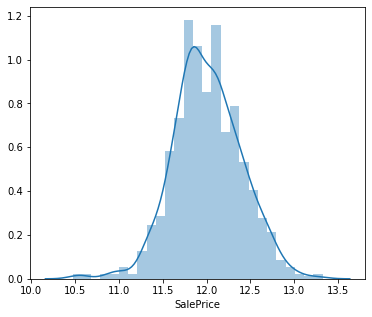
#### 5. Checking the target variable

plt.figure(figsize=(6,5))  
sns.distplot(df.SalePrice)  
plt.show()



* Data is not normalized
* So, we are doing log transformation

plt.figure(figsize=(6,5))  
sns.distplot(np.log(df.SalePrice))  
plt.show()



## Creating the dummy variables

# Taking all categorical variables  
df\_cat = df.select\_dtypes(include=['object'])  
df\_cat.head()

MSSubClass MSZoning LotShape LandContour LotConfig Neighborhood Condition1 \  
0 120 RL IR1 Lvl Inside NPkVill Norm   
2 60 RL IR1 Lvl Others NoRidge Norm   
3 20 RL IR1 Lvl Inside NWAmes Norm   
4 20 RL IR1 Lvl Others NWAmes Norm   
5 60 RL IR1 Lvl Inside Gilbert Norm   
  
 BldgType HouseStyle OverallQual ... Functional Fireplaces GarageType \  
0 TwnhsE 1Story 6 ... Typ 1 Attchd   
2 1Fam 2Story 7 ... Typ 1 Attchd   
3 1Fam 1Story 6 ... Typ 1 Attchd   
4 1Fam 1Story 6 ... Typ 1 Attchd   
5 1Fam 2Story 7 ... Typ 1 BuiltIn   
  
 GarageFinish GarageCars GarageQual GarageCond PavedDrive SaleType \  
0 RFn 2 TA TA Y WD   
2 Unf 2 TA TA Y WD   
3 RFn 2 TA TA Y COD   
4 Fin 2 TA TA Y WD   
5 Fin 3 TA TA Y New   
  
 SaleCondition   
0 Normal   
2 Normal   
3 Normal   
4 Normal   
5 Partial   
  
[5 rows x 42 columns]

# convert into dummies  
df\_dummies = pd.get\_dummies(df\_cat, drop\_first=True)  
df\_dummies.head()

MSSubClass\_30 MSSubClass\_40 MSSubClass\_45 MSSubClass\_50 MSSubClass\_60 \  
0 0 0 0 0 0   
2 0 0 0 0 1   
3 0 0 0 0 0   
4 0 0 0 0 0   
5 0 0 0 0 1   
  
 MSSubClass\_70 MSSubClass\_75 MSSubClass\_80 MSSubClass\_85 MSSubClass\_90 \  
0 0 0 0 0 0   
2 0 0 0 0 0   
3 0 0 0 0 0   
4 0 0 0 0 0   
5 0 0 0 0 0   
  
 ... SaleType\_ConLI SaleType\_ConLw SaleType\_New SaleType\_Oth \  
0 ... 0 0 0 0   
2 ... 0 0 0 0   
3 ... 0 0 0 0   
4 ... 0 0 0 0   
5 ... 0 0 1 0   
  
 SaleType\_WD SaleCondition\_AdjLand SaleCondition\_Alloca \  
0 1 0 0   
2 1 0 0   
3 0 0 0   
4 1 0 0   
5 0 0 0   
  
 SaleCondition\_Family SaleCondition\_Normal SaleCondition\_Partial   
0 0 1 0   
2 0 1 0   
3 0 1 0   
4 0 1 0   
5 0 0 1   
  
[5 rows x 220 columns]

# Drop categorical variables   
df=df.drop(list(df\_cat.columns), axis=1)

# concat dummy variables with df  
df = pd.concat([df, df\_dummies], axis=1)  
df.shape

(890, 241)

## Splitting the data into Train and Test data set

#y = np.log(df.SalePrice)  
#X = df.drop("SalePrice",1)

# split into train and test  
#  
#X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7,test\_size = 0.3, random\_state=100)

from sklearn.model\_selection import train\_test\_split  
df\_train, df\_test = train\_test\_split(df,train\_size=0.7,  
 test\_size = 0.3, random\_state=100)

## Scalling the data sets

# scaling the features  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
df\_train[numerical\_col] = scaler.fit\_transform(df\_train[numerical\_col])  
df\_test[numerical\_col] = scaler.transform(df\_test[numerical\_col])  
X\_train = df\_train.drop('SalePrice',axis=1)  
y\_train = df\_train.SalePrice  
X\_test = df\_test.drop('SalePrice',axis=1)  
y\_test = df\_test.SalePrice

## Linear Regression

# Instantiate  
lm = LinearRegression()  
  
# Fit a line  
lm.fit(X\_train, y\_train)

LinearRegression()

