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:

i) Which variables are important to predict the price of variable?

ii)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.

In [2]:
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt

import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# To print all columns and rows
pd.set_option('display.max_columns',None)
pd.set_option('display.max_rows',None)
```

In [4]:
```
#importing train dataset
df = pd.read_csv("train.csv") #Reading the csv file
df.head()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	All
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	All
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	All
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	All
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	All

Sales price is my target columns. above is data which I will use for my model training

In [5]:
```
# importing test dataset
dff = pd.read_csv("test.csv") #Reading the csv file
dff.head()
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Uti
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	A
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	A
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	A
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	A
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	A

Once our model is trained and ready for use then I will use above data to predict prices

EDA

In [6]:
```
# checking shape of the train dataset
df.shape
```

Out[6]:
```
(1168, 81)
```

In [7]:
```
Our train data is having 1168 rows and 81 columns
```

```
  File "<ipython-input-7-a143495a9b9a>", line 1
    Our train data is having 1168 rows and 81 columns
        ^
SyntaxError: invalid syntax
```

In [8]:
```
# checking shape of the test dataset
dff.shape
```

Out[8]:
```
(292, 80)
```

Test data is having 292 rows and 80 column. So as we know in this data our target is not given

In [9]:
```
# checking all column names of train dataset
df.columns
```

Out[9]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [10]:
```
# checking all column names of the train dataset
dff.columns
```

Out[10]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition'],
      dtype='object')
```

In [11]:
```
# checking the data types of all columns in train dataset
df.dtypes
```

Out[11]:
```
Id               int64
MSSubClass       int64
MSZoning        object
LotFrontage    float64
LotArea          int64
Street          object
Alley           object
LotShape        object
LandContour     object
Utilities       object
LotConfig       object
LandSlope       object
Neighborhood    object
Condition1      object
Condition2      object
BldgType        object
HouseStyle      object
OverallQual      int64
OverallCond      int64
YearBuilt        int64
YearRemodAdd     int64
RoofStyle       object
RoofMatl        object
Exterior1st     object
Exterior2nd     object
MasVnrType      object
MasVnrArea     float64
ExterQual       object
ExterCond       object
Foundation      object
BsmtQual        object
BsmtCond        object
BsmtExposure    object
BsmtFinType1    object
BsmtFinSF1       int64
BsmtFinType2    object
BsmtFinSF2       int64
BsmtUnfSF        int64
TotalBsmtSF      int64
Heating         object
HeatingQC       object
CentralAir      object
Electrical      object
1stFlrSF         int64
2ndFlrSF         int64
LowQualFinSF     int64
GrLivArea        int64
BsmtFullBath     int64
BsmtHalfBath     int64
FullBath         int64
HalfBath         int64
BedroomAbvGr     int64
KitchenAbvGr     int64
KitchenQual     object
```

```
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC            object
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
SalePrice          int64
dtype: object
```

In [12]:
```
# checking the data types of all columns in test dataset
dff.dtypes
```

Out[12]:
```
Id               int64
MSSubClass       int64
MSZoning        object
LotFrontage    float64
LotArea          int64
Street          object
Alley           object
LotShape        object
LandContour     object
Utilities       object
LotConfig       object
LandSlope       object
Neighborhood    object
Condition1      object
Condition2      object
BldgType        object
HouseStyle      object
OverallQual      int64
OverallCond      int64
YearBuilt        int64
YearRemodAdd     int64
RoofStyle       object
RoofMatl        object
Exterior1st     object
Exterior2nd     object
MasVnrType      object
MasVnrArea     float64
ExterQual       object
ExterCond       object
Foundation      object
BsmtQual        object
BsmtCond        object
BsmtExposure    object
BsmtFinType1    object
BsmtFinSF1       int64
BsmtFinType2    object
BsmtFinSF2       int64
BsmtUnfSF        int64
TotalBsmtSF      int64
Heating         object
HeatingQC       object
CentralAir      object
Electrical      object
1stFlrSF         int64
2ndFlrSF         int64
LowQualFinSF     int64
GrLivArea        int64
BsmtFullBath     int64
BsmtHalfBath     int64
FullBath         int64
HalfBath         int64
BedroomAbvGr     int64
KitchenAbvGr     int64
KitchenQual     object
```

```
TotRmsAbvGrd       int64
Functional        object
Fireplaces         int64
FireplaceQu       object
GarageType        object
GarageYrBlt      float64
GarageFinish      object
GarageCars         int64
GarageArea         int64
GarageQual        object
GarageCond        object
PavedDrive        object
WoodDeckSF         int64
OpenPorchSF        int64
EnclosedPorch      int64
3SsnPorch          int64
ScreenPorch        int64
PoolArea           int64
PoolQC           float64
Fence             object
MiscFeature       object
MiscVal            int64
MoSold             int64
YrSold             int64
SaleType          object
SaleCondition     object
dtype: object
```

I have just check type of data in both datasent as we can see there are some Object data available in both datasent which we need to convert using encoding

In [13]:
```
# checking the info about the train dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1168 entries, 0 to 1167
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1168 non-null   int64  
 1   MSSubClass     1168 non-null   int64  
 2   MSZoning       1168 non-null   object 
 3   LotFrontage    954 non-null    float64
 4   LotArea        1168 non-null   int64  
 5   Street         1168 non-null   object 
 6   Alley          77 non-null     object 
 7   LotShape       1168 non-null   object 
 8   LandContour    1168 non-null   object 
 9   Utilities      1168 non-null   object 
 10  LotConfig      1168 non-null   object 
 11  LandSlope      1168 non-null   object 
 12  Neighborhood   1168 non-null   object 
 13  Condition1     1168 non-null   object 
 14  Condition2     1168 non-null   object 
 15  BldgType       1168 non-null   object 
 16  HouseStyle     1168 non-null   object 
 17  OverallQual    1168 non-null   int64  
 18  OverallCond    1168 non-null   int64  
 19  YearBuilt      1168 non-null   int64  
 20  YearRemodAdd   1168 non-null   int64  
 21  RoofStyle      1168 non-null   object 
 22  RoofMatl       1168 non-null   object 
 23  Exterior1st    1168 non-null   object 
 24  Exterior2nd    1168 non-null   object 
 25  MasVnrType     1161 non-null   object 
 26  MasVnrArea     1161 non-null   float64
 27  ExterQual      1168 non-null   object 
 28  ExterCond      1168 non-null   object 
 29  Foundation     1168 non-null   object 
 30  BsmtQual       1138 non-null   object 
 31  BsmtCond       1138 non-null   object 
 32  BsmtExposure   1137 non-null   object 
 33  BsmtFinType1   1138 non-null   object 
 34  BsmtFinSF1     1168 non-null   int64  
 35  BsmtFinType2   1137 non-null   object 
 36  BsmtFinSF2     1168 non-null   int64  
 37  BsmtUnfSF      1168 non-null   int64  
 38  TotalBsmtSF    1168 non-null   int64  
 39  Heating        1168 non-null   object 
 40  HeatingQC      1168 non-null   object 
 41  CentralAir     1168 non-null   object 
 42  Electrical     1168 non-null   object 
 43  1stFlrSF       1168 non-null   int64  
 44  2ndFlrSF       1168 non-null   int64  
 45  LowQualFinSF   1168 non-null   int64  
 46  GrLivArea      1168 non-null   int64  
 47  BsmtFullBath   1168 non-null   int64  
 48  BsmtHalfBath   1168 non-null   int64  
```

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

In [14]:
```
# Dropping unnecessary columns in train dataset
df = df.drop(["Alley"],axis=1)
df = df.drop(["PoolQC"],axis=1)
df = df.drop(["Fence"],axis=1)
df = df.drop(["MiscFeature"],axis=1)
```

In [15]:
```
# checking the info about the test dataset
dff.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 292 entries, 0 to 291
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             292 non-null    int64  
 1   MSSubClass     292 non-null    int64  
 2   MSZoning       292 non-null    object 
 3   LotFrontage    247 non-null    float64
 4   LotArea        292 non-null    int64  
 5   Street         292 non-null    object 
 6   Alley          14 non-null     object 
 7   LotShape       292 non-null    object 
 8   LandContour    292 non-null    object 
 9   Utilities      292 non-null    object 
 10  LotConfig      292 non-null    object 
 11  LandSlope      292 non-null    object 
 12  Neighborhood   292 non-null    object 
 13  Condition1     292 non-null    object 
 14  Condition2     292 non-null    object 
 15  BldgType       292 non-null    object 
 16  HouseStyle     292 non-null    object 
 17  OverallQual    292 non-null    int64  
 18  OverallCond    292 non-null    int64  
 19  YearBuilt      292 non-null    int64  
 20  YearRemodAdd   292 non-null    int64  
 21  RoofStyle      292 non-null    object 
 22  RoofMatl       292 non-null    object 
 23  Exterior1st    292 non-null    object 
 24  Exterior2nd    292 non-null    object 
 25  MasVnrType     291 non-null    object 
 26  MasVnrArea     291 non-null    float64
 27  ExterQual      292 non-null    object 
 28  ExterCond      292 non-null    object 
 29  Foundation     292 non-null    object 
 30  BsmtQual       285 non-null    object 
 31  BsmtCond       285 non-null    object 
 32  BsmtExposure   285 non-null    object 
 33  BsmtFinType1   285 non-null    object 
 34  BsmtFinSF1     292 non-null    int64  
 35  BsmtFinType2   285 non-null    object 
 36  BsmtFinSF2     292 non-null    int64  
 37  BsmtUnfSF      292 non-null    int64  
 38  TotalBsmtSF    292 non-null    int64  
 39  Heating        292 non-null    object 
 40  HeatingQC      292 non-null    object 
 41  CentralAir     292 non-null    object 
 42  Electrical     291 non-null    object 
 43  1stFlrSF       292 non-null    int64  
 44  2ndFlrSF       292 non-null    int64  
 45  LowQualFinSF   292 non-null    int64  
 46  GrLivArea      292 non-null    int64  
 47  BsmtFullBath   292 non-null    int64  
 48  BsmtHalfBath   292 non-null    int64  
```

```
 49  FullBath       292 non-null    int64
 50  HalfBath       292 non-null    int64
 51  BedroomAbvGr   292 non-null    int64
 52  KitchenAbvGr   292 non-null    int64
 53  KitchenQual    292 non-null    object
 54  TotRmsAbvGrd   292 non-null    int64
 55  Functional     292 non-null    object
 56  Fireplaces     292 non-null    int64
 57  FireplaceQu    153 non-null    object
 58  GarageType     275 non-null    object
 59  GarageYrBlt    275 non-null    float64
 60  GarageFinish   275 non-null    object
 61  GarageCars     292 non-null    int64
 62  GarageArea     292 non-null    int64
 63  GarageQual     275 non-null    object
 64  GarageCond     275 non-null    object
 65  PavedDrive     292 non-null    object
 66  WoodDeckSF     292 non-null    int64
 67  OpenPorchSF    292 non-null    int64
 68  EnclosedPorch  292 non-null    int64
 69  3SsnPorch      292 non-null    int64
 70  ScreenPorch    292 non-null    int64
 71  PoolArea       292 non-null    int64
 72  PoolQC         0 non-null      float64
 73  Fence          44 non-null     object
 74  MiscFeature    10 non-null     object
 75  MiscVal        292 non-null    int64
 76  MoSold         292 non-null    int64
 77  YrSold         292 non-null    int64
 78  SaleType       292 non-null    object
 79  SaleCondition  292 non-null    object
dtypes: float64(4), int64(34), object(42)
memory usage: 182.6+ KB
```

In [16]:
```
# Dropping unnecessary columns in test dataset
dff = dff.drop(["Alley"],axis=1)
dff = dff.drop(["PoolQC"],axis=1)
dff = dff.drop(["Fence"],axis=1)
dff = dff.drop(["MiscFeature"],axis=1)
```

In [17]:
```
# checking unique values of each column in train dataset
df.nunique()
```

Out[17]:
```
Id              1168
MSSubClass        15
MSZoning           5
LotFrontage      106
LotArea          892
Street             2
LotShape           4
LandContour        4
Utilities          1
LotConfig          5
LandSlope          3
Neighborhood      25
Condition1         9
Condition2         8
BldgType           5
HouseStyle         8
OverallQual       10
OverallCond        9
YearBuilt        110
YearRemodAdd      61
RoofStyle          6
RoofMatl           8
Exterior1st       14
Exterior2nd       15
MasVnrType         4
MasVnrArea       283
ExterQual          4
ExterCond          5
Foundation         6
BsmtQual           4
BsmtCond           4
BsmtExposure       4
BsmtFinType1       6
BsmtFinSF1       551
BsmtFinType2       6
BsmtFinSF2       122
BsmtUnfSF        681
TotalBsmtSF      636
Heating            6
HeatingQC          5
CentralAir         2
Electrical         5
1stFlrSF         669
2ndFlrSF         351
LowQualFinSF      21
GrLivArea        746
BsmtFullBath       4
BsmtHalfBath       3
FullBath           4
HalfBath           3
BedroomAbvGr       8
KitchenAbvGr       4
KitchenQual        4
TotRmsAbvGrd      12
```

```
Functional          7
Fireplaces          4
FireplaceQu         5
GarageType          6
GarageYrBlt        97
GarageFinish        3
GarageCars          5
GarageArea        392
GarageQual          5
GarageCond          5
PavedDrive          3
WoodDeckSF        244
OpenPorchSF       176
EnclosedPorch     106
3SsnPorch          18
ScreenPorch        65
PoolArea            8
MiscVal            20
MoSold             12
YrSold              5
SaleType            9
SaleCondition       6
SalePrice         581
dtype: int64
```

In [18]:
```
# Droping unnecessary columns in train dataset
df = df.drop(["Id"],axis=1)
df = df.drop(["Utilities"],axis=1)
```

In [19]:
```
# checking unique values of each column in test dataset
dff.nunique()
```

Out[19]:
```
Id               292
MSSubClass        15
MSZoning           4
LotFrontage       65
LotArea          249
Street             2
LotShape           4
LandContour        4
Utilities          2
LotConfig          5
LandSlope          3
Neighborhood      24
Condition1         8
Condition2         2
BldgType           5
HouseStyle         8
OverallQual        8
OverallCond        7
YearBuilt         84
YearRemodAdd      57
RoofStyle          5
RoofMatl           3
Exterior1st       12
Exterior2nd       14
MasVnrType         4
MasVnrArea       104
ExterQual          4
ExterCond          4
Foundation         6
BsmtQual           4
BsmtCond           3
BsmtExposure       4
BsmtFinType1       6
BsmtFinSF1       184
BsmtFinType2       6
BsmtFinSF2        32
BsmtUnfSF        231
TotalBsmtSF      224
Heating            4
HeatingQC          4
CentralAir         2
Electrical         4
1stFlrSF         238
2ndFlrSF         113
LowQualFinSF       4
GrLivArea        246
BsmtFullBath       3
BsmtHalfBath       2
FullBath           4
HalfBath           3
BedroomAbvGr       7
KitchenAbvGr       3
KitchenQual        4
```

```
TotRmsAbvGrd      10
Functional         6
Fireplaces         3
FireplaceQu        5
GarageType         6
GarageYrBlt       71
GarageFinish       3
GarageCars         5
GarageArea       166
GarageQual         5
GarageCond         5
PavedDrive         3
WoodDeckSF        87
OpenPorchSF       92
EnclosedPorch     36
3SsnPorch          3
ScreenPorch       21
PoolArea           1
MiscVal            8
MoSold            12
YrSold             5
SaleType           6
SaleCondition      4
dtype: int64
```

In [20]:
```
# Droping unnecessary columns in test dataset
dff = dff.drop(["Id"],axis=1)
dff = dff.drop(["Utilities"],axis=1)
```

In [21]:
```
#Lets check the value count of each column to see if there are any unexpected and
for i in df.columns:
        print(df[i].value_counts())
        print('**************************************')
```

```
20     428
60     244
50     113
120     69
70      53
30      52
160     47
80      43
90      41
190     26
85      19
75      14
45      10
180      6
40       3
Name: MSSubClass, dtype: int64
**************************************
RL         928
RM         163
```

I can see more than 80% zero values in

- BsmtFinSF2
- LowQualFinSF
- EnclosedPorch
- 3SsnPorch
- ScreenPorch
- PoolArea
- MiscVal So i have to drop these columns. In some other columns also there are zero values below 60% they are acceptable and also reasonable.

In [22]:
```
#Dropping unnecessary columns in train dataset
df.drop(columns = ['BsmtFinSF2','LowQualFinSF','EnclosedPorch','3SsnPorch','Scree
```

In [23]:
```
#Lets check the value count of each column to see if there are any unexpected and
for i in dff.columns:
        print(dff[i].value_counts())
        print('**************************************')
```

```
20     108
60      55
50      31
120     18
30      17
160     16
80      15
90      11
70       7
190      4
180      4
75       2
45       2
85       1
40       1
Name: MSSubClass, dtype: int64
**************************************
RL         223
RM          55
```

I can see more than 80% zero values in

- BsmtFinSF2
- LowQualFinSF
- EnclosedPorch
- 3SsnPorch
- ScreenPorch
- PoolArea
- MiscVal

So i have to drop these columns. In some other columns also there are zero values below 60% they are acceptable and also reasonable.

In [24]:
```
#Dropping unnecessary columns in test dataset
dff.drop(columns = ['BsmtFinSF2','LowQualFinSF','EnclosedPorch','3SsnPorch','Scre
```

In [25]:
```
# checking null values in the  present train dataset
df.isnull().sum()
```

Out[25]:
```
MSSubClass         0
MSZoning           0
LotFrontage      214
LotArea            0
Street             0
LotShape           0
LandContour        0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         7
MasVnrArea         7
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          30
BsmtCond          30
BsmtExposure      31
BsmtFinType1      30
BsmtFinSF1         0
BsmtFinType2      31
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         0
1stFlrSF           0
2ndFlrSF           0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        0
TotRmsAbvGrd       0
Functional         0
Fireplaces         0
FireplaceQu      551
GarageType        64
```

```
GarageYrBlt       64
GarageFinish      64
GarageCars         0
GarageArea         0
GarageQual        64
GarageCond        64
PavedDrive         0
WoodDeckSF         0
OpenPorchSF        0
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
dtype: int64
```

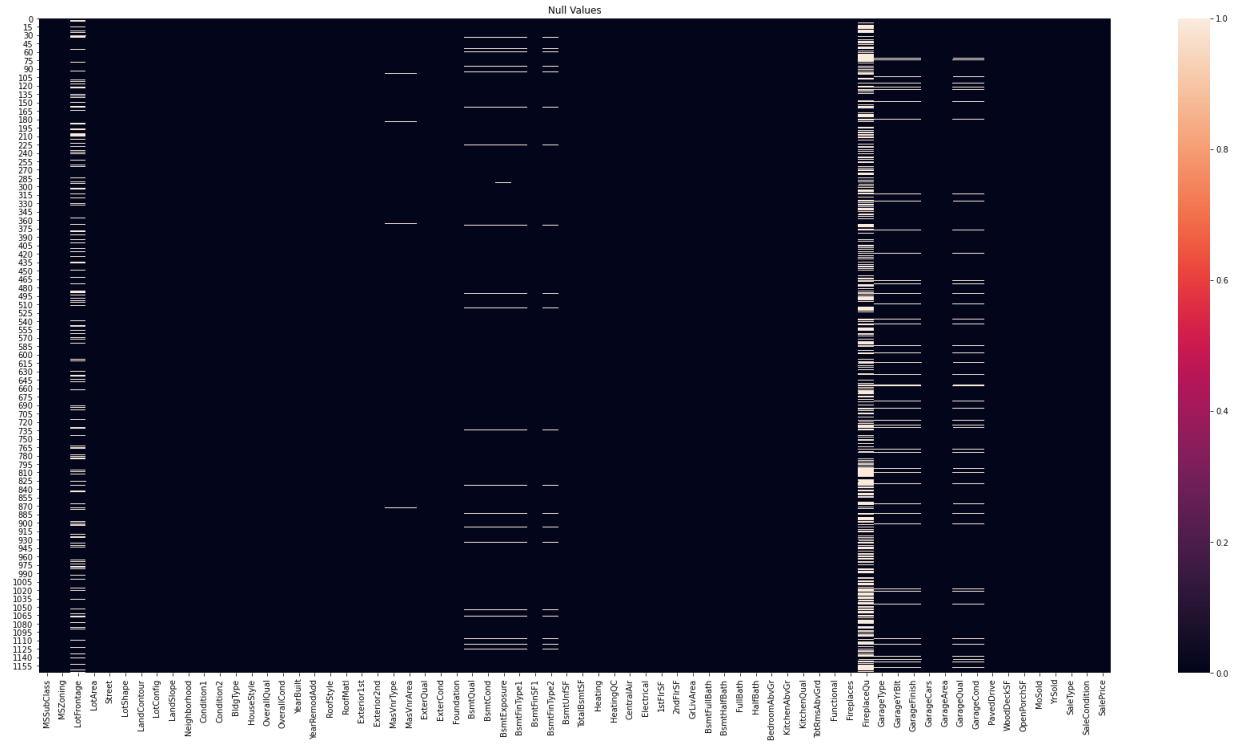
In [26]:
```
# checking null values in the test dataset
dff.isnull().sum()
```

Out[26]:
```
MSSubClass         0
MSZoning           0
LotFrontage       45
LotArea            0
Street             0
LotShape           0
LandContour        0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         1
MasVnrArea         1
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual           7
BsmtCond           7
BsmtExposure       7
BsmtFinType1       7
BsmtFinSF1         0
BsmtFinType2       7
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         1
1stFlrSF           0
2ndFlrSF           0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        0
TotRmsAbvGrd       0
Functional         0
Fireplaces         0
FireplaceQu      139
```

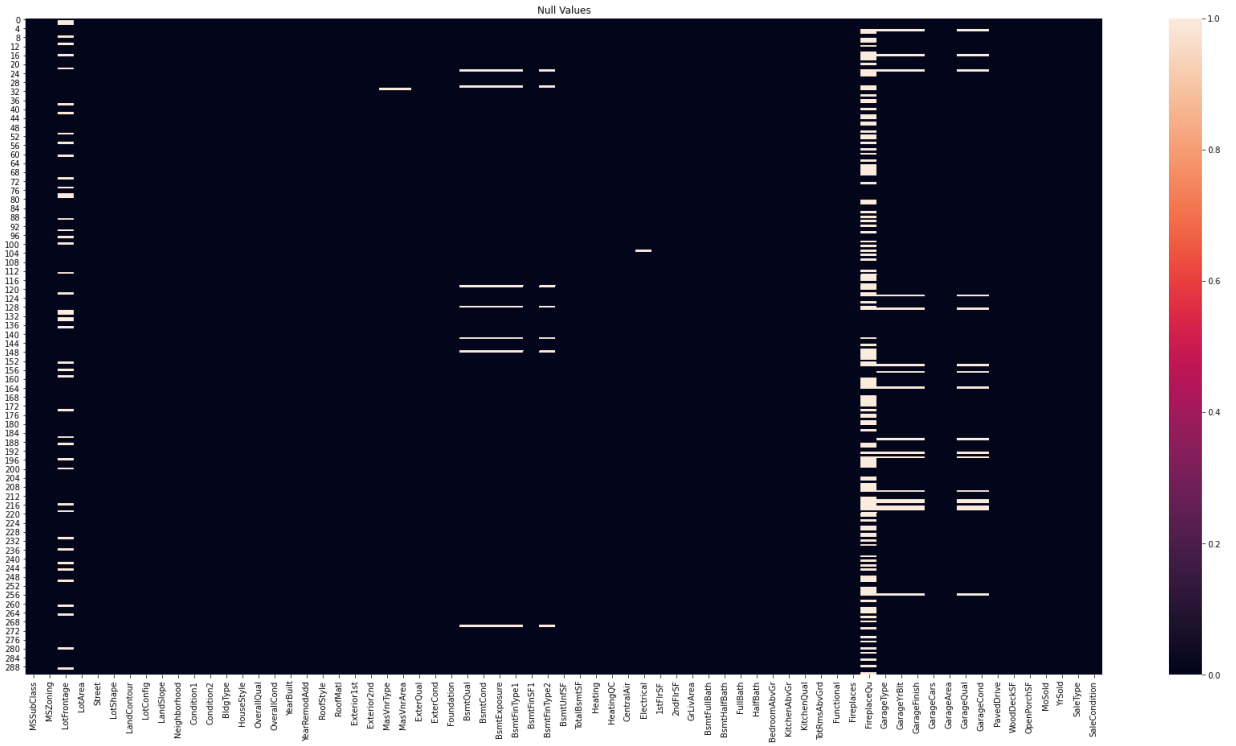
```
GarageType        17
GarageYrBlt       17
GarageFinish      17
GarageCars         0
GarageArea         0
GarageQual        17
GarageCond        17
PavedDrive         0
WoodDeckSF         0
OpenPorchSF        0
MoSold             0
YrSold             0
SaleType           0
SaleCondition      0
dtype: int64
```

We have null values present in both of the data so we will treat them

In [27]:
```
# Visualizing null values in train dataset
plt.figure(figsize=[30,15])
sns.heatmap(df.isnull())
plt.title("Null Values")
plt.show()
```



In [28]:
```
# Visualizing null values in test dataset
plt.figure(figsize=[30,15])
sns.heatmap(dff.isnull())
plt.title("Null Values")
plt.show()
```



Imputation technique to replace nan values:

In [29]:
```
# creating a list of categorical and numerical datatypes in train dataset
df_categorical=[]
df_numerical=[]
for col in df.columns:
    if (df[col].dtype=='object'):
          df_categorical.append(col)
    else:
        df_numerical.append(col)
```

In [30]:
```
#creating a list of categorical and numerical datatypes in test dataset
dff_categorical=[]
dff_numerical=[]
for col in dff.columns:
    if (dff[col].dtype=='object'):
          dff_categorical.append(col)
    else:
        dff_numerical.append(col)
```

In [31]:
```
# Replacing null values of categorical column with mode of that column in train d
catcol=df.columns.values
for i in range(0,len(catcol)):
    if df[catcol[i]].dtype == "object":
        df[catcol[i]].fillna(df[catcol[i]].mode()[0], inplace=True)
```

In [32]:
```
# Replacing null values of categorical column with mode of that column in test da
catcol1=dff.columns.values
for i in range(0,len(catcol1)):
    if dff[catcol1[i]].dtype == "object":
        dff[catcol1[i]].fillna(dff[catcol1[i]].mode()[0], inplace=True)
```

In [33]:
```
# Replacing null values of numerical column with mean of that column in train dat
numcol=df.columns.values
for i in range(0,len(numcol)):
    if df[numcol[i]].dtype != "object":
        df[numcol[i]].fillna(df[numcol[i]].mean(), inplace=True)
```

In [34]:
```
# Replacing null values of numerical column with mean of that column in test data
numcol1=dff.columns.values
for i in range(0,len(numcol1)):
    if dff[numcol1[i]].dtype != "object":
        dff[numcol1[i]].fillna(dff[numcol1[i]].mean(), inplace=True)
```