# Print the coefficients and intercept  
print(lm.intercept\_)  
print(lm.coef\_)

-20.906318880186856  
[ 3.96484055e-03 7.40937353e-02 7.53641449e-03 2.10877000e-03  
 7.06111921e-03 1.35360861e-01 -1.17204790e-02 -3.39666147e-03  
 1.41856073e-01 5.07697969e-02 1.74162030e-01 2.00427947e-01  
 -6.03379301e-04 1.59870266e-02 4.94384947e-02 3.31808736e-02  
 3.75693894e-02 -1.08801856e-14 9.10286899e-03 1.08974549e-03  
 -6.24393599e-03 1.00752739e-14 1.33289758e-02 -1.24343750e-01  
 -7.03767653e-02 2.33379743e-02 2.28294507e-01 -2.31982245e-01  
 4.75401115e-02 -1.13601091e-01 -1.33854455e-01 -2.57448450e-01  
 -1.89536793e-02 1.63948420e-02 -1.62041624e-01 -2.23515782e-01  
 -1.04721633e-01 4.39686743e-03 2.39745762e-03 -2.28714362e-01  
 -5.26656262e-02 1.09357288e-02 7.28406445e-02 -1.29699359e-01  
 1.76412978e-01 -2.31264400e-01 -6.93535956e-02 -2.01868491e-01  
 -9.86491147e-03 -3.48688159e-01 -2.55727871e-01 -2.76146473e-01  
 1.66607596e-01 -3.30816092e-01 -3.07388574e-01 1.30608531e-01  
 -2.20768554e-01 3.04910513e-02 8.55457484e-02 -3.30071569e-01  
 -4.24032550e-01 -2.24351609e-01 -2.26579959e-01 -1.87800694e-01  
 5.40686792e-01 -2.75807713e-01 -2.78344451e-01 5.02144126e-02  
 1.00076988e-01 -3.02814120e-02 1.79565730e-01 -3.86204301e-01  
 1.33755365e-01 -1.77639132e-01 -1.13601091e-01 -2.79653147e-01  
 -1.30603437e-01 1.33289758e-02 -3.54729699e-02 -8.84507819e-02  
 2.28294507e-01 -9.84635735e-02 1.97640597e-01 1.14150233e-01  
 -1.94088376e-01 -1.86073872e-01 -1.55319984e-01 -7.37094394e-02  
 2.11544587e-02 2.31705874e-01 4.32529776e-01 1.51371654e+00  
 2.76001914e-01 2.73669555e-01 4.10517891e-01 4.49110720e-01  
 5.13281518e-01 6.21340787e-01 4.73924039e-03 1.31287901e-01  
 -1.60814968e-02 -6.16482481e-01 -2.72017865e-01 -3.30614976e-01  
 -4.39030967e-01 -2.47565867e-01 -2.91697659e-01 -3.13301988e-01  
 -2.89671331e-01 -2.14429943e-01 2.81884979e-01 2.86773503e-01  
 2.80932545e-01 3.61813956e-01 1.69205851e-01 1.58723031e-01  
 3.51821246e-01 1.53561385e-01 1.65676288e-01 3.00145499e-01  
 2.94951273e-01 3.18552296e-01 1.98714497e-01 1.08367114e-02  
 3.65338954e-02 2.45768946e-02 -3.45922537e-02 -1.83240510e-01  
 -2.51400902e-01 -1.40007903e-01 7.76572920e-03 -1.17073858e-01  
 8.16001222e-02 -3.90989416e-02 8.68296167e-02 -1.03892098e-01  
 1.12728336e-01 -5.88207431e-02 1.10915064e-01 1.12728336e-01  
 6.36314329e-03 1.88091492e-01 2.37480033e-01 9.62758533e-02  
 6.47365218e-02 -8.34682920e-02 -2.81246821e-02 4.46890325e-02  
 -6.62813045e-02 1.12728336e-01 -2.09708927e-02 1.63188973e-02  
 2.31103901e-02 2.78935189e-01 -1.44232121e-01 1.12728336e-01  
 -1.79208323e-01 -1.07526618e-01 5.23899011e-02 -4.08666038e-03  
 3.80665812e-01 1.92239477e-03 9.31323744e-02 -9.63188486e-02  
 -2.16776913e-01 6.36314329e-03 -3.77861910e-02 3.44172701e-02  
 2.25597616e-01 -8.88178420e-16 4.65361594e-01 4.60311347e-01  
 5.40607079e-01 1.10908480e-02 1.13929318e-01 2.39671579e-01  
 1.29688921e-01 2.36804220e-02 -2.92976945e-03 -1.42570020e-01  
 -1.48424413e-01 -1.74273806e-01 -1.80420414e-01 -1.93795649e-01  
 1.48932913e-01 1.54998161e-01 1.58067509e-01 1.88058755e-01  
 2.30436755e-01 1.76986121e-01 3.21992856e-01 2.29600074e-01  
 3.29742319e-02 6.20178348e-02 -5.29239215e-01 -2.27206781e-01  
 5.07669868e-03 -1.69138346e-01 6.16341663e-02 2.01000560e-02  
 1.22130229e-01 -2.92730021e-02 6.17558988e-01 4.72280192e-01  
 5.87363530e-01 4.81790109e-01 7.14923729e-01 3.75465588e-02  
 3.75465588e-02 -6.01054271e-02 -4.47292291e-02 -1.19611039e-01  
 -1.08882491e-01 5.53070615e-02 1.35639909e-01 -2.30761900e-01  
 -2.32765109e-01 3.75465588e-02 -2.10413770e-01 -2.42363616e-01  
 -2.07471976e-01 -2.84848152e-01 3.75465588e-02 -2.92524110e-01  
 -1.31460157e-01 -4.37167176e-02 3.73280874e-02 2.84154982e-01  
 1.61754439e-01 -4.03826710e-03 -1.00934511e-03 -3.60424445e-02  
 7.23129750e-02 -2.52968880e-01 -4.54897000e-02 2.19889874e-01  
 2.25428312e-01 1.12382029e-01 1.50112070e-01 1.58588853e-01]

y\_pred\_train = lm.predict(X\_train)  
y\_pred\_test = lm.predict(X\_test)  
  
metric = []  
r2\_train\_lr = r2\_score(y\_train, y\_pred\_train)  
print(r2\_train\_lr)  
metric.append(r2\_train\_lr)  
  
r2\_test\_lr = r2\_score(y\_test, y\_pred\_test)  
print(r2\_test\_lr)  
metric.append(r2\_test\_lr)  
  
rss1\_lr = np.sum(np.square(y\_train - y\_pred\_train))  
print(rss1\_lr)  
metric.append(rss1\_lr)  
  
rss2\_lr = np.sum(np.square(y\_test - y\_pred\_test))  
print(rss2\_lr)  
metric.append(rss2\_lr)  
  
mse\_train\_lr = mean\_squared\_error(y\_train, y\_pred\_train)  
print(mse\_train\_lr)  
metric.append(mse\_train\_lr\*\*0.5)  
  
mse\_test\_lr = mean\_squared\_error(y\_test, y\_pred\_test)  
print(mse\_test\_lr\*\*0.5)  
metric.append(mse\_test\_lr\*\*0.5)

0.9583889623429703  
0.8655013582924591  
25.923676460329514  
43.31460623179854  
0.04161103765702972  
0.4027741173811995

## Ridge Regression

# list of alphas to tune - if value too high it will lead to underfitting, if it is too low,   
# it will not handle the overfitting  
params = {'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1,   
 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,   
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50, 100, 500, 1000 ]}  
  
ridge = Ridge()  
  
# cross validation  
folds = 5  
model\_cv = GridSearchCV(estimator = ridge,   
 param\_grid = params,   
 scoring= 'neg\_mean\_absolute\_error',   
 cv = folds,   
 return\_train\_score=True,  
 verbose = 1)   
model\_cv.fit(X\_train, y\_train)

Fitting 5 folds for each of 28 candidates, totalling 140 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.  
[Parallel(n\_jobs=1)]: Done 140 out of 140 | elapsed: 1.9s finished

GridSearchCV(cv=5, estimator=Ridge(),  
 param\_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,  
 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,  
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,  
 100, 500, 1000]},  
 return\_train\_score=True, scoring='neg\_mean\_absolute\_error',  
 verbose=1)

# Printing the best hyperparameter alpha  
print(model\_cv.best\_params\_['alpha'])

8.0

#Fitting Ridge model for alpha = 6 and printing coefficients which have been penalised  
alpha = model\_cv.best\_params\_['alpha']  
ridge = Ridge(alpha=alpha)  
  
ridge.fit(X\_train, y\_train)  
print(ridge.coef\_)

[ 2.42443197e-02 4.82070709e-02 6.48928190e-03 3.92204794e-03  
 2.44527112e-02 1.35454310e-01 -1.39237572e-02 -1.94863097e-02  
 1.23215212e-01 1.02256626e-01 1.28941244e-01 1.96498211e-01  
 -3.30677725e-04 4.42100098e-02 4.17082444e-02 3.89711991e-02  
 3.06438097e-02 0.00000000e+00 7.65062148e-03 9.12636053e-03  
 9.31169757e-03 0.00000000e+00 2.82250260e-02 -3.56897753e-02  
 2.10183119e-02 9.98113845e-02 6.25808208e-02 -3.96832578e-02  
 -1.34756885e-02 -8.95328326e-02 -2.92581087e-02 -1.14614644e-01  
 4.30914601e-03 -5.28719028e-02 7.96710015e-03 -7.35087580e-02  
 -4.12019873e-02 -9.77564744e-03 6.30189128e-02 -1.34276241e-01  
 -2.85714151e-02 1.14287344e-02 4.50100775e-02 -5.87549735e-03  
 6.06065112e-02 2.48657340e-02 3.50828740e-02 -7.31400492e-02  
 1.74808069e-01 -1.31551633e-01 -1.01448399e-01 -2.04832955e-03  
 -4.57286894e-02 -1.09518203e-01 -9.06482189e-02 5.59723393e-02  
 -4.40510093e-02 8.42237882e-02 1.21942845e-01 -7.05143164e-02  
 -5.63394997e-02 -5.52909767e-02 -7.41035673e-02 2.52973203e-02  
 3.32754276e-01 -8.66925595e-02 1.93509636e-02 1.37855926e-02  
 7.13532086e-02 -2.15538027e-02 5.67559252e-02 -1.15672556e-01  
 6.44203736e-02 -4.45516813e-02 -8.95328326e-02 -8.21233090e-02  
 -5.74402980e-02 2.82250260e-02 -4.54221672e-02 -2.46908761e-02  
 6.25808208e-02 -3.01194314e-02 1.53767171e-02 -3.52777289e-02  
 -7.78760847e-02 -1.40581574e-01 -1.32740242e-01 -8.97169338e-02  
 -1.10888479e-02 1.55514105e-01 2.67741936e-01 1.09049289e-01  
 -6.49646455e-02 -1.78323787e-02 1.10762950e-01 9.97971452e-02  
 1.27317640e-01 1.25530344e-01 2.46415546e-02 4.81218869e-02  
 1.65941388e-01 -2.65206716e-02 -5.06346252e-02 1.73276905e-02  
 -2.54814575e-02 -3.64543632e-02 6.12782980e-03 -1.13745842e-02  
 -1.38121191e-02 7.36886130e-03 -4.19639684e-03 1.89110949e-02  
 6.25124283e-02 -1.35307802e-02 -3.21964072e-02 -2.60570211e-02  
 -9.79893351e-03 -6.40192147e-02 9.47293968e-03 3.50414100e-02  
 2.08108954e-02 4.11234332e-02 -1.71392637e-02 2.89315003e-02  
 2.87196112e-02 9.66729084e-02 -6.66754195e-02 -5.49162609e-02  
 1.79363397e-02 -3.12007462e-03 -2.74916615e-02 -4.90373422e-02  
 4.57836052e-02 2.54958722e-02 -2.14156361e-02 -1.21462646e-01  
 3.23594471e-02 -8.85713957e-02 6.67064490e-02 3.23594471e-02  
 -7.85962048e-03 1.07562373e-01 1.70135678e-01 2.80613172e-02  
 7.65070232e-03 1.68853790e-02 -4.84529628e-02 4.96630260e-02  
 -7.60532698e-02 3.23594471e-02 -2.87525510e-02 1.81033815e-02  
 -3.01950278e-03 2.06496684e-02 8.00769201e-03 3.23594471e-02  
 -2.59763332e-02 -3.97370955e-02 -1.22613438e-04 -2.82608944e-02  
 8.23808269e-03 -2.77059182e-02 9.99495977e-02 -5.84497097e-02  
 3.25504519e-04 -7.85962048e-03 -2.73648936e-02 -1.79323541e-03  
 1.30360852e-02 0.00000000e+00 1.98785673e-02 -2.62010577e-02  
 5.57835257e-02 2.30757877e-02 -2.95270526e-04 9.50082952e-02  
 7.77215018e-02 1.78981051e-03 -2.75561713e-02 -5.03269812e-02  
 -8.05665382e-02 -7.78614431e-02 -1.36719310e-01 -1.41016852e-01  
 7.18782299e-03 -2.47770128e-02 -2.92293930e-02 -1.79188098e-02  
 8.93104847e-03 -2.49799660e-02 8.83465581e-02 6.50450854e-02  
 -2.69263355e-02 -6.55079550e-04 -7.71093780e-02 -3.28663267e-02  
 -1.98453963e-02 -6.17716752e-02 1.68190004e-01 3.39334562e-02  
 7.64112679e-02 2.03004443e-02 4.95394083e-02 -6.91345071e-02  
 5.44228782e-02 -5.11043417e-02 9.42628301e-02 2.74579645e-03  
 2.74579645e-03 -5.86917685e-02 -2.81194188e-02 -4.26334625e-02  
 -7.31693177e-02 1.15110029e-01 -2.05304496e-03 -1.79712869e-02  
 9.70173828e-03 2.74579645e-03 -7.53411596e-03 -6.88392886e-03  
 -4.66709366e-02 -7.35729688e-03 2.74579645e-03 -1.71782476e-02  
 4.85188876e-02 -1.23857813e-02 3.75190635e-02 3.27321815e-02  
 5.04955240e-02 2.21192932e-02 3.35238384e-02 -2.78618743e-02  
 9.29079034e-02 -5.67533890e-02 -7.23819843e-02 2.29840696e-03  
 5.43689390e-02 3.18168599e-02 1.28656143e-01 8.89447107e-02]

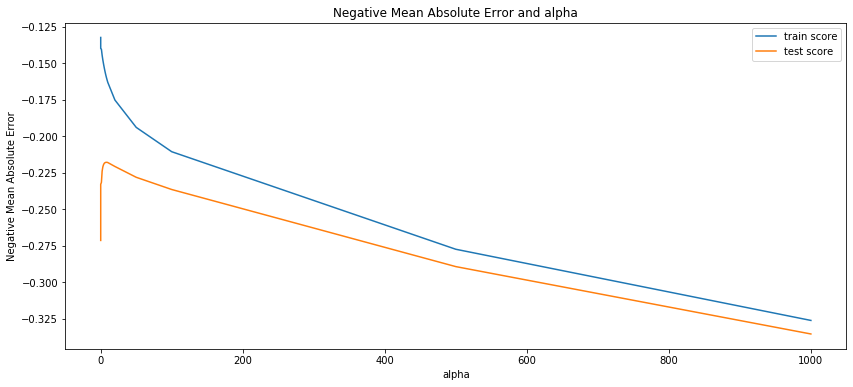
# Lets calculate some metrics such as R2 score, RSS and RMSE  
y\_pred\_train = ridge.predict(X\_train)  
y\_pred\_test = ridge.predict(X\_test)  
  
metric2 = []  
r2\_train\_lr = r2\_score(y\_train, y\_pred\_train)  
print("R2 score (Train): ",end="")  
print(r2\_train\_lr)  
metric2.append(r2\_train\_lr)  
  
r2\_test\_lr = r2\_score(y\_test, y\_pred\_test)  
print("R2 score (Test): ",end="")  
print(r2\_test\_lr)  
metric2.append(r2\_test\_lr)  
  
rss1\_lr = np.sum(np.square(y\_train - y\_pred\_train))  
print("RSS (Train): ",end="")  
print(rss1\_lr)  
metric2.append(rss1\_lr)  
  
rss2\_lr = np.sum(np.square(y\_test - y\_pred\_test))  
print("RSS (Test): ",end="")  
print(rss2\_lr)  
metric2.append(rss2\_lr)  
  
mse\_train\_lr = mean\_squared\_error(y\_train, y\_pred\_train)  
print("RMSE (Train): ",end="")  
print(mse\_train\_lr\*\*0.5)  
metric2.append(mse\_train\_lr\*\*0.5)  
  
mse\_test\_lr = mean\_squared\_error(y\_test, y\_pred\_test)  
print("RMSE (Test): ",end="")  
print(mse\_test\_lr\*\*0.5)  
metric2.append(mse\_test\_lr\*\*0.5)