In [35]:
```
# checking null values again in train dataset
df.isnull().sum()
```

Out[35]:
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
LotShape         0
LandContour      0
LotConfig        0
LandSlope        0
Neighborhood     0
Condition1       0
Condition2       0
BldgType         0
HouseStyle       0
OverallQual      0
OverallCond      0
YearBuilt        0
YearRemodAdd     0
RoofStyle        0
RoofMatl         0
Exterior1st      0
Exterior2nd      0
MasVnrType       0
MasVnrArea       0
ExterQual        0
ExterCond        0
Foundation       0
BsmtQual         0
BsmtCond         0
BsmtExposure     0
BsmtFinType1     0
BsmtFinSF1       0
BsmtFinType2     0
BsmtUnfSF        0
TotalBsmtSF      0
Heating          0
HeatingQC        0
CentralAir       0
Electrical       0
1stFlrSF         0
2ndFlrSF         0
GrLivArea        0
BsmtFullBath     0
BsmtHalfBath     0
FullBath         0
HalfBath         0
BedroomAbvGr     0
KitchenAbvGr     0
KitchenQual      0
TotRmsAbvGrd     0
Functional       0
Fireplaces       0
FireplaceQu      0
GarageType       0
```

```
GarageYrBlt      0
GarageFinish     0
GarageCars       0
GarageArea       0
GarageQual       0
GarageCond       0
PavedDrive       0
WoodDeckSF       0
OpenPorchSF      0
MoSold           0
YrSold           0
SaleType         0
SaleCondition    0
SalePrice        0
dtype: int64
```

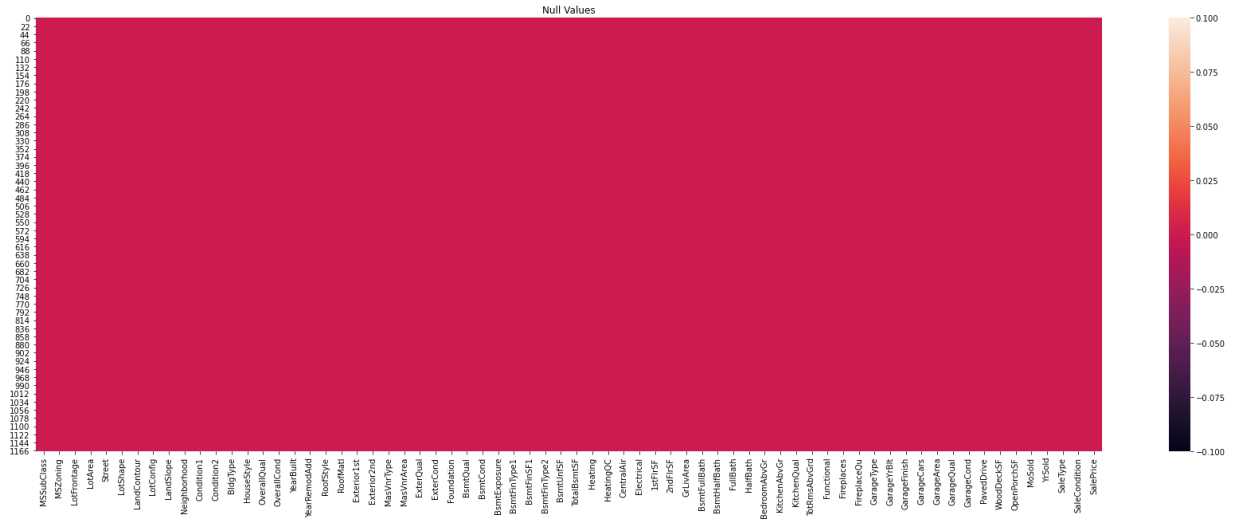
In [36]:
```
# checking null values again in test dataset
dff.isnull().sum()
```

Out[36]:
```
MSSubClass       0
MSZoning         0
LotFrontage      0
LotArea          0
Street           0
LotShape         0
LandContour      0
LotConfig        0
LandSlope        0
Neighborhood     0
Condition1       0
Condition2       0
BldgType         0
HouseStyle       0
OverallQual      0
OverallCond      0
YearBuilt        0
YearRemodAdd     0
RoofStyle        0
RoofMatl         0
Exterior1st      0
Exterior2nd      0
MasVnrType       0
MasVnrArea       0
ExterQual        0
ExterCond        0
Foundation       0
BsmtQual         0
BsmtCond         0
BsmtExposure     0
BsmtFinType1     0
BsmtFinSF1       0
BsmtFinType2     0
BsmtUnfSF        0
TotalBsmtSF      0
Heating          0
HeatingQC        0
CentralAir       0
Electrical       0
1stFlrSF         0
2ndFlrSF         0
GrLivArea        0
BsmtFullBath     0
BsmtHalfBath     0
FullBath         0
HalfBath         0
BedroomAbvGr     0
KitchenAbvGr     0
KitchenQual      0
TotRmsAbvGrd     0
Functional       0
Fireplaces       0
FireplaceQu      0
```

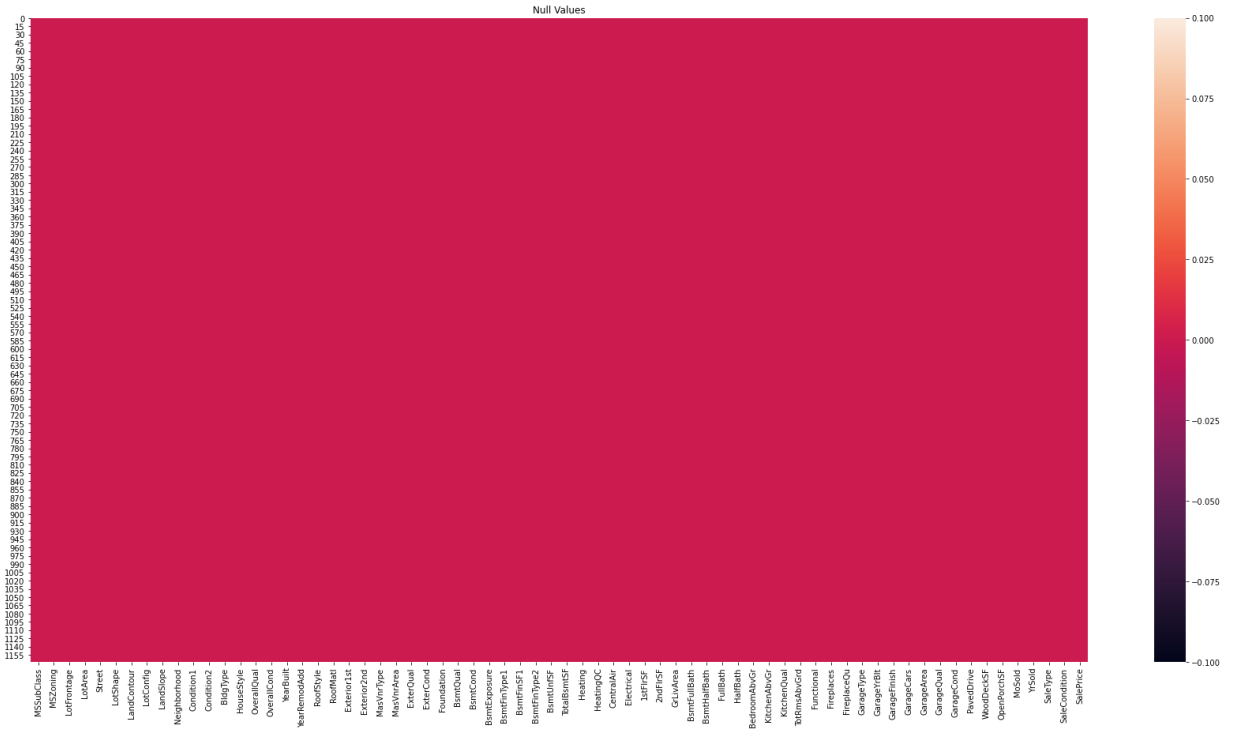
```
GarageType      0
GarageYrBlt     0
GarageFinish    0
GarageCars      0
GarageArea      0
GarageQual      0
GarageCond      0
PavedDrive      0
WoodDeckSF      0
OpenPorchSF     0
MoSold          0
YrSold          0
SaleType        0
SaleCondition   0
dtype: int64
```

We can see that we have imputed null values with mean and mode as per their datatype and now we dont have any missing or null value in dataset

In [37]:
```
#Visualizeing null values again after imputation in train dataset
plt.figure(figsize=[30,10])
sns.heatmap(df.isnull())
plt.title("Null Values")
plt.show()
```



In [40]:
```
# Visualizing null values again after imputation in test dataset
plt.figure(figsize=[30,15])
sns.heatmap(df.isnull())
plt.title("Null Values")
plt.show()
```



In [41]:
```
# checking for empty observations in target column
df.loc[df['SalePrice'] == " "]
```

Out[41]:

MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	LandS

Feature Extraction:

In [42]:
```
# Converting years column to age column in train dataset
df['Year_SinceBuilt'] = df['YearBuilt'].max() - df['YearBuilt']
df['Year_SinceRemodAdded'] = df['YearRemodAdd'].max() - df['YearRemodAdd']
df['Year_SinceSold'] = df['YrSold'].max() - df['YrSold']
df['GarageAge'] = df['GarageYrBlt'].max() - df['GarageYrBlt']
```

In [43]:
```
# Dropping old columns in train dataset
df.drop(['YearBuilt','YearRemodAdd','YrSold','GarageYrBlt'], axis=1, inplace = Tr
```

In [44]:
```
# Converting years column to age column in test dataset
dff['Year_SinceBuilt'] = dff['YearBuilt'].max() - dff['YearBuilt']
dff['Year_SinceRemodAdded'] = dff['YearRemodAdd'].max() - dff['YearRemodAdd']
dff['Year_SinceSold'] = dff['YrSold'].max() - dff['YrSold']
dff['GarageAge'] = dff['GarageYrBlt'].max() - dff['GarageYrBlt']
```

In [45]:
```
# Dropping old columns in test dataset
dff.drop(['YearBuilt','YearRemodAdd','YrSold','GarageYrBlt'], axis=1, inplace = T
```

In [46]:
```
# checking description of data set in train dataset
df.describe()
```

Out[46]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinS
count	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.0000
mean	56.767979	70.988470	10484.749144	6.104452	5.595890	102.310078	444.7260
std	41.940650	22.437056	8957.442311	1.390153	1.124343	182.047152	462.6647
min	20.000000	21.000000	1300.000000	1.000000	1.000000	0.000000	0.0000
25%	20.000000	60.000000	7621.500000	5.000000	5.000000	0.000000	0.0000
50%	50.000000	70.988470	9522.500000	6.000000	5.000000	0.000000	385.5000
75%	70.000000	79.250000	11515.500000	7.000000	6.000000	160.000000	714.5000
max	190.000000	313.000000	164660.000000	10.000000	9.000000	1600.000000	5644.0000

In [47]:
```
# checking description of data set in test dataset
dff.describe()
```

Out[47]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	MasVnrArea	BsmtFinSI
count	292.000000	292.000000	292.000000	292.000000	292.000000	292.000000	292.0000
mean	57.414384	66.425101	10645.143836	6.078767	5.493151	109.171821	439.2945
std	43.780649	19.975962	13330.669795	1.356147	1.063267	174.729023	429.5596
min	20.000000	21.000000	1526.000000	3.000000	3.000000	0.000000	0.0000
25%	20.000000	57.750000	7200.000000	5.000000	5.000000	0.000000	0.0000
50%	50.000000	66.425101	9200.000000	6.000000	5.000000	0.000000	369.5000
75%	70.000000	76.000000	11658.750000	7.000000	6.000000	180.000000	700.5000
max	190.000000	150.000000	215245.000000	10.000000	9.000000	1031.000000	1767.0000

Visualization:

Univariate Analysis:

In [48]:
```
# checking for categorical columns in train dataset
categorical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_columns.append(i)
print(categorical_columns)
```

```
['MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Ne
ighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle',
'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCon
d', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'Bsmt
FinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'Garag
eCond', 'PavedDrive', 'SaleType', 'SaleCondition']
```

In [49]:
```
# checking for categorical columns in test dataset
categorical_columns1=[]
for i in dff.dtypes.index:
    if dff.dtypes[i]=='object':
        categorical_columns1.append(i)
print(categorical_columns1)
```

```
['MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Ne
ighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle',
'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCon
d', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'Bsmt
FinType2', 'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'Garag
eCond', 'PavedDrive', 'SaleType', 'SaleCondition']
```

In [50]:
```
# now checking for numerical columns in train dataset
numerical_columns=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_columns.append(i)
print(numerical_columns)
```

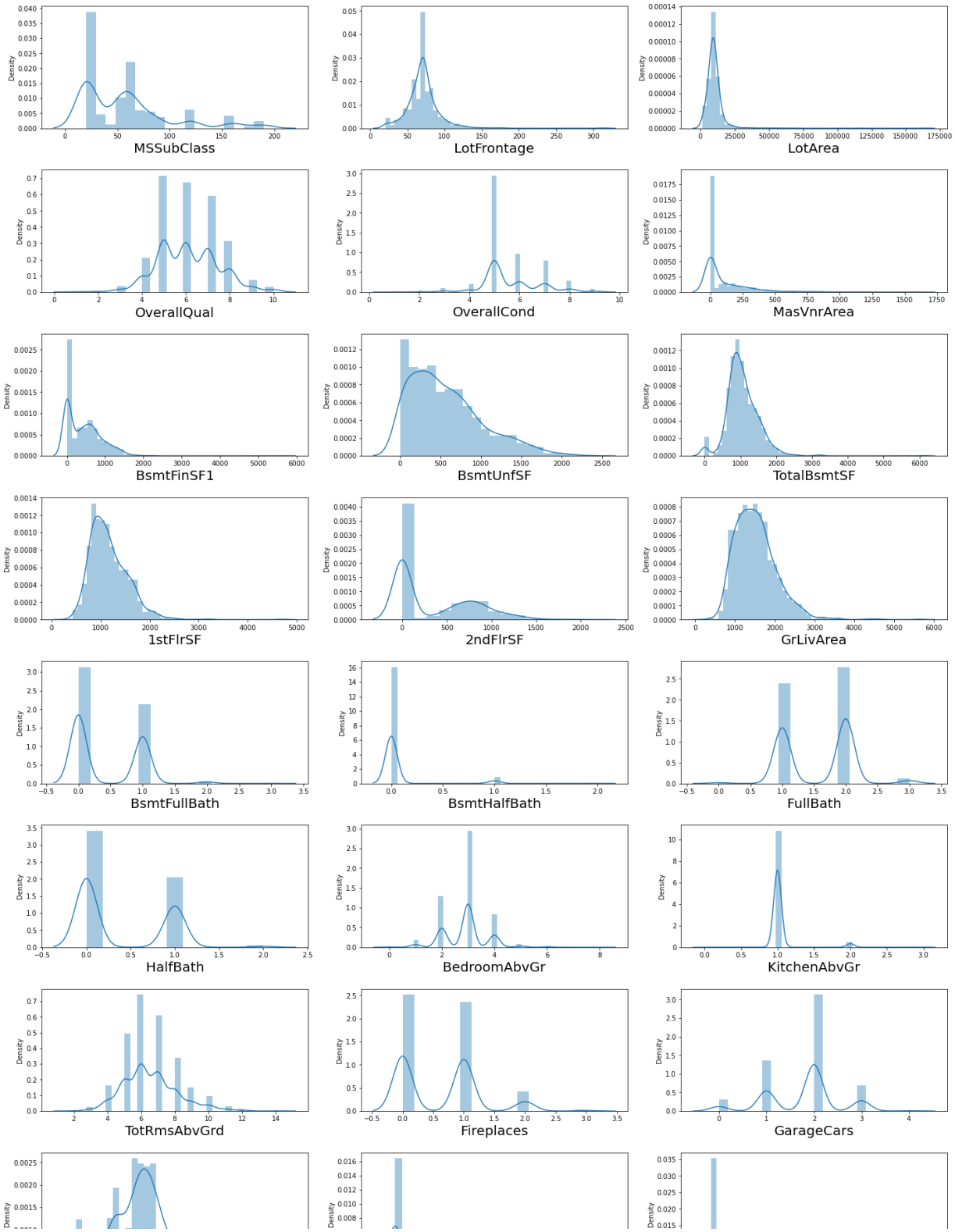
```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'MasVnrA
rea', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivA
rea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'Wood
DeckSF', 'OpenPorchSF', 'MoSold', 'SalePrice', 'Year_SinceBuilt', 'Year_SinceRe
modAdded', 'Year_SinceSold', 'GarageAge']
```

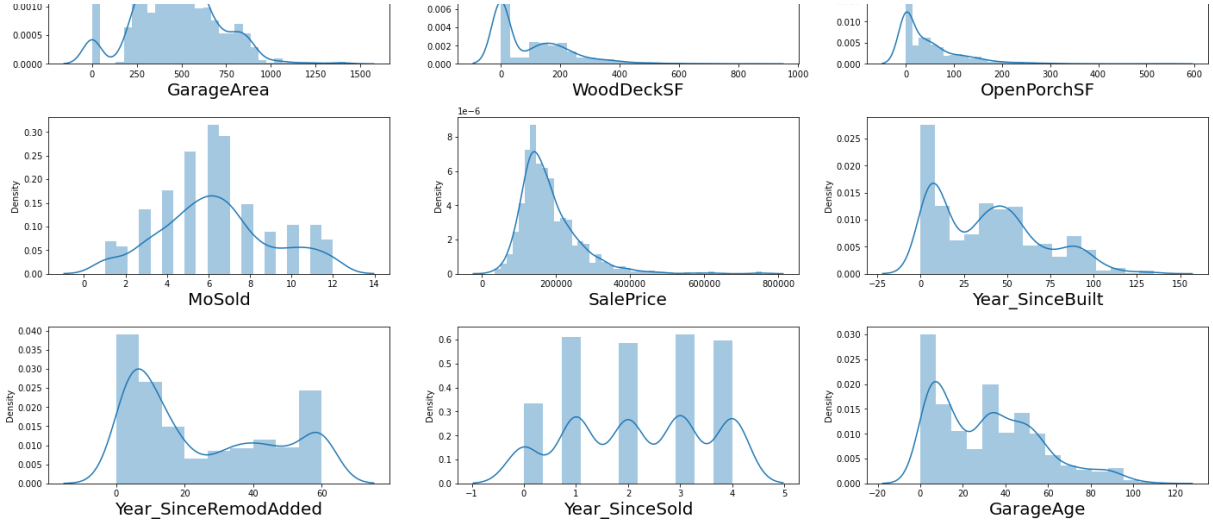
In [51]:
```
# now checking for numerical columns in test dataset
numerical_columns1=[]
for i in dff.dtypes.index:
    if dff.dtypes[i]!='object':
        numerical_columns1.append(i)
print(numerical_columns1)
```