R2 score (Train): 0.9457201321917775  
R2 score (Test): 0.887577391878362  
RSS (Train): 33.81635764452262  
RSS (Test): 36.20513144607857  
RMSE (Train): 0.23298040219774388  
RMSE (Test): 0.3682387041631283

ridge\_results = pd.DataFrame(model\_cv.cv\_results\_)  
ridge\_results = ridge\_results[ridge\_results['param\_alpha']<=1000]  
ridge\_results

mean\_fit\_time std\_fit\_time mean\_score\_time std\_score\_time param\_alpha \  
0 0.023541 3.362819e-02 0.003786 7.479942e-04 0.0001   
1 0.006395 1.498210e-03 0.003982 8.923400e-04 0.001   
2 0.006183 9.767468e-04 0.003391 4.885973e-04 0.01   
3 0.005191 4.017563e-04 0.003588 4.916487e-04 0.05   
4 0.005987 5.576345e-06 0.003587 7.996611e-04 0.1   
5 0.006183 7.460095e-04 0.003394 1.019279e-03 0.2   
6 0.006184 9.771651e-04 0.002393 4.888307e-04 0.3   
7 0.006188 1.465092e-03 0.003593 4.993817e-04 0.4   
8 0.006386 1.491811e-03 0.002989 6.316055e-04 0.5   
9 0.006184 9.780733e-04 0.003789 9.774092e-04 0.6   
10 0.006583 1.197934e-03 0.003191 7.462638e-04 0.7   
11 0.006784 1.464837e-03 0.003987 8.896768e-04 0.8   
12 0.006982 1.668924e-03 0.003191 3.992086e-04 0.9   
13 0.006982 1.261314e-03 0.003989 8.926031e-04 1   
14 0.005886 9.176726e-04 0.003989 8.919634e-04 2   
15 0.007580 1.017373e-03 0.003989 1.261351e-03 3   
16 0.005985 6.307516e-04 0.003590 4.885802e-04 4   
17 0.005984 5.917394e-07 0.003989 8.921235e-04 5   
18 0.006383 1.738576e-03 0.002992 1.360449e-06 6   
19 0.005586 4.879159e-04 0.003192 3.988769e-04 7   
20 0.006883 1.351944e-03 0.002792 1.163595e-03 8   
21 0.006383 7.978679e-04 0.003590 7.978439e-04 9   
22 0.006383 1.352958e-03 0.002992 4.101908e-07 10   
23 0.005586 1.197386e-03 0.003390 7.977724e-04 20   
24 0.006384 1.016691e-03 0.003193 3.982054e-04 50   
25 0.006186 7.467196e-04 0.003189 4.006075e-04 100   
26 0.006383 1.492757e-03 0.002992 6.302991e-04 500   
27 0.005386 4.890451e-04 0.002991 4.623108e-07 1000   
  
 params split0\_test\_score split1\_test\_score \  
0 {'alpha': 0.0001} -0.282712 -0.259252   
1 {'alpha': 0.001} -0.281631 -0.258940   
2 {'alpha': 0.01} -0.273697 -0.256854   
3 {'alpha': 0.05} -0.262844 -0.251504   
4 {'alpha': 0.1} -0.255047 -0.248050   
5 {'alpha': 0.2} -0.245300 -0.243394   
6 {'alpha': 0.3} -0.239615 -0.240071   
7 {'alpha': 0.4} -0.235440 -0.237312   
8 {'alpha': 0.5} -0.231929 -0.234998   
9 {'alpha': 0.6} -0.228921 -0.232993   
10 {'alpha': 0.7} -0.226282 -0.231315   
11 {'alpha': 0.8} -0.224019 -0.229979   
12 {'alpha': 0.9} -0.222015 -0.228946   
13 {'alpha': 1.0} -0.220178 -0.227974   
14 {'alpha': 2.0} -0.208671 -0.223591   
15 {'alpha': 3.0} -0.204621 -0.222146   
16 {'alpha': 4.0} -0.202254 -0.221179   
17 {'alpha': 5.0} -0.200649 -0.220481   
18 {'alpha': 6.0} -0.199546 -0.220258   
19 {'alpha': 7.0} -0.198888 -0.220232   
20 {'alpha': 8.0} -0.198546 -0.220440   
21 {'alpha': 9.0} -0.198638 -0.220656   
22 {'alpha': 10.0} -0.199217 -0.220801   
23 {'alpha': 20} -0.204331 -0.223056   
24 {'alpha': 50} -0.215658 -0.227458   
25 {'alpha': 100} -0.229624 -0.233537   
26 {'alpha': 500} -0.303446 -0.285494   
27 {'alpha': 1000} -0.358615 -0.327782   
  
 split2\_test\_score split3\_test\_score ... mean\_test\_score \  
0 -0.256187 -0.287146 ... -0.271484   
1 -0.256045 -0.286645 ... -0.271026   
2 -0.254512 -0.282841 ... -0.267520   
3 -0.249745 -0.275672 ... -0.260712   
4 -0.246052 -0.271298 ... -0.255915   
5 -0.241834 -0.264408 ... -0.249621   
6 -0.239521 -0.259812 ... -0.245545   
7 -0.238072 -0.256655 ... -0.242567   
8 -0.236839 -0.254218 ... -0.240112   
9 -0.235708 -0.252001 ... -0.237956   
10 -0.234665 -0.250007 ... -0.236099   
11 -0.233659 -0.248209 ... -0.234475   
12 -0.232688 -0.246528 ... -0.233025   
13 -0.231751 -0.245045 ... -0.231694   
14 -0.224693 -0.237342 ... -0.224055   
15 -0.222011 -0.232986 ... -0.221087   
16 -0.220612 -0.229815 ... -0.219522   
17 -0.219362 -0.227673 ... -0.218641   
18 -0.218678 -0.226508 ... -0.218232   
19 -0.218042 -0.225746 ... -0.218009   
20 -0.217404 -0.225389 ... -0.217949   
21 -0.216785 -0.225069 ... -0.217969   
22 -0.216330 -0.224762 ... -0.218117   
23 -0.216522 -0.224960 ... -0.220802   
24 -0.223491 -0.233963 ... -0.228213   
25 -0.235010 -0.241608 ... -0.236588   
26 -0.304282 -0.286470 ... -0.289420   
27 -0.360820 -0.329024 ... -0.335586   
  
 std\_test\_score rank\_test\_score split0\_train\_score split1\_train\_score \  
0 0.012292 26 -0.133351 -0.129267   
1 0.012064 25 -0.133376 -0.129258   
2 0.010589 24 -0.133671 -0.129188   
3 0.009412 23 -0.134416 -0.129730   
4 0.009021 22 -0.134954 -0.130383   
5 0.008358 21 -0.135779 -0.131322   
6 0.007937 20 -0.136592 -0.132354   
7 0.007809 19 -0.137574 -0.133206   
8 0.007860 18 -0.138530 -0.133955   
9 0.007918 17 -0.139361 -0.134616   
10 0.007991 15 -0.140084 -0.135233   
11 0.008037 14 -0.140731 -0.135778   
12 0.008050 13 -0.141345 -0.136260   
13 0.008100 12 -0.141914 -0.136712   
14 0.009138 10 -0.146711 -0.140268   
15 0.009180 9 -0.150051 -0.143096   
16 0.009229 7 -0.152965 -0.145525   
17 0.009488 6 -0.155699 -0.147866   
18 0.009849 5 -0.158170 -0.150272   
19 0.010136 3 -0.160311 -0.152552   
20 0.010379 1 -0.162207 -0.154604   
21 0.010467 2 -0.163960 -0.156478   
22 0.010400 4 -0.165601 -0.158222   
23 0.010175 8 -0.177928 -0.170337   
24 0.008537 11 -0.197526 -0.190306   
25 0.005073 16 -0.214019 -0.207926   
26 0.013611 27 -0.277224 -0.275227   
27 0.022000 28 -0.325116 -0.324599   
  
 split2\_train\_score split3\_train\_score split4\_train\_score \  
0 -0.137292 -0.131475 -0.130594   
1 -0.137304 -0.131574 -0.130619   
2 -0.137434 -0.132305 -0.130813   
3 -0.138002 -0.133724 -0.131265   
4 -0.138594 -0.134731 -0.131698   
5 -0.139635 -0.136158 -0.132478   
6 -0.140560 -0.137219 -0.133341   
7 -0.141312 -0.137997 -0.134157   
8 -0.141918 -0.138616 -0.134957   
9 -0.142465 -0.139130 -0.135795   
10 -0.142959 -0.139569 -0.136575   
11 -0.143399 -0.139969 -0.137287   
12 -0.143832 -0.140389 -0.137963   
13 -0.144275 -0.140806 -0.138640   
14 -0.147784 -0.144310 -0.143730   
15 -0.150938 -0.147275 -0.147579   
16 -0.153734 -0.150010 -0.150857   
17 -0.156176 -0.152595 -0.153759   
18 -0.158458 -0.154878 -0.156272   
19 -0.160643 -0.156973 -0.158562   
20 -0.162593 -0.158807 -0.160616   
21 -0.164450 -0.160422 -0.162450   
22 -0.166111 -0.161960 -0.164111   
23 -0.177823 -0.174138 -0.176109   
24 -0.195043 -0.192814 -0.194369   
25 -0.211681 -0.209292 -0.210530   
26 -0.278964 -0.276430 -0.279828   
27 -0.325052 -0.326551 -0.330194   
  
 mean\_train\_score std\_train\_score   
0 -0.132396 0.002784   
1 -0.132426 0.002783   
2 -0.132682 0.002807   
3 -0.133427 0.002840   
4 -0.134072 0.002860   
5 -0.135074 0.002942   
6 -0.136013 0.002932   
7 -0.136849 0.002908   
8 -0.137595 0.002857   
9 -0.138273 0.002793   
10 -0.138884 0.002727   
11 -0.139433 0.002671   
12 -0.139958 0.002637   
13 -0.140469 0.002613   
14 -0.144561 0.002615   
15 -0.147788 0.002734   
16 -0.150618 0.002884   
17 -0.153219 0.002974   
18 -0.155610 0.002971   
19 -0.157808 0.002939   
20 -0.159765 0.002907   
21 -0.161552 0.002899   
22 -0.163201 0.002877   
23 -0.175267 0.002825   
24 -0.194012 0.002396   
25 -0.210690 0.002083   
26 -0.277535 0.001670   
27 -0.326302 0.002053   
  
[28 rows x 21 columns]

# plotting mean test and train scoes with alpha   
ridge\_results['param\_alpha'] = ridge\_results['param\_alpha'].astype('int32')  
plt.figure(figsize=(14,6))  
  
# plotting  
plt.plot(ridge\_results['param\_alpha'], ridge\_results['mean\_train\_score'])  
plt.plot(ridge\_results['param\_alpha'], ridge\_results['mean\_test\_score'])  
plt.xlabel('alpha')  
plt.ylabel('Negative Mean Absolute Error')  
plt.title("Negative Mean Absolute Error and alpha")  
plt.legend(['train score', 'test score'], loc='upper right')  
plt.show()



Observation:

* The above plot defines the way to decide the optimum value of alpha.
* The point in which train and test score has less gap between them is the value which we take as an optimum value of alpha
* From the above plot, we came to know that the value with alpha = 6 has a minimum gap between the test and the training score.
* From the above result, we can see the train data has 0.94 as its R2 value, on test data we have 0.91 as R2 value. So it is pretty much predicting well.
* We can say it hasn't overfitted because the test data(91% r2 value) comparable value when compared to train data(94% r2 value)

## Lasso Regression

lasso = Lasso()  
  
# cross validation  
model\_cv = GridSearchCV(estimator = lasso,   
 param\_grid = params,   
 scoring= 'neg\_mean\_absolute\_error',   
 cv = folds,   
 return\_train\_score=True,  
 verbose = 1)   
  
model\_cv.fit(X\_train, y\_train)

Fitting 5 folds for each of 28 candidates, totalling 140 fits

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.  
[Parallel(n\_jobs=1)]: Done 140 out of 140 | elapsed: 2.2s finished

GridSearchCV(cv=5, estimator=Lasso(),  
 param\_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,  
 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0,  
 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,  
 100, 500, 1000]},  
 return\_train\_score=True, scoring='neg\_mean\_absolute\_error',  
 verbose=1)

# Printing the best hyperparameter alpha  
print(model\_cv.best\_params\_['alpha'])

0.001

#Fitting Ridge model for alpha = 0.001 and printing coefficients which have been penalised  
  
alpha =model\_cv.best\_params\_['alpha']  
  
lasso = Lasso(alpha=alpha)  
   
lasso.fit(X\_train, y\_train)

Lasso(alpha=0.001)

lasso.coef\_

array([ 1.58227000e-02, 4.58439845e-02, 6.81179915e-03, 3.52160205e-03,  
 1.27136466e-02, 1.43629321e-01, -6.29749532e-03, 0.00000000e+00,  
 8.35984829e-02, 0.00000000e+00, 8.44755602e-03, 3.48099289e-01,  
 2.32928386e-04, 3.78524955e-02, 3.80161359e-02, 3.51530374e-02,  
 2.75440889e-02, 0.00000000e+00, 5.49603936e-03, 6.07115487e-03,  
 0.00000000e+00, 0.00000000e+00, 1.89455676e-02, -2.22060369e-02,  
 0.00000000e+00, 6.71634035e-02, 4.97521587e-02, -0.00000000e+00,  
 -0.00000000e+00, -1.55684342e-01, -0.00000000e+00, -1.27122314e-01,  
 -0.00000000e+00, -9.02097250e-02, -0.00000000e+00, -3.95827463e-02,  
 -0.00000000e+00, -0.00000000e+00, 1.38745514e-02, -1.71630448e-01,  
 -2.49557947e-02, 8.98347443e-03, 3.47692230e-02, -0.00000000e+00,  
 3.31464278e-02, 4.64484819e-02, 0.00000000e+00, -8.45874109e-03,  
 2.87237326e-01, -9.59567207e-02, -3.40856130e-02, 0.00000000e+00,  
 -0.00000000e+00, -7.18514381e-02, -5.36956701e-02, 6.10600080e-02,  
 -0.00000000e+00, 1.99592443e-01, 2.20781311e-01, -5.37883762e-02,  
 -0.00000000e+00, -0.00000000e+00, -0.00000000e+00, 9.25207467e-02,  
 6.10660662e-01, -2.13070629e-02, 0.00000000e+00, -0.00000000e+00,  
 4.94582547e-02, -0.00000000e+00, 2.86569826e-02, -1.83809062e-01,  
 3.14213474e-02, -0.00000000e+00, -2.73294305e-02, -1.23804794e-01,  
 -1.17894784e-01, 7.51182413e-04, -0.00000000e+00, -0.00000000e+00,  
 7.37167339e-02, -0.00000000e+00, 0.00000000e+00, -0.00000000e+00,  
 -2.65364407e-03, -7.85892540e-02, -5.78363040e-02, -0.00000000e+00,  
 1.12814112e-01, 3.11458709e-01, 5.74843671e-01, 7.24001531e-01,  
 -3.23279698e-02, 2.83574305e-03, 1.45888118e-01, 1.43757101e-01,  
 1.92859390e-01, 1.83655835e-01, -0.00000000e+00, 4.06804353e-03,  
 2.75183716e-01, -3.18184002e-02, -6.22283698e-02, 1.57202847e-02,  
 -0.00000000e+00, -0.00000000e+00, 0.00000000e+00, 0.00000000e+00,  
 0.00000000e+00, 0.00000000e+00, -0.00000000e+00, 0.00000000e+00,  
 0.00000000e+00, -0.00000000e+00, -6.37982017e-03, -0.00000000e+00,  
 0.00000000e+00, -6.24222504e-02, -0.00000000e+00, 0.00000000e+00,  
 2.40561635e-02, 2.62142424e-02, -0.00000000e+00, 0.00000000e+00,  
 0.00000000e+00, 7.08015784e-02, -4.68319624e-02, -4.24565931e-02,  
 0.00000000e+00, 0.00000000e+00, -9.38457512e-03, -0.00000000e+00,  
 4.30967311e-02, 0.00000000e+00, -0.00000000e+00, -1.19263313e-01,  
 4.99865697e-03, -7.00756423e-02, 3.20188126e-02, 6.37890275e-03,  
 -0.00000000e+00, 1.00594599e-01, 2.22035046e-01, 0.00000000e+00,  
 5.34784926e-05, 0.00000000e+00, -3.89114161e-02, 2.95674388e-02,  
 -5.97508383e-02, 7.10491574e-02, -2.35939442e-03, 0.00000000e+00,  
 -0.00000000e+00, 0.00000000e+00, -0.00000000e+00, 2.37585116e-02,  
 -0.00000000e+00, -1.21234673e-02, 0.00000000e+00, -2.01151237e-02,  
 0.00000000e+00, -1.56934776e-02, 9.98818009e-02, -1.50214059e-02,  
 -0.00000000e+00, -0.00000000e+00, -2.17017470e-02, 0.00000000e+00,  
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, -4.30639967e-02,  
 1.32449808e-03, 1.24946178e-02, 0.00000000e+00, 1.21448133e-01,  
 6.51252159e-02, 0.00000000e+00, -3.13077910e-02, -0.00000000e+00,  
 -5.94923009e-02, -3.31223882e-02, -1.02435794e-01, -1.03395116e-01,  
 -0.00000000e+00, -0.00000000e+00, -4.54404547e-03, -0.00000000e+00,  
 2.82968144e-02, -0.00000000e+00, 7.72458433e-02, 2.87389153e-02,  
 -0.00000000e+00, -0.00000000e+00, -0.00000000e+00, -0.00000000e+00,  
 0.00000000e+00, -0.00000000e+00, 2.21691725e-01, 2.50780364e-02,  
 7.31122802e-02, 0.00000000e+00, 8.70587490e-03, -1.21652918e-01,  
 0.00000000e+00, -0.00000000e+00, 5.43390204e-02, 0.00000000e+00,  
 0.00000000e+00, -5.66828857e-02, -1.71987526e-02, -0.00000000e+00,  
 -1.82458490e-02, 1.57957430e-01, 0.00000000e+00, -0.00000000e+00,  
 0.00000000e+00, 0.00000000e+00, -0.00000000e+00, 0.00000000e+00,  
 -3.28696584e-02, -0.00000000e+00, 0.00000000e+00, -0.00000000e+00,  
 2.52888084e-02, -0.00000000e+00, 2.57065846e-02, 0.00000000e+00,  
 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, -0.00000000e+00,  
 1.12745455e-01, -0.00000000e+00, -7.35912372e-02, 0.00000000e+00,  
 0.00000000e+00, 0.00000000e+00, 1.35765743e-01, 8.57558385e-02])