```
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'MasVnrA
rea', 'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivA
rea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars', 'GarageArea', 'Wood
DeckSF', 'OpenPorchSF', 'MoSold', 'Year_SinceBuilt', 'Year_SinceRemodAdded', 'Y
ear_SinceSold', 'GarageAge']
```

Univariate analysis for numerical columns:

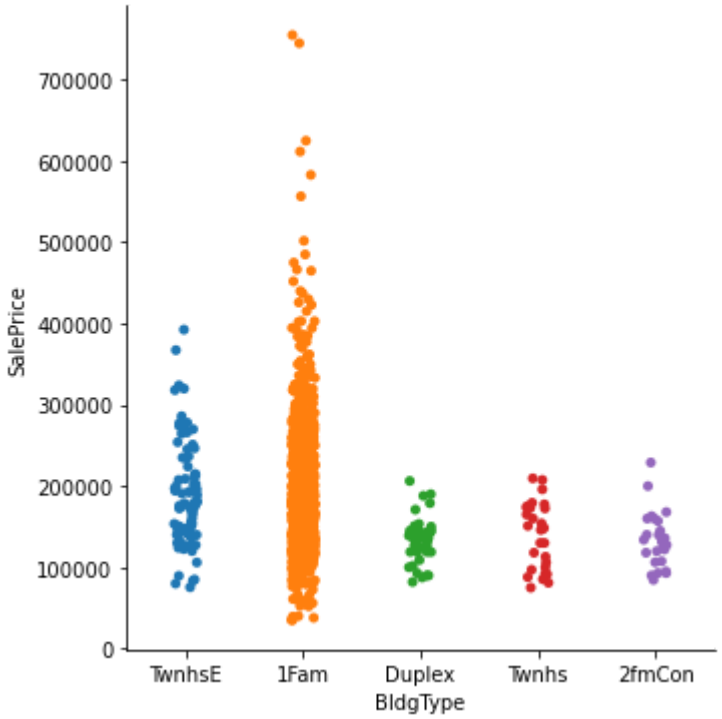
In [52]:
```
# distribution plot for all numerical columns
plt.figure(figsize = (20,40))
plotnumber = 1
for column in df[numerical_columns]:
    if plotnumber <=35:
        ax = plt.subplot(12,3,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize = 20)
    plotnumber+=1
plt.tight_layout()
```





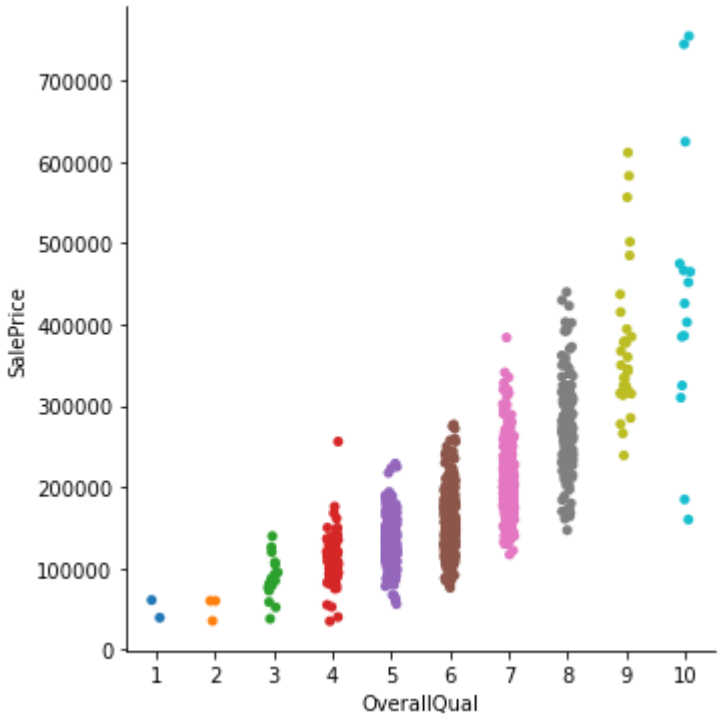
In [53]:
```
sns.catplot(x="BldgType",y="SalePrice", data=df)
```

Out[53]:
```
<seaborn.axisgrid.FacetGrid at 0x29b6619f610>
```



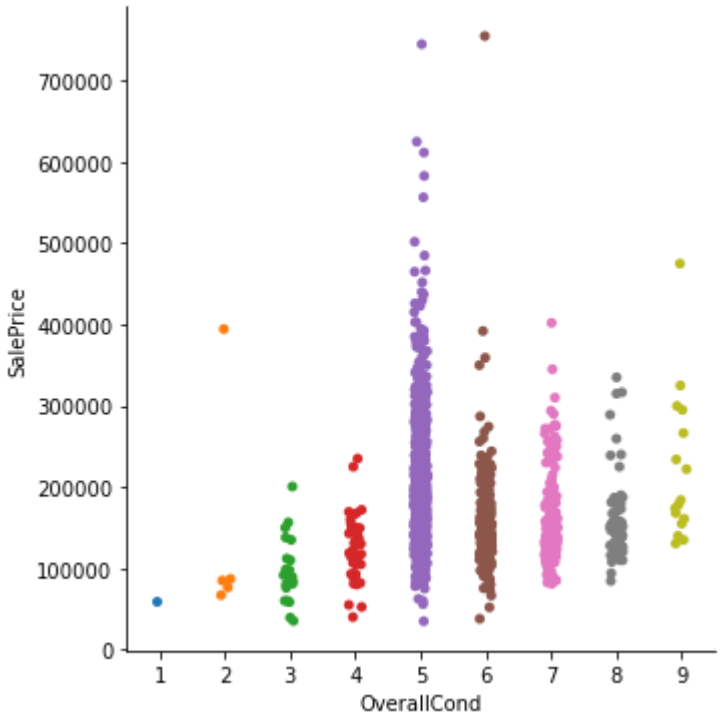
In [54]:
```
sns.catplot(x="OverallQual",y="SalePrice", data=df)
```

Out[54]:
```
<seaborn.axisgrid.FacetGrid at 0x29b661a48e0>
```



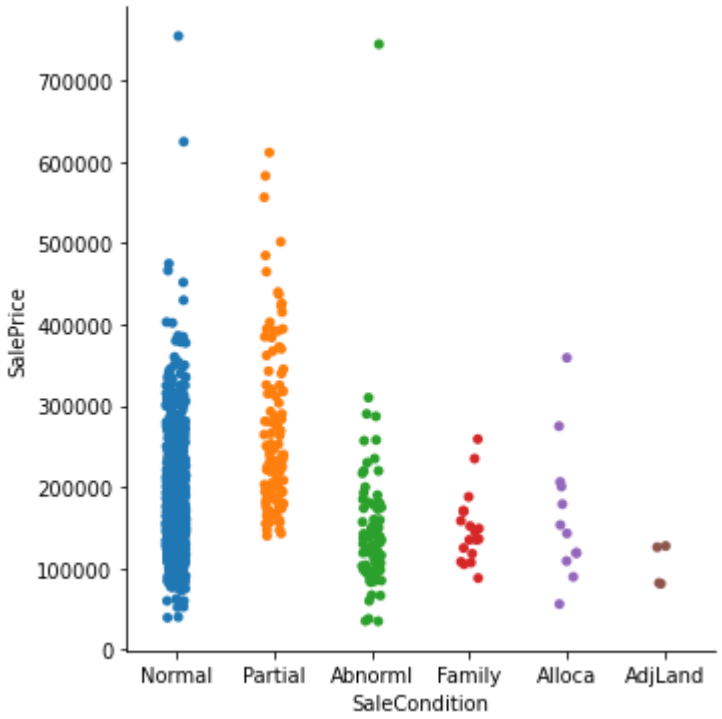
In [55]:
```
sns.catplot(x="OverallCond",y="SalePrice", data=df)
```

Out[55]: <seaborn.axisgrid.FacetGrid at 0x29b670e7fa0>



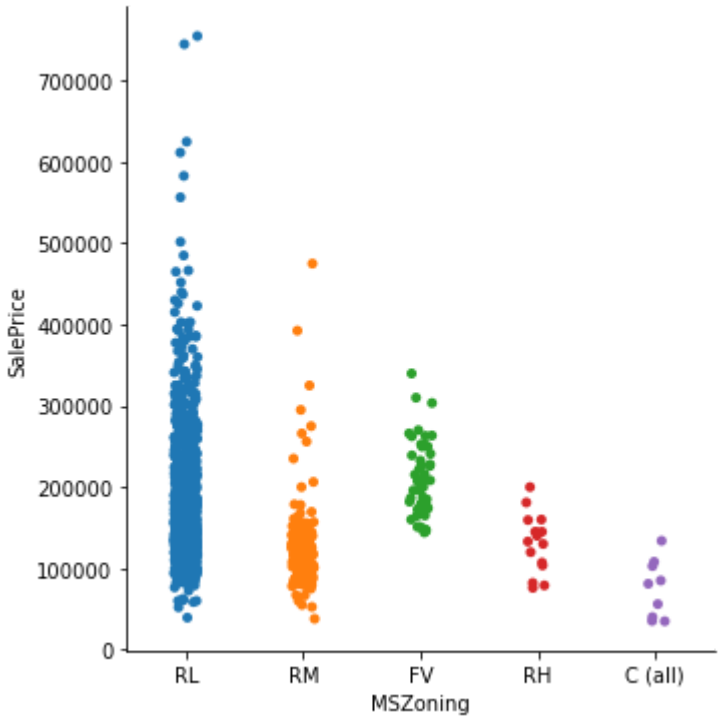
In [56]:
```
sns.catplot(x="SaleCondition",y="SalePrice", data=df)
```

Out[56]:
```
<seaborn.axisgrid.FacetGrid at 0x29b66a1aaf0>
```



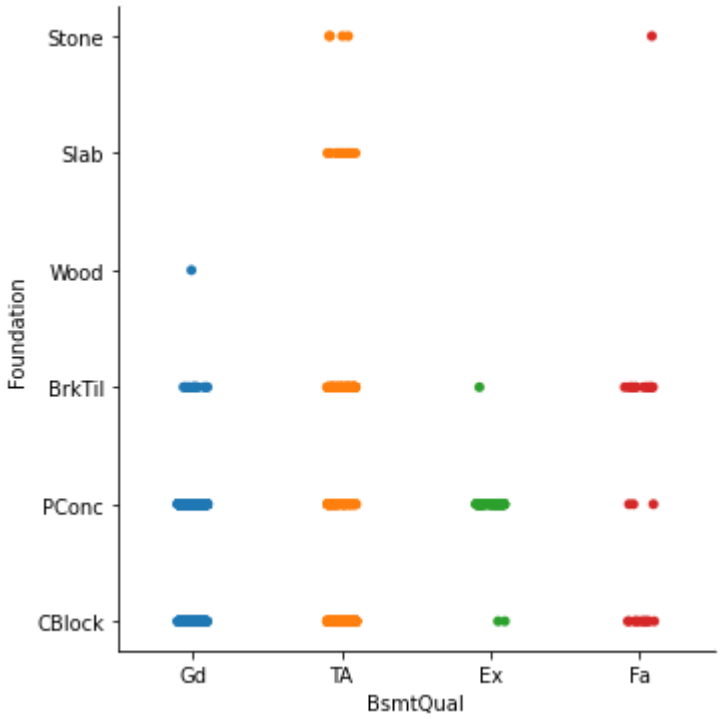
In [57]:
```
sns.catplot(x="MSZoning",y="SalePrice", data=df)
```

Out[57]:
```
<seaborn.axisgrid.FacetGrid at 0x29b673f6850>
```



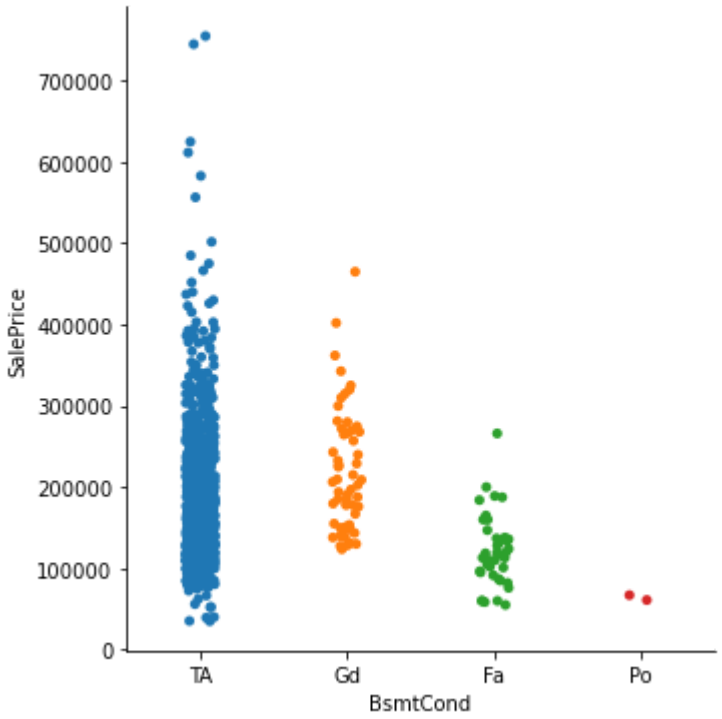
In [58]:
```
sns.catplot(x="BsmtQual",y="Foundation", data=df)
```

Out[58]:
```
<seaborn.axisgrid.FacetGrid at 0x29b6758b550>
```



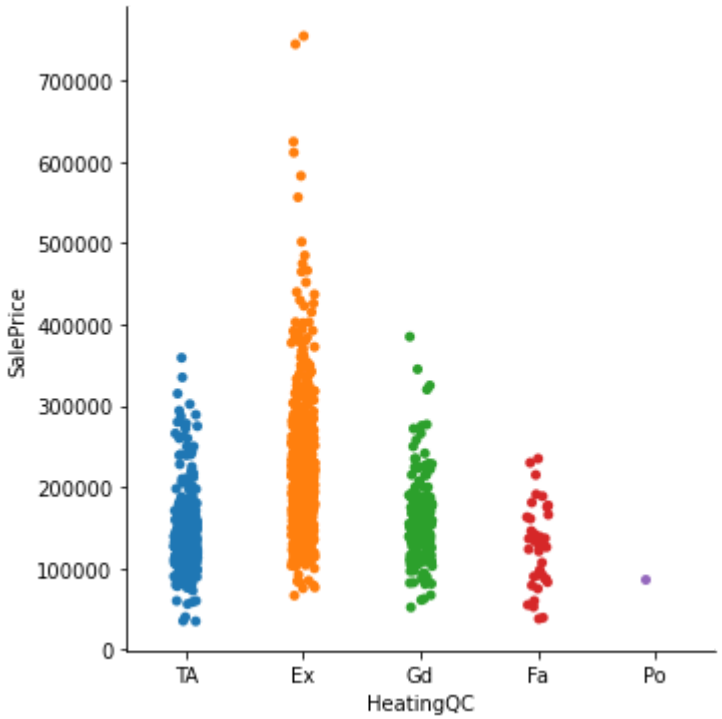
In [59]:
```
sns.catplot(x="BsmtCond",y="SalePrice", data=df)
```

Out[59]:
```
<seaborn.axisgrid.FacetGrid at 0x29b67592580>
```



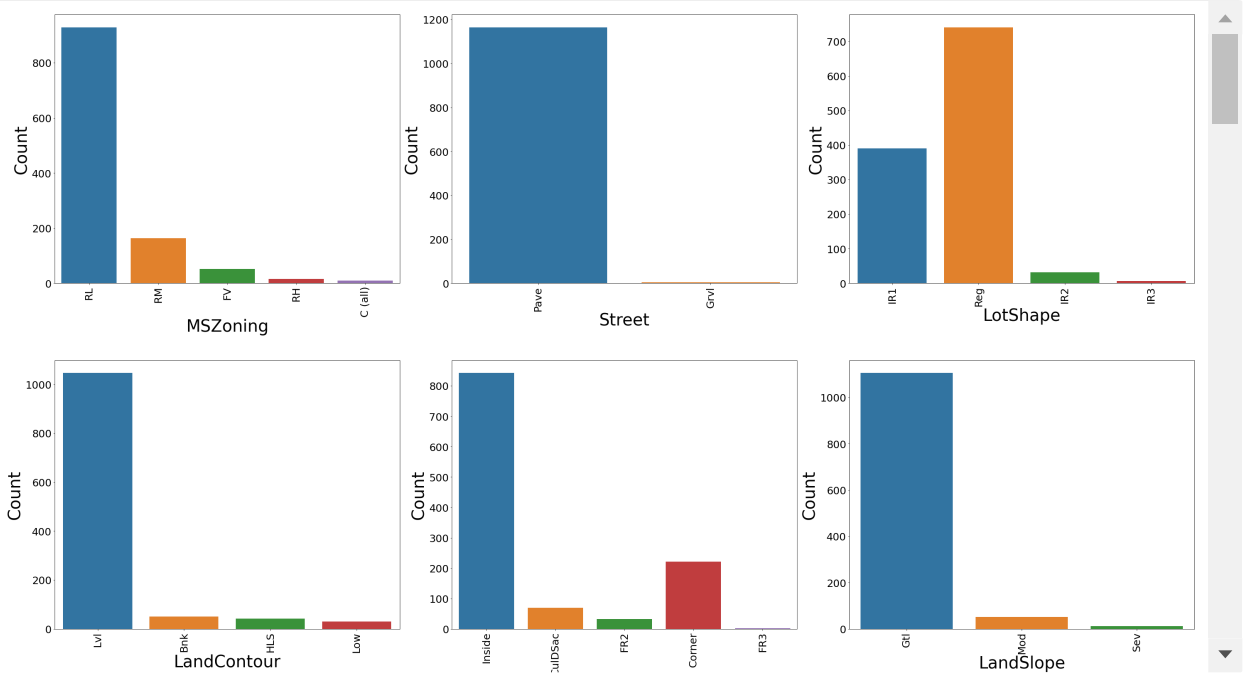
In [60]:
```
sns.catplot(x="HeatingQC",y="SalePrice", data=df)
```

Out[60]:
```
<seaborn.axisgrid.FacetGrid at 0x29b65f8b130>
```



Univariate analysis for categorical columns:

In [61]:
```
# Bar plot for all categorical columns
plt.figure(figsize = (40,150))
plotnumber = 1
for column in df[categorical_columns]:
    if plotnumber <=40:
        ax = plt.subplot(13,3,plotnumber)
        sns.countplot(df[column])
        plt.xticks(rotation=90,fontsize = 25)
        plt.yticks(rotation=0,fontsize = 25)
        plt.xlabel(column,fontsize = 40)
        plt.ylabel('Count',fontsize = 40)
    plotnumber+=1
plt.tight_layout()
```



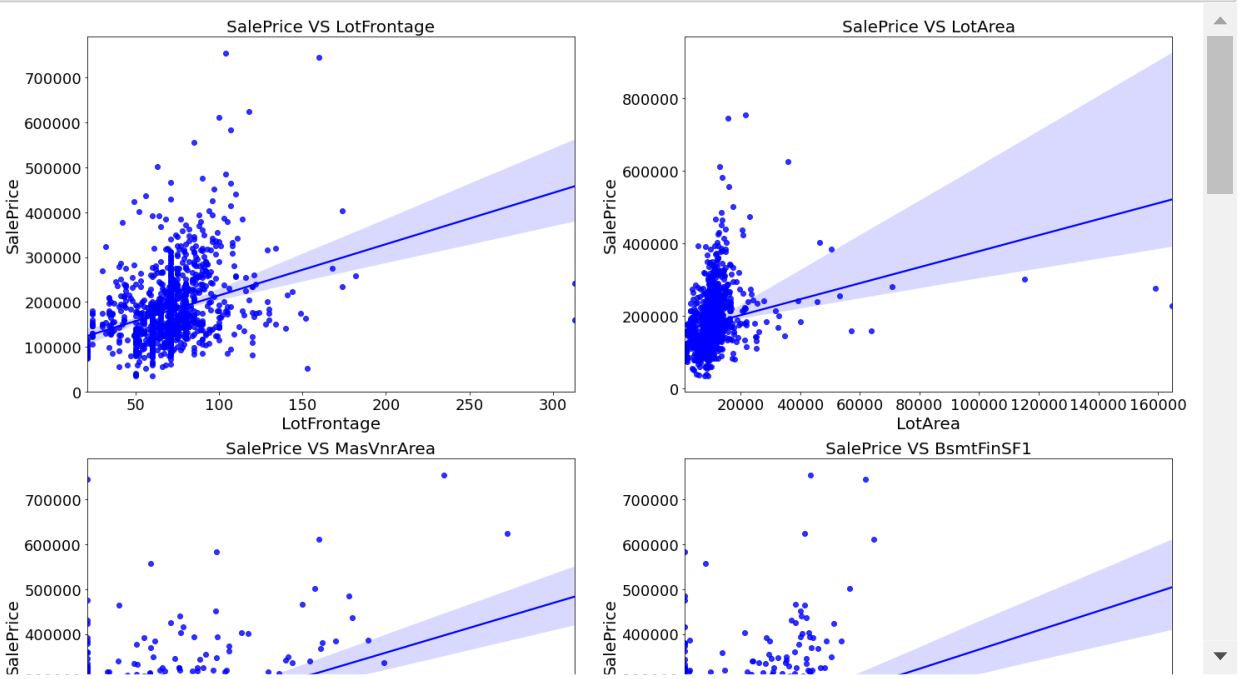
- RL has maximum count in MSzoning and C all has very low sontribution
- street Pave looks famous Street as count is high
- Lotshape Reg has highest count in all shape
- LandContour Lvl has highest count and low is very less
- Housestyle 1story is maximum preferrence and 25story is second largest
- Roofstyle Hip and Gable is the most preferred styles
- Roofmati compshg is the most high used type
- C block and PConc are most used in Foundation
- Most of the housed are having CentralAir systme
- KitchenQual TA and gd are famous one

Bivariate Analysis:

Bivariate Analysis for numerical columns:

In [62]:
```
col=['LotFrontage','LotArea','MasVnrArea','BsmtFinSF1','BsmtUnfSF','TotalBsmtSF',
```

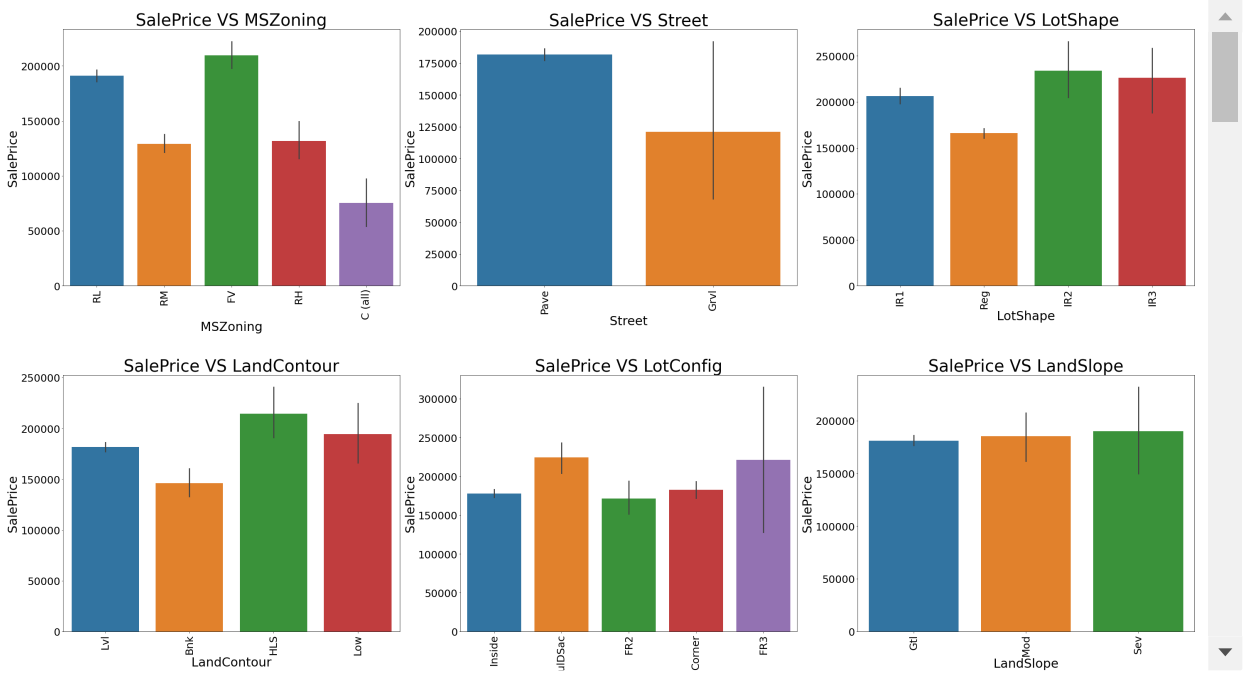
In [63]:
```
# regplot for numerical columns
plt.figure(figsize=(20,140))
for i in range(len(col)):
    plt.subplot(20,2,i+1)
    sns.regplot(x=df[col[i]] , y=df['SalePrice'],color="b")
    plt.title(f"SalePrice VS {col[i]}",fontsize=20)
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)
    plt.xlabel(col[i],fontsize = 20)
    plt.ylabel('SalePrice',fontsize = 20)
    plt.tight_layout()
```



- 1.As Linear feet of street connected to property(LotFrontage) is increasing sales is decreasing and the SalePrice is rangeing between 0-3 lakhs.
- 2.As Lot size in square feet(LotArea) is increseing sales is decreasing and the saleprice is in between 0-4 lakhs.
- 3.As Masonry veneer area in square feet(MasVnrArea) is increasing sales is decreasing and saleprice is rangeing between 0-4 lakhs.
- 4.As Type 1 finished square feet(BsmtFinSF1) is increseing sales is decreasing and the saleprice is in between 0-4 lakhs.
- 5.As Unfinished square feet of basement area(BsmtUnfSF) is increasing sales is decreasing and the saleprice is in between 0-4 lakhs. There are some outliers also.
- 6.As Total square feet of basement area(TotalBsmtSF) is increseing sales is decreasing and the saleprice is in between 0-4 lakhs.
- 7.As First Floor square feet(1stFlrSF) is increseing sales is decreasing and the saleprice is in between 0-4 lakhs.
- 8.As Second floor square feet(2ndFlrSF) is increseing sales is increasing in the range 500-1000 and the saleprice is in between 0-4 lakhs.

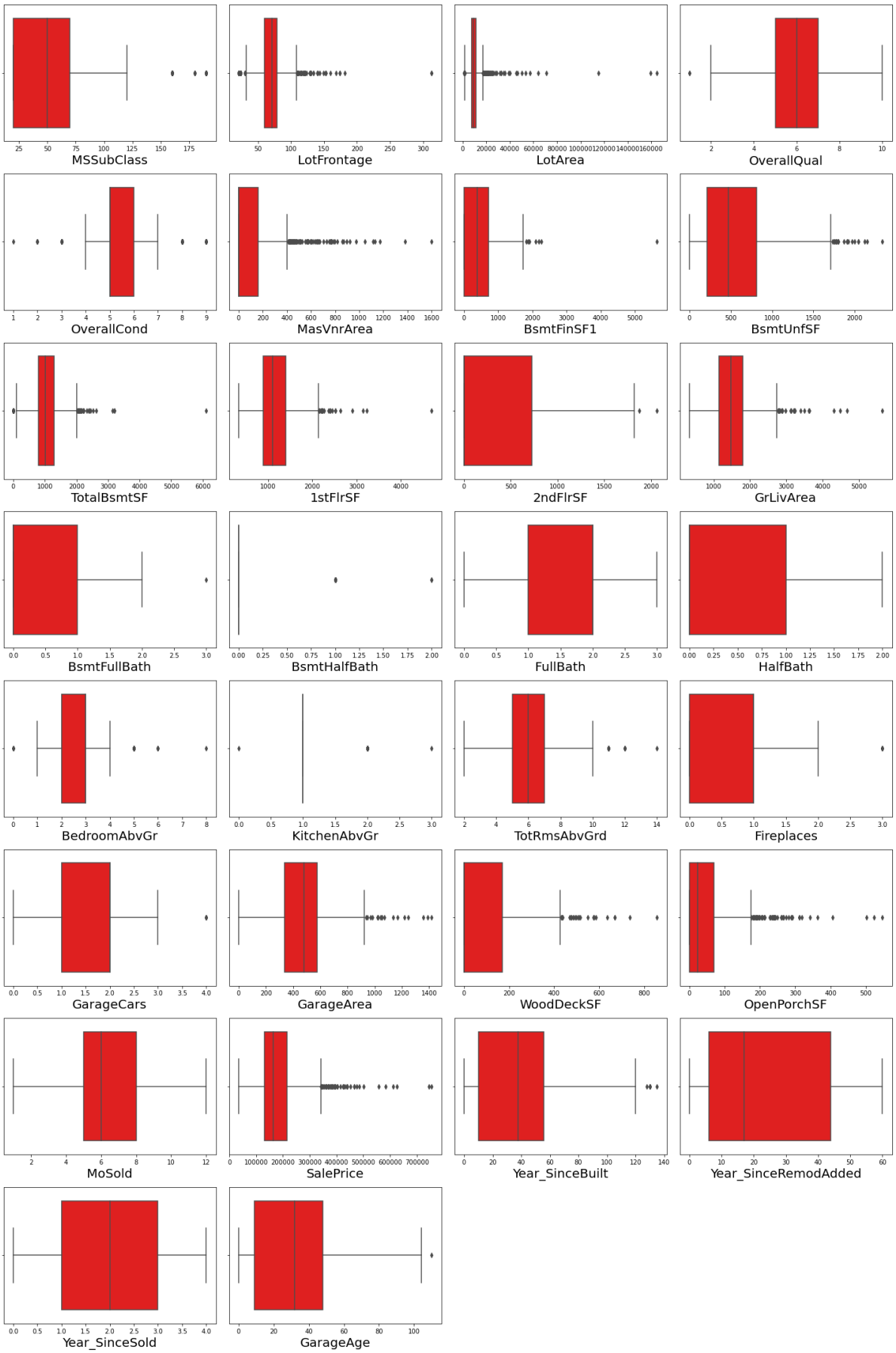
- 9.As Above grade (ground) living area square feet(GrLivArea) is increasing sales is decreasing and the saleprice is in between 0-4 lakhs.
- 10.As Size of garage in square feet(GarageArea) is increasing sales is increasing and the saleprice is in between 0-4 lakhs.
- 11.As Wood deck area in square feet(WoodDeckSF) is increasing sales is decreasing and the saleprice is in between 0-4 lakhs.

In [64]:
```
# Bar plot for all categorical columns
plt.figure(figsize=(40,150))
for i in range(len(categorical_columns)):
    plt.subplot(13,3,i+1)
    sns.barplot(y=df['SalePrice'],x=df[categorical_columns[i]])
    plt.title(f"SalePrice VS {categorical_columns[i]}",fontsize=40)
    plt.xticks(rotation=90,fontsize=25)
    plt.yticks(rotation=0,fontsize=25)
    plt.xlabel(categorical_columns[i],fontsize = 30)
    plt.ylabel('SalePrice',fontsize = 30)
    plt.tight_layout()
```



Cheking outliers

In [65]:
```
plt.figure(figsize=(20,30),facecolor='white')
plotnumber=1
for column in numerical_columns:
    if plotnumber<=30:
        ax=plt.subplot(8,4,plotnumber)
        sns.boxplot(df[column],color='red')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Below are the columns with outliers

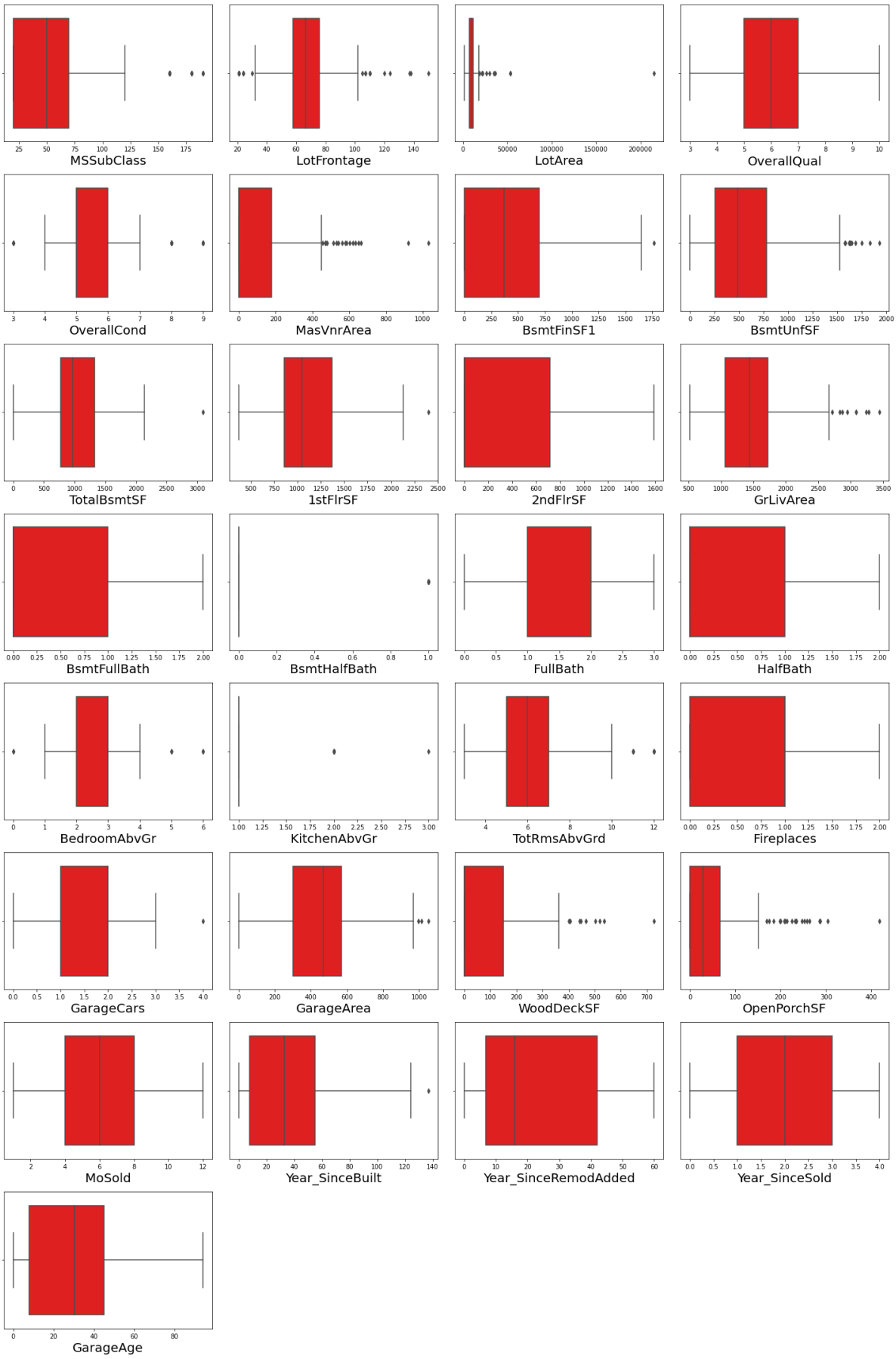
- MSSubClass

- LotFrontage
- LotArea
- OverallQual
- OverallCond
- MasVnrArea
- BsmtFinSF1
- BsmtUnfSF
- TotalBsmtSF
- 1stFlrSF
- 2ndFlrSF
- GrLivArea
- BsmtFullBath
- BsmtHalfBath
- BedroomAbvGr
- KitchenAbvGr
- TotRmsAbvGrd
- Fireplaces
- GarageCars
- GarageArea
- WoodDeckSF
- OpenPorchSF
- Year_SinceBuilt
- GarageAge
- SalePrice

we can see that salesprice is our target column hence we should not remove outliers for it. and we should only handle outliers for numarical columnd