# Lets calculate some metrics such as R2 score, RSS and RMSE  
  
y\_pred\_train = lasso.predict(X\_train)  
y\_pred\_test = lasso.predict(X\_test)  
  
metric3 = []  
r2\_train\_lr = r2\_score(y\_train, y\_pred\_train)  
print("R2 score (Train): ",end="")  
print(r2\_train\_lr)  
metric3.append(r2\_train\_lr)  
  
r2\_test\_lr = r2\_score(y\_test, y\_pred\_test)  
print("R2 score (Test): ",end="")  
print(r2\_test\_lr)  
metric3.append(r2\_test\_lr)  
  
rss1\_lr = np.sum(np.square(y\_train - y\_pred\_train))  
print("RSS (Train): ",end="")  
print(rss1\_lr)  
metric3.append(rss1\_lr)  
  
rss2\_lr = np.sum(np.square(y\_test - y\_pred\_test))  
print("RSS (Test): ",end="")  
print(rss2\_lr)  
metric3.append(rss2\_lr)  
  
mse\_train\_lr = mean\_squared\_error(y\_train, y\_pred\_train)  
print("RMSE (Train): ",end="")  
print(mse\_train\_lr\*\*0.5)  
metric3.append(mse\_train\_lr\*\*0.5)  
  
mse\_test\_lr = mean\_squared\_error(y\_test, y\_pred\_test)  
print("RMSE (Test): ",end="")  
print(mse\_test\_lr\*\*0.5)  
metric3.append(mse\_test\_lr\*\*0.5)

R2 score (Train): 0.9476112896290417  
R2 score (Test): 0.8957748756977405  
RSS (Train): 32.63816656110703  
RSS (Test): 33.56517330806261  
RMSE (Train): 0.22888580203009165  
RMSE (Test): 0.3545592469260375

# cv results  
lasso\_results = pd.DataFrame(model\_cv.cv\_results\_)  
lasso\_results

mean\_fit\_time std\_fit\_time mean\_score\_time std\_score\_time param\_alpha \  
0 0.077978 6.934367e-03 0.004788 7.467229e-04 0.0001   
1 0.034138 7.451194e-03 0.003397 1.351262e-03 0.001   
2 0.008178 7.463402e-04 0.002992 6.309019e-04 0.01   
3 0.006578 1.948715e-03 0.003200 3.950808e-04 0.05   
4 0.005887 1.357606e-03 0.003191 3.992091e-04 0.1   
5 0.004986 4.623108e-07 0.003591 7.978917e-04 0.2   
6 0.006383 1.739807e-03 0.003590 7.974744e-04 0.3   
7 0.005786 1.599240e-03 0.003190 3.954411e-04 0.4   
8 0.005585 1.197219e-03 0.002792 7.462129e-04 0.5   
9 0.004987 2.780415e-07 0.002792 3.989698e-04 0.6   
10 0.007181 2.309633e-03 0.003590 1.017608e-03 0.7   
11 0.005784 1.163072e-03 0.003196 7.515577e-04 0.8   
12 0.005384 4.903579e-04 0.002993 6.316560e-04 0.9   
13 0.005984 1.092711e-03 0.002393 4.885971e-04 1   
14 0.005187 3.990178e-04 0.003789 9.773307e-04 2   
15 0.005186 3.992084e-04 0.002593 7.981899e-04 3   
16 0.004784 3.972191e-04 0.002594 4.890090e-04 4   
17 0.005585 1.739140e-03 0.003191 9.772625e-04 5   
18 0.005385 4.883441e-04 0.002992 5.001110e-07 6   
19 0.005186 7.466970e-04 0.003391 1.017224e-03 7   
20 0.006184 1.595521e-03 0.003390 4.893575e-04 8   
21 0.006684 1.397747e-03 0.003394 1.021962e-03 9   
22 0.004986 8.918036e-04 0.003391 7.982731e-04 10   
23 0.004786 3.986951e-04 0.003191 3.991606e-04 20   
24 0.005288 1.394992e-03 0.002800 7.487338e-04 50   
25 0.005386 7.980348e-04 0.002593 4.884804e-04 100   
26 0.004388 4.884803e-04 0.002793 3.987313e-04 500   
27 0.005388 4.869904e-04 0.002990 1.094414e-03 1000   
  
 params split0\_test\_score split1\_test\_score \  
0 {'alpha': 0.0001} -0.244009 -0.241251   
1 {'alpha': 0.001} -0.219236 -0.213502   
2 {'alpha': 0.01} -0.238160 -0.233897   
3 {'alpha': 0.05} -0.300882 -0.293311   
4 {'alpha': 0.1} -0.333584 -0.312569   
5 {'alpha': 0.2} -0.415207 -0.381174   
6 {'alpha': 0.3} -0.489492 -0.439345   
7 {'alpha': 0.4} -0.548998 -0.481489   
8 {'alpha': 0.5} -0.606347 -0.534567   
9 {'alpha': 0.6} -0.605729 -0.557678   
10 {'alpha': 0.7} -0.605147 -0.558420   
11 {'alpha': 0.8} -0.604661 -0.559273   
12 {'alpha': 0.9} -0.604218 -0.560125   
13 {'alpha': 1.0} -0.603775 -0.560978   
14 {'alpha': 2.0} -0.601555 -0.568747   
15 {'alpha': 3.0} -0.602193 -0.579819   
16 {'alpha': 4.0} -0.608505 -0.593268   
17 {'alpha': 5.0} -0.621056 -0.608928   
18 {'alpha': 6.0} -0.634854 -0.625989   
19 {'alpha': 7.0} -0.646211 -0.641684   
20 {'alpha': 8.0} -0.653381 -0.653981   
21 {'alpha': 9.0} -0.661489 -0.668099   
22 {'alpha': 10.0} -0.670518 -0.682796   
23 {'alpha': 20} -0.801174 -0.826338   
24 {'alpha': 50} -0.801174 -0.826338   
25 {'alpha': 100} -0.801174 -0.826338   
26 {'alpha': 500} -0.801174 -0.826338   
27 {'alpha': 1000} -0.801174 -0.826338   
  