In [66]:
```
# for testing data

plt.figure(figsize=(20,30),facecolor='white')
plotnumber=1
for column in numerical_columns1:
    if plotnumber<=30:
        ax=plt.subplot(8,4,plotnumber)
        sns.boxplot(dff[column],color='red')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



Below are the columns with outliers

- MSSubClass

- LotFrontage
- LotArea
- OverallCond
- MasVnrArea
- BsmtFinSF1
- BsmtUnfSF
- TotalBsmtSF
- 1stFlrSF
- GrLivArea
- BsmtHalfBath
- BedroomAbvGr
- KitchenAbvGr
- TotRmsAbvGrd
- GarageCars
- GarageArea
- WoodDeckSF
- OpenPorchSF
- Year_SinceBuilt

Lets remove outliers now

In [67]:
```
#Features having outliers in train dataset
features=df[['LotFrontage','LotArea','MasVnrArea','BsmtFinSF1','BsmtUnfSF','Total
```

In [68]:
```
from scipy.stats import zscore
z=np.abs(zscore(features))
df_new=df[(z<3).all(axis=1)]
df_new.head()
```

Out[68]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	Land
0	120	RL	70.98847	4928	Pave	IR1	Lvl	Inside	
1	20	RL	95.00000	15865	Pave	IR1	Lvl	Inside	
2	60	RL	92.00000	9920	Pave	IR1	Lvl	CulDSac	
3	20	RL	105.00000	11751	Pave	IR1	Lvl	Inside	
5	60	RL	58.00000	14054	Pave	IR1	Lvl	Inside	

In [69]:
```
# checking shape of new train dataset
df_new.shape
```

Out[69]:
```
(1002, 68)
```

In [70]:
```
#Checking shape of old train dataset
df.shape
```

Out[70]:
```
(1168, 68)
```

In [71]:
```
# Data Loss calculation
Dataloss = (((1168-1002)/1168)*100)
Dataloss
```

Out[71]:
```
14.212328767123289
```

We have data loss more than 10 % which is huge loss of data. Lets try with another outliers handling method

Percentile Method:

In [72]:
```
for col in features:
    if df[col].dtypes != 'object':
        percentile = df[col].quantile([0.01,0.98]).values
        df[col][df[col]<=percentile[0]]=percentile[0]
        df[col][df[col]>=percentile[1]]=percentile[1]
```

In [74]:
```
# features having outliers in test dataset
features1=dff[['LotFrontage','LotArea','MasVnrArea','BsmtFinSF1','BsmtUnfSF','Tot
```

In [75]:
```
# For test data

for col1 in features1:
    if dff[col1].dtypes != 'object':
        percentile = dff[col1].quantile([0.01,0.98]).values
        dff[col][dff[col]<=percentile[0]]=percentile[0]
        dff[col][dff[col]>=percentile[1]]=percentile[1]
```

Skewness checking

In [76]:
```
# checking for skewness of train dataset
df.skew()
```

Out[76]:
```
MSSubClass              1.422019
LotFrontage             0.188060
LotArea                 1.191912
OverallQual             0.175082
OverallCond             0.580714
MasVnrArea              1.873138
BsmtFinSF1              0.639523
BsmtUnfSF               0.777624
TotalBsmtSF             0.166773
1stFlrSF                0.645842
2ndFlrSF                0.717390
GrLivArea               0.592755
BsmtFullBath            0.355224
BsmtHalfBath            3.954345
FullBath                0.057809
HalfBath                0.656492
BedroomAbvGr           -0.145762
KitchenAbvGr            4.365259
TotRmsAbvGrd            0.644657
Fireplaces              0.671966
GarageCars             -0.358556
GarageArea             -0.135675
WoodDeckSF              1.053617
OpenPorchSF             1.513678
MoSold                  0.220979
SalePrice               1.953878
Year_SinceBuilt         0.579204
Year_SinceRemodAdded    0.495864
Year_SinceSold         -0.115765
GarageAge               0.662934
dtype: float64
```

Below are the column whicha re havinf skewness

- MSSubClass
- LotArea
- OverallCond
- MasVnrArea
- BsmtFinSF1
- BsmtUnfSF
- 1stFlrSF
- 2ndFlrSF
- GrLivArea
- BsmtHalfBath
- HalfBath
- KitchenAbvGr
- Fireplaces
- WoodDeckSF
- OpenPorchSF
- SalePrice

- GarageAge

In [77]:
```
# checking for skewness of test dataset
dff.skew()
```

Out[77]:
```
MSSubClass               1.358597
LotFrontage              0.466813
LotArea                 12.781805
OverallQual              0.397312
OverallCond              1.209714
MasVnrArea               1.976804
BsmtFinSF1               0.739790
BsmtUnfSF                0.960708
TotalBsmtSF              0.519257
1stFlrSF                 0.692047
2ndFlrSF                 0.765511
GrLivArea                1.010586
BsmtFullBath             0.463685
BsmtHalfBath             3.544994
FullBath                -0.049800
HalfBath                 0.758892
BedroomAbvGr             0.075315
KitchenAbvGr             4.849432
TotRmsAbvGrd             0.805535
Fireplaces               0.540164
GarageCars              -0.280324
GarageArea               0.133547
WoodDeckSF               1.708221
OpenPorchSF              0.000000
MoSold                   0.186504
Year_SinceBuilt          0.755233
Year_SinceRemodAdded     0.535600
Year_SinceSold          -0.018412
GarageAge                0.683042
dtype: float64
```

Skewed columns as per below

- MSSubClass
- LotArea
- OverallCond
- MasVnrArea
- BsmtFinSF1
- BsmtUnfSF
- 1stFlrSF
- 2ndFlrSF
- GrLivArea
- BsmtHalfBath
- HalfBath
- KitchenAbvGr
- TotRmsAbvGrd

- Fireplaces
- WoodDeckSF
- OpenPorchSF
- Year_SinceBuilt
- Year_SinceRemodAdded

In [79]:
```
# Creating a list of skewed features in train dataset
col1=['LotArea','MasVnrArea','BsmtFinSF1','BsmtUnfSF','1stFlrSF','2ndFlrSF','GrLi
```

removing skewness with yeo-johnson

In [80]:
```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
```

In [82]:
```
df[col1] = scaler.fit_transform(df[col1].values)
```

In [84]:
```
# checking skewness again in train dataset
df[col1].skew()
```

Out[84]:
```
LotArea         0.077862
MasVnrArea      0.415092
BsmtFinSF1     -0.418554
BsmtUnfSF      -0.304290
1stFlrSF       -0.000731
2ndFlrSF        0.279883
GrLivArea      -0.005974
BsmtHalfBath    3.954345
HalfBath        0.498003
KitchenAbvGr   -2.370593
Fireplaces      0.084950
WoodDeckSF      0.110387
OpenPorchSF    -0.010092
dtype: float64
```

In [85]:
```
# Dropping unnecessary column
df = df.drop(["KitchenAbvGr"],axis=1)
```

Removing skeenwss from test data

In [86]:
```
#Creating a list of skewed features in test dataset
col2=['LotArea','MasVnrArea','BsmtFinSF1','BsmtUnfSF','1stFlrSF','2ndFlrSF','GrLi
```

In [87]:
```
dff[col2] = scaler.fit_transform(dff[col2].values)
```

In [89]:
```
# checking skewness again in test dataset
dff[col2].skew()
```

Out[89]:
```
LotArea                0.003111
MasVnrArea             0.363253
BsmtFinSF1            -0.441605
BsmtUnfSF             -0.232099
1stFlrSF              -0.000548
2ndFlrSF               0.284246
GrLivArea             -0.000307
BsmtHalfBath           3.544994
HalfBath               0.621093
KitchenAbvGr           0.000000
TotRmsAbvGrd          -0.000469
Fireplaces             0.076669
WoodDeckSF             0.210102
OpenPorchSF            0.000000
Year_SinceBuilt       -0.147007
Year_SinceRemodAdded  -0.081089
dtype: float64
```

In [90]:
```
# Dropping unnecessary column
dff = dff.drop(["KitchenAbvGr"],axis=1)
```

Ordinal Encoding

In [91]:
```
# Replacing ratings with suitable numbers in required columns in train dataset
column = ['ExterQual','ExterCond','BsmtQual','BsmtCond','HeatingQC','KitchenQual'
for i in column:
    df[i] = df[i].replace({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1, 'None':0})
```

In [92]:
```
# Replacing ratings with suitable numbers in required columns in test dataset
column = ['ExterQual','ExterCond','BsmtQual','BsmtCond','HeatingQC','KitchenQual'
for i in column:
    dff[i] = dff[i].replace({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1, 'None':0})
```

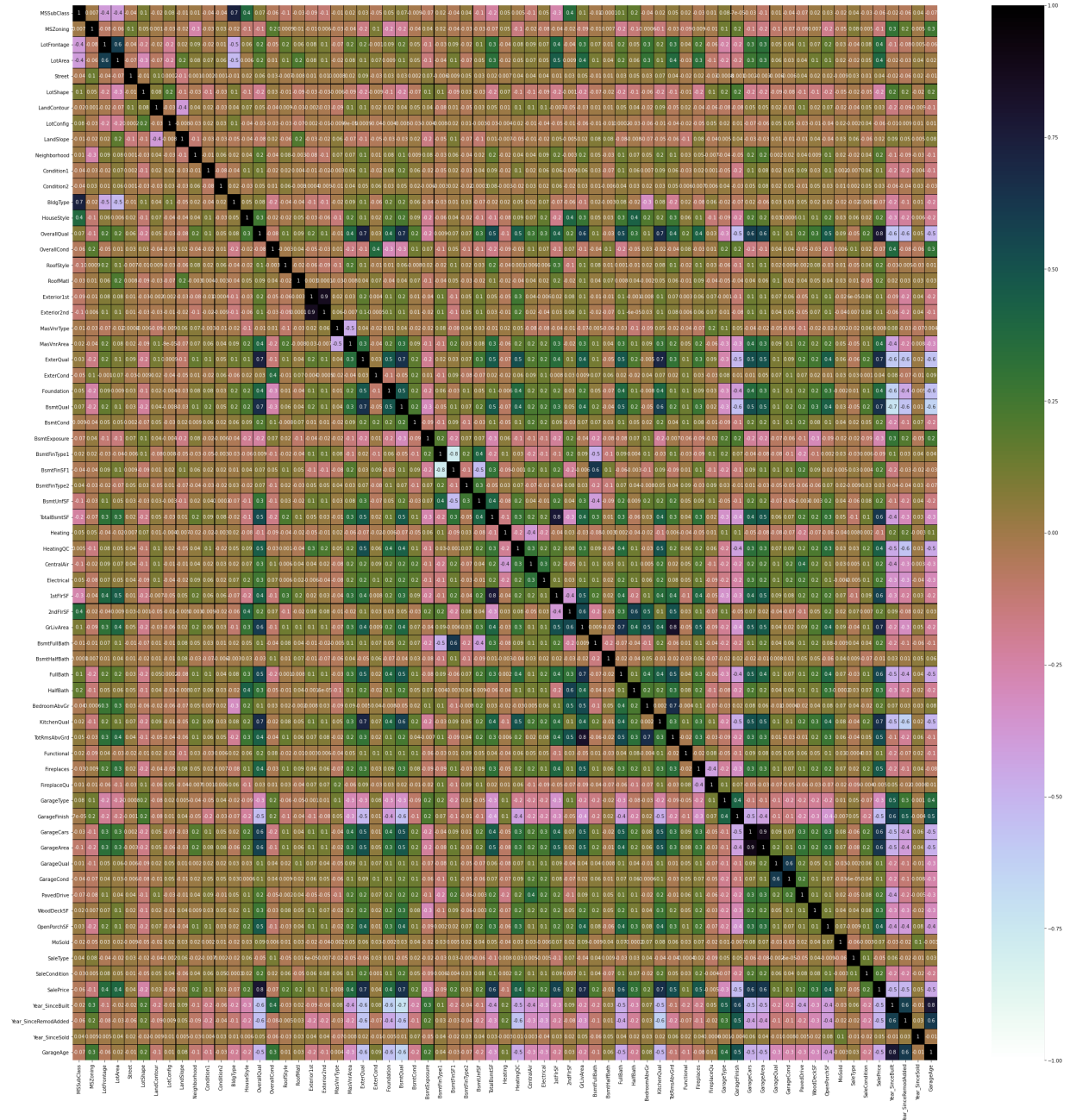
In [93]:
```
# Ordinal encoding for train dataset
from sklearn.preprocessing import OrdinalEncoder
OE = OrdinalEncoder()
for i in df.columns:
    if df[i].dtypes=='object':
        df[i]=OE.fit_transform(df[i].values.reshape(-1,1))
```

In [94]:
```
# Ordinal encoding for test dataset
from sklearn.preprocessing import OrdinalEncoder
OE = OrdinalEncoder()
for i in dff.columns:
    if dff[i].dtypes=='object':
        dff[i]=OE.fit_transform(dff[i].values.reshape(-1,1))
```

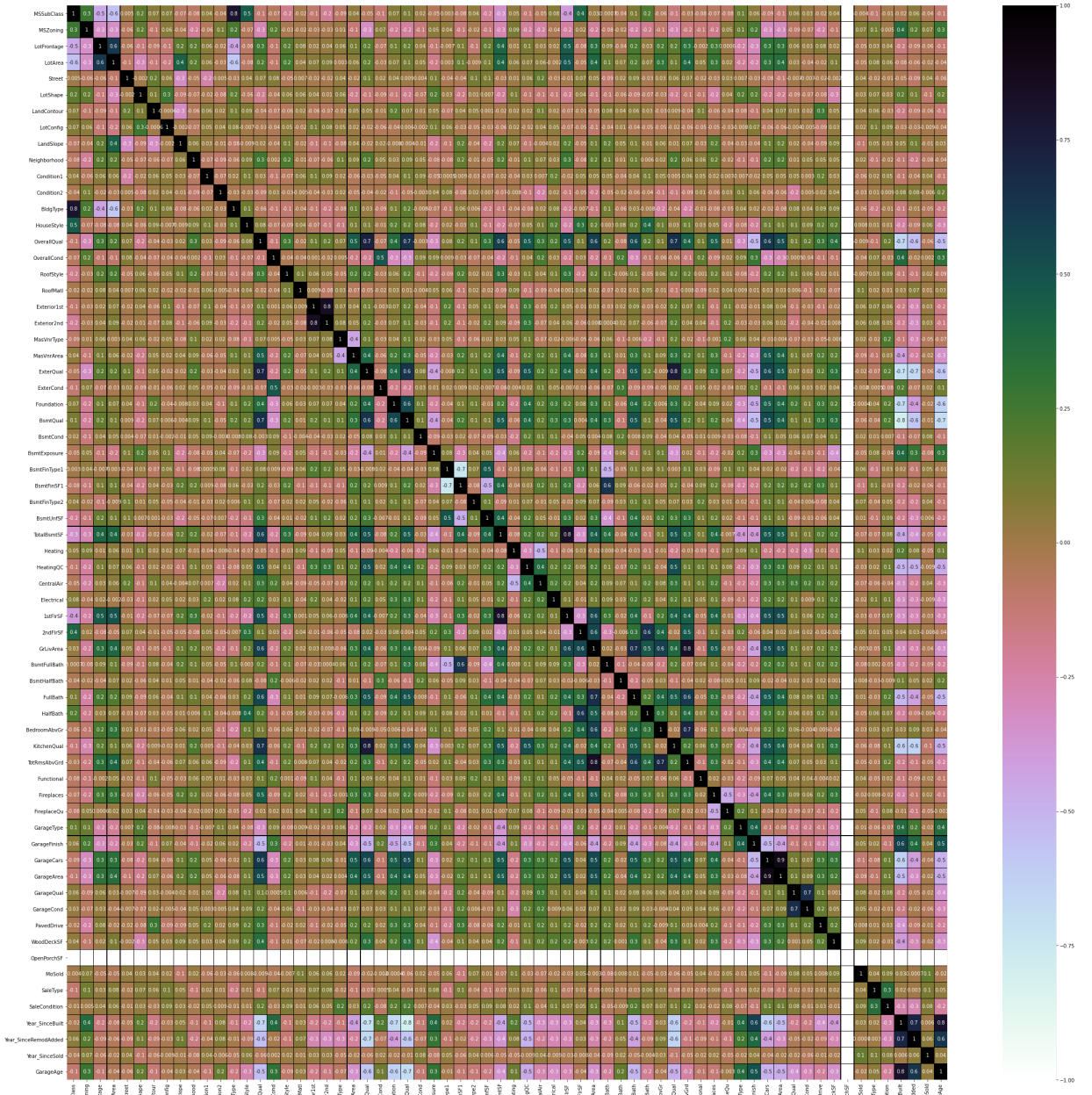
We have encode our data in numerical format

Correlation

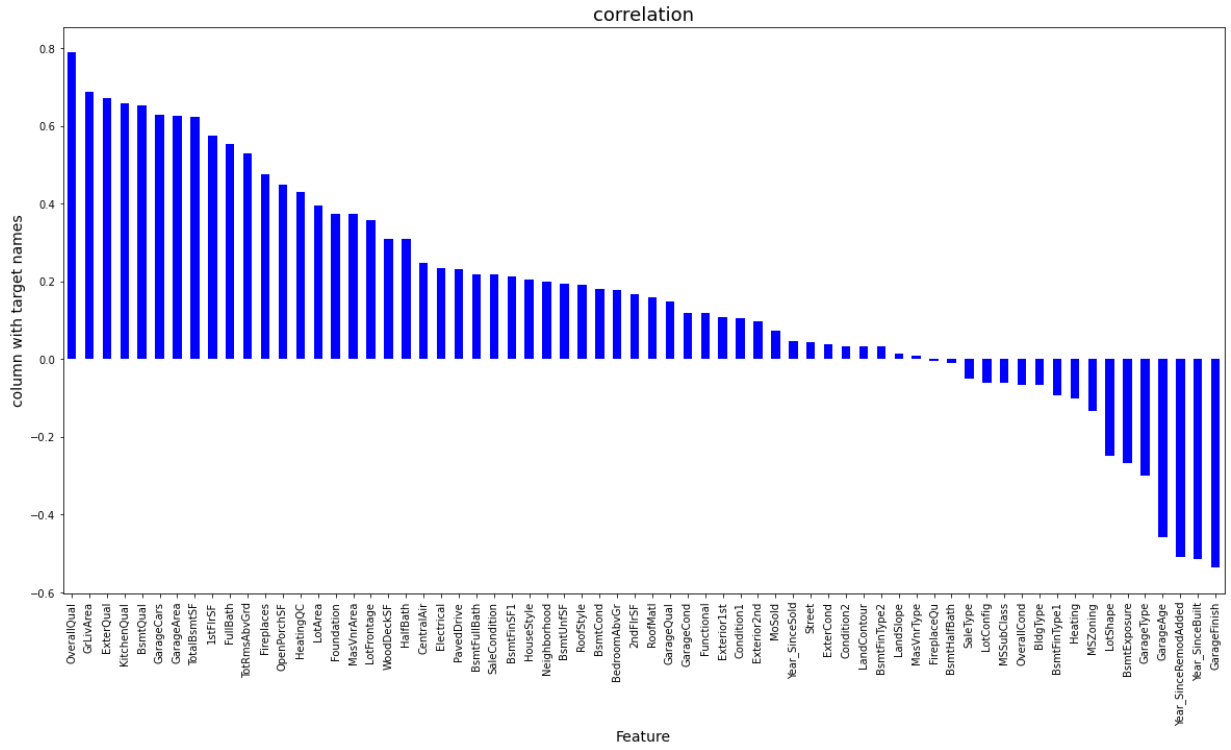
In [95]:
```
# visualizing the correlation matrix by plotting heat map for train dataset.
plt.figure(figsize=(40,40))
sns.heatmap(df.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.1g', annot = True, lir
plt.yticks(rotation=0);
```



In [96]:
```
# Visualizing the correlation matrix by plotting heat map for test dataset.
plt.figure(figsize=(40,40))
sns.heatmap(dff.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.1g', annot = True, li
plt.yticks(rotation=0);
```



In [97]:
```
plt.figure(figsize=(20,10))
df.corr()['SalePrice'].sort_values(ascending=False).drop(['SalePrice']).plot(kind
plt.xlabel('Feature',fontsize=14)
plt.ylabel('column with target names',fontsize=14)
plt.title('correlation',fontsize=18)
plt.show()
```



We can see that nearby 70% coulumns are aving postive correlation with target and rest all have nagetive correaltion with target

Our data is ready now and we can move forward

Defining x and Y

In [98]:
```
x = df.drop("SalePrice",axis=1)
y = df["SalePrice"]
```

In [99]:
```
x.shape
```

Out[99]:
```
(1168, 66)
```

In [100]:
```
y.shape
```

Out[100]:
```
(1168,)
```

Scaling the train data using standard scaler:

In [101]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
```