 split2\_test\_score split3\_test\_score ... mean\_test\_score \  
0 -0.243240 -0.262667 ... -0.248311   
1 -0.225342 -0.226990 ... -0.222504   
2 -0.241418 -0.247064 ... -0.242559   
3 -0.320968 -0.327784 ... -0.307262   
4 -0.338502 -0.347517 ... -0.327813   
5 -0.405627 -0.384551 ... -0.385719   
6 -0.480164 -0.433699 ... -0.447866   
7 -0.548406 -0.482263 ... -0.499491   
8 -0.611191 -0.535615 ... -0.553406   
9 -0.611091 -0.552624 ... -0.565942   
10 -0.610992 -0.552540 ... -0.565789   
11 -0.610894 -0.552509 ... -0.565700   
12 -0.610795 -0.552541 ... -0.565646   
13 -0.610695 -0.552630 ... -0.565602   
14 -0.610472 -0.553838 ... -0.566415   
15 -0.612651 -0.556774 ... -0.570382   
16 -0.618527 -0.564426 ... -0.578090   
17 -0.625944 -0.575286 ... -0.588637   
18 -0.637962 -0.589460 ... -0.601712   
19 -0.644947 -0.601893 ... -0.613427   
20 -0.649336 -0.611385 ... -0.622019   
21 -0.654543 -0.622298 ... -0.631439   
22 -0.662934 -0.634525 ... -0.642092   
23 -0.780347 -0.788779 ... -0.778206   
24 -0.780347 -0.788779 ... -0.778206   
25 -0.780347 -0.788779 ... -0.778206   
26 -0.780347 -0.788779 ... -0.778206   
27 -0.780347 -0.788779 ... -0.778206   
  
 std\_test\_score rank\_test\_score split0\_train\_score split1\_train\_score \  
0 0.007804 3 -0.135297 -0.131345   
1 0.005373 1 -0.155521 -0.151864   
2 0.006480 2 -0.222182 -0.219981   
3 0.014404 4 -0.303185 -0.304412   
4 0.015529 5 -0.325405 -0.327843   
5 0.025282 6 -0.381565 -0.384256   
6 0.033688 7 -0.447570 -0.437659   
7 0.043493 8 -0.501070 -0.486716   
8 0.049605 9 -0.548504 -0.541447   
9 0.039702 14 -0.548328 -0.564762   
10 0.039776 13 -0.548250 -0.564526   
11 0.039851 12 -0.548203 -0.564356   
12 0.039915 11 -0.548188 -0.564256   
13 0.039981 10 -0.548214 -0.564216   
14 0.040234 15 -0.549985 -0.565163   
15 0.039901 16 -0.555363 -0.569559   
16 0.040531 17 -0.563418 -0.577872   
17 0.042215 18 -0.573425 -0.588613   
18 0.044228 19 -0.586023 -0.601818   
19 0.043738 20 -0.597716 -0.613950   
20 0.043047 21 -0.606117 -0.622040   
21 0.043295 22 -0.616310 -0.631112   
22 0.044153 23 -0.627576 -0.641787   
23 0.044688 24 -0.768353 -0.764146   
24 0.044688 24 -0.768353 -0.764146   
25 0.044688 24 -0.768353 -0.764146   
26 0.044688 24 -0.768353 -0.764146   
27 0.044688 24 -0.768353 -0.764146   
  
 split2\_train\_score split3\_train\_score split4\_train\_score \  
0 -0.139883 -0.137116 -0.132965   
1 -0.158844 -0.153715 -0.155200   
2 -0.225243 -0.219081 -0.221359   
3 -0.297202 -0.298673 -0.300514   
4 -0.319736 -0.318949 -0.322742   
5 -0.379944 -0.381980 -0.379616   
6 -0.443236 -0.441626 -0.447807   
7 -0.495884 -0.490421 -0.496723   
8 -0.548008 -0.546964 -0.553500   
9 -0.547941 -0.564793 -0.583546   
10 -0.547881 -0.564670 -0.583401   
11 -0.547842 -0.564564 -0.583313   
12 -0.547823 -0.564499 -0.583318   
13 -0.547865 -0.564484 -0.583325   
14 -0.549525 -0.566599 -0.585030   
15 -0.553926 -0.572000 -0.588330   
16 -0.562277 -0.580731 -0.595379   
17 -0.573390 -0.591508 -0.605967   
18 -0.586311 -0.604286 -0.619240   
19 -0.595743 -0.615958 -0.633893   
20 -0.603575 -0.624282 -0.642591   
21 -0.613063 -0.634293 -0.651129   
22 -0.623186 -0.645109 -0.660923   
23 -0.770903 -0.771271 -0.806056   
24 -0.770903 -0.771271 -0.806056   
25 -0.770903 -0.771271 -0.806056   
26 -0.770903 -0.771271 -0.806056   
27 -0.770903 -0.771271 -0.806056   
  
 mean\_train\_score std\_train\_score   
0 -0.135321 0.003013   
1 -0.155029 0.002305   
2 -0.221569 0.002127   
3 -0.300797 0.002694   
4 -0.322935 0.003357   
5 -0.381472 0.001661   
6 -0.443580 0.003815   
7 -0.494163 0.005034   
8 -0.547685 0.003847   
9 -0.561874 0.013146   
10 -0.561746 0.013112   
11 -0.561656 0.013089   
12 -0.561617 0.013091   
13 -0.561621 0.013076   
14 -0.563260 0.013066   
15 -0.567836 0.012565   
16 -0.575936 0.012231   
17 -0.586581 0.012258   
18 -0.599536 0.012438   
19 -0.611452 0.013897   
20 -0.619721 0.014107   
21 -0.629181 0.013690   
22 -0.639716 0.013444   
23 -0.776146 0.015170   
24 -0.776146 0.015170   
25 -0.776146 0.015170   
26 -0.776146 0.015170   
27 -0.776146 0.015170   
  
[28 rows x 21 columns]

# plotting cv results  
plt.figure(figsize=(16,6))  
  
plt.plot(lasso\_results["param\_alpha"], lasso\_results["mean\_test\_score"])  
plt.plot(lasso\_results["param\_alpha"], lasso\_results["mean\_train\_score"])  
plt.xlabel('number of features')  
plt.ylabel('r-squared')  
plt.title("Optimal Number of Features")  
plt.legend(['test score', 'train score'], loc='upper right')  
plt.show()



Observation

* The above plot defines the way to decide the optimum value of alpha.
* The point in which train and test score has less gap between them is the value which we take as an optimum value of alpha
* From the above plot, we came to know that the value with alpha = 0.001 has a minimum gap between the test and the training score.
* From the above result, we can see the train data has 0.94 as its R2 value, on test data we have 0.91 as R2 value. So it is pretty much predicting well.
* We can say it hasn't overfitted because the test data(91% r2 value) comparable value when compared to train data(94% r2 value)

## Final table comparing all the 3 regression techniques

# Creating a table which contain all the metrics  
  
lr\_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS (Test)',  
 'RMSE (Train)','RMSE (Test)'],   
 'Linear Regression': metric  
 }  
  
lr\_metric = pd.DataFrame(lr\_table ,columns = ['Metric', 'Linear Regression'] )  
  
rg\_metric = pd.Series(metric2, name = 'Ridge Regression')  
ls\_metric = pd.Series(metric3, name = 'Lasso Regression')  
  
final\_metric = pd.concat([lr\_metric, rg\_metric, ls\_metric], axis=1)  
final\_metric

Metric Linear Regression Ridge Regression Lasso Regression  
0 R2 Score (Train) 0.958389 0.945720 0.947611  
1 R2 Score (Test) 0.865501 0.887577 0.895775  
2 RSS (Train) 25.923676 33.816358 32.638167  
3 RSS (Test) 43.314606 36.205131 33.565173  
4 RMSE (Train) 0.203988 0.232980 0.228886  
5 RMSE (Test) 0.402774 0.368239 0.354559

**Conclusion**

* From the above two techniques of Lasso and Ridge Regression, we can say that both almost having the same r2 value.
* When comparing the complexity, it is better to use Lasso because as we have 238 variables, Lasso will make the feature selection among the present variables, but Ridge will not reduce columns, it will keep all 238 variables with the reducing the coefficient of variables.

X\_train.shape

(623, 240)

## Lets observe the changes in the coefficients after regularization

betas = pd.DataFrame(index=X\_train.columns)

betas.rows = X\_train.columns

betas['Linear'] = lm.coef\_  
betas['Ridge'] = ridge.coef\_  
betas['Lasso'] = lasso.coef\_

pd.set\_option('display.max\_rows', None)  
betas.head(50)

Linear Ridge Lasso  
LotFrontage 3.964841e-03 0.024244 0.015823  
LotArea 7.409374e-02 0.048207 0.045844  
YearBuilt 7.536414e-03 0.006489 0.006812  
YearRemodAdd 2.108770e-03 0.003922 0.003522  
MasVnrArea 7.061119e-03 0.024453 0.012714  
BsmtFinSF1 1.353609e-01 0.135454 0.143629  
BsmtFinSF2 -1.172048e-02 -0.013924 -0.006297  
BsmtUnfSF -3.396661e-03 -0.019486 0.000000  
TotalBsmtSF 1.418561e-01 0.123215 0.083598  
1stFlrSF 5.076980e-02 0.102257 0.000000  
2ndFlrSF 1.741620e-01 0.128941 0.008448  
GrLivArea 2.004279e-01 0.196498 0.348099  
GarageYrBlt -6.033793e-04 -0.000331 0.000233  
GarageArea 1.598703e-02 0.044210 0.037852  
WoodDeckSF 4.943849e-02 0.041708 0.038016  
OpenPorchSF 3.318087e-02 0.038971 0.035153  
EnclosedPorch 3.756939e-02 0.030644 0.027544  
ScreenPorch -1.088019e-14 0.000000 0.000000  
MoSold 9.102869e-03 0.007651 0.005496  
YrSold 1.089745e-03 0.009126 0.006071  
MSSubClass\_30 -6.243936e-03 0.009312 0.000000  
MSSubClass\_40 1.007527e-14 0.000000 0.000000  
MSSubClass\_45 1.332898e-02 0.028225 0.018946  
MSSubClass\_50 -1.243437e-01 -0.035690 -0.022206  
MSSubClass\_60 -7.037677e-02 0.021018 0.000000  
MSSubClass\_70 2.333797e-02 0.099811 0.067163  
MSSubClass\_75 2.282945e-01 0.062581 0.049752  
MSSubClass\_80 -2.319822e-01 -0.039683 -0.000000  
MSSubClass\_85 4.754011e-02 -0.013476 -0.000000  
MSSubClass\_90 -1.136011e-01 -0.089533 -0.155684  
MSSubClass\_120 -1.338545e-01 -0.029258 -0.000000  
MSSubClass\_160 -2.574485e-01 -0.114615 -0.127122  
MSSubClass\_180 -1.895368e-02 0.004309 -0.000000  
MSSubClass\_190 1.639484e-02 -0.052872 -0.090210  
MSZoning\_RL -1.620416e-01 0.007967 -0.000000  
MSZoning\_RM -2.235158e-01 -0.073509 -0.039583  
LotShape\_Others -1.047216e-01 -0.041202 -0.000000  
LotShape\_Reg 4.396867e-03 -0.009776 -0.000000  
LandContour\_HLS 2.397458e-03 0.063019 0.013875  
LandContour\_Low -2.287144e-01 -0.134276 -0.171630  
LandContour\_Lvl -5.266563e-02 -0.028571 -0.024956  
LotConfig\_Inside 1.093573e-02 0.011429 0.008983  
LotConfig\_Others 7.284064e-02 0.045010 0.034769  
Neighborhood\_Blueste -1.296994e-01 -0.005875 -0.000000  
Neighborhood\_BrDale 1.764130e-01 0.060607 0.033146  
Neighborhood\_BrkSide -2.312644e-01 0.024866 0.046448  
Neighborhood\_ClearCr -6.935360e-02 0.035083 0.000000  
Neighborhood\_CollgCr -2.018685e-01 -0.073140 -0.008459  
Neighborhood\_Crawfor -9.864911e-03 0.174808 0.287237  
Neighborhood\_Edwards -3.486882e-01 -0.131552 -0.095957

## Top 20 features from RIDGE regressoin

betas.Ridge.sort\_values(ascending=False).head(20)

Neighborhood\_StoneBr 0.332754  
OverallQual\_9 0.267742  
GrLivArea 0.196498  
Neighborhood\_Crawfor 0.174808  
BsmtExposure\_Gd 0.170136  
Functional\_Typ 0.168190  
Exterior1st\_BrkFace 0.165941  
OverallQual\_8 0.155514  
BsmtFinSF1 0.135454  
2ndFlrSF 0.128941  
SaleCondition\_Normal 0.128656  
OverallCond\_8 0.127318  
OverallCond\_Others 0.125530  
TotalBsmtSF 0.123215  
Neighborhood\_NridgHt 0.121943  
GarageCars\_3 0.115110  
OverallCond\_6 0.110763  
OverallQual\_10 0.109049  
BsmtCond\_TA 0.107562  
1stFlrSF 0.102257  
Name: Ridge, dtype: float64

## Top 20 features from LASSO regressoin

betas.Lasso.sort\_values(ascending=False).head(20)

OverallQual\_10 0.724002  
Neighborhood\_StoneBr 0.610661  
OverallQual\_9 0.574844  
GrLivArea 0.348099  
OverallQual\_8 0.311459  
Neighborhood\_Crawfor 0.287237  
Exterior1st\_BrkFace 0.275184  
BsmtExposure\_Gd 0.222035  
Functional\_Typ 0.221692  
Neighborhood\_NridgHt 0.220781  
Neighborhood\_NoRidge 0.199592  
OverallCond\_8 0.192859  
OverallCond\_Others 0.183656  
GarageCars\_3 0.157957  
OverallCond\_6 0.145888  
OverallCond\_7 0.143757  
BsmtFinSF1 0.143629  
SaleCondition\_Normal 0.135766  
BedroomAbvGr\_1 0.121448  
OverallQual\_7 0.112814  
Name: Lasso, dtype: float64

## Conclusion:

### - From the above two techniques of Lasso and Ridge Regression, we can say that both almost having the same r2 value.

### - When comparing the complexity, it is better to use Lasso because as we have 238 variables, Lasso will make the feature selection among the present variables, but Ridge will not reduce columns, it will keep all 238 variables with the reducing the coefficient of variables.