In [102]:
```
X.head()
```

Out[102]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	LotConfig	L
0	1.508301	-0.021646	0.039092	-1.306083	0.058621	-1.373107	0.318473	0.606420	
1	-0.877042	-0.021646	1.321126	1.356458	0.058621	-1.373107	0.318473	0.606420	
2	0.077095	-0.021646	1.160948	0.113089	0.058621	-1.373107	0.318473	-1.220661	
3	-0.877042	-0.021646	1.855050	0.530989	0.058621	-1.373107	0.318473	0.606420	
4	-0.877042	-0.021646	0.039092	1.497522	0.058621	-1.373107	0.318473	-0.611634	

Scaling the test data using standard scaler

In [103]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_1 = pd.DataFrame(scaler.fit_transform(dff), columns=dff.columns)
```

In [106]:
```
X_1.shape
```

Out[106]:
```
(292, 66)
```

Checking for multicolinearity issue in train dataset using VIF

In [107]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shap
vif["Features"]=X.columns
vif
```

Out[107]:

	vif_Features	Features
0	5.093250	MSSubClass
1	1.356357	MSZoning
2	2.036100	LotFrontage
3	2.645424	LotArea
4	1.107727	Street
5	1.288892	LotShape
6	1.336911	LandContour
7	1.151456	LotConfig
8	1.461325	LandSlope
9	1.255489	Neighborhood
10	1.146316	Condition1
11	1.086460	Condition2
12	5.037705	BldgType
13	2.340086	HouseStyle
14	4.190570	OverallQual
15	2.119268	OverallCond
16	1.258403	RoofStyle
17	1.197352	RoofMatl
18	4.311945	Exterior1st
19	4.323531	Exterior2nd
20	1.542480	MasVnrType
21	1.982516	MasVnrArea
22	3.276437	ExterQual
23	1.315560	ExterCond
24	2.119233	Foundation
25	3.244953	BsmtQual
26	1.252565	BsmtCond
27	1.564053	BsmtExposure
28	3.033460	BsmtFinType1
29	6.213143	BsmtFinSF1
30	1.340639	BsmtFinType2
31	4.575481	BsmtUnfSF

	vif_Features	Features
32	7.549402	TotalBsmtSF
33	1.322687	Heating
34	1.889108	HeatingQC
35	1.697958	CentralAir
36	1.378540	Electrical
37	13.055582	1stFlrSF
38	12.680391	2ndFlrSF
39	19.618638	GrLivArea
40	2.237983	BsmtFullBath
41	1.228506	BsmtHalfBath
42	3.118382	FullBath
43	2.491455	HalfBath
44	2.679456	BedroomAbvGr
45	2.772798	KitchenQual
46	4.581848	TotRmsAbvGrd
47	1.238171	Functional
48	2.186128	Fireplaces
49	1.551971	FireplaceQu
50	1.776246	GarageType
51	2.077976	GarageFinish
52	6.188816	GarageCars
53	6.478492	GarageArea
54	1.858669	GarageQual
55	1.824798	GarageCond
56	1.548704	PavedDrive
57	1.327511	WoodDeckSF
58	1.605707	OpenPorchSF
59	1.084174	MoSold
60	1.117479	SaleType
61	1.186947	SaleCondition
62	7.717894	Year_SinceBuilt
63	3.092263	Year_SinceRemodAdded
64	1.087883	Year_SinceSold
65	4.002911	GarageAge

In [108]:
```
# Droping high VIF columns
X = X.drop(["GrLivArea"],axis=1)
```

In [109]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shap
vif["Features"]=X.columns
vif
```

Out[109]:

	vif_Features	Features
0	5.086826	MSSubClass
1	1.355687	MSZoning
2	2.031739	LotFrontage
3	2.643804	LotArea
4	1.101570	Street
5	1.287431	LotShape
6	1.336283	LandContour
7	1.151225	LotConfig
8	1.456064	LandSlope
9	1.252611	Neighborhood
10	1.145516	Condition1
11	1.083847	Condition2
12	5.021554	BldgType
13	2.294745	HouseStyle
14	4.173592	OverallQual
15	2.112760	OverallCond
16	1.256445	RoofStyle
17	1.195361	RoofMatl
18	4.308002	Exterior1st
19	4.315466	Exterior2nd
20	1.534392	MasVnrType
21	1.982048	MasVnrArea
22	3.276390	ExterQual
23	1.311104	ExterCond
24	2.097885	Foundation
25	3.239886	BsmtQual
26	1.252542	BsmtCond
27	1.561280	BsmtExposure
28	3.033386	BsmtFinType1
29	6.211366	BsmtFinSF1
30	1.339725	BsmtFinType2
31	4.574865	BsmtUnfSF

	vif_Features	Features
32	7.509023	TotalBsmtSF
33	1.320257	Heating
34	1.884648	HeatingQC
35	1.691584	CentralAir
36	1.377826	Electrical
37	6.422458	1stFlrSF
38	5.698870	2ndFlrSF
39	2.237754	BsmtFullBath
40	1.227206	BsmtHalfBath
41	2.944251	FullBath
42	2.397731	HalfBath
43	2.637482	BedroomAbvGr
44	2.772160	KitchenQual
45	4.273309	TotRmsAbvGrd
46	1.231095	Functional
47	2.148708	Fireplaces
48	1.542988	FireplaceQu
49	1.776235	GarageType
50	2.070200	GarageFinish
51	6.186105	GarageCars
52	6.439049	GarageArea
53	1.852192	GarageQual
54	1.823510	GarageCond
55	1.547534	PavedDrive
56	1.326397	WoodDeckSF
57	1.595574	OpenPorchSF
58	1.083199	MoSold
59	1.115731	SaleType
60	1.186606	SaleCondition
61	7.611941	Year_SinceBuilt
62	3.089185	Year_SinceRemodAdded
63	1.087537	Year_SinceSold
64	4.001722	GarageAge

Checking for multicolinearity issue in test dataset using VIF

In [110]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X_1.values, i) for i in range(X_1.
vif["Features"]=X_1.columns
vif
```

Out[110]:

	vif_Features	Features
0	13.626586	MSSubClass
1	1.884112	MSZoning
2	2.172429	LotFrontage
3	4.421420	LotArea
4	1.527782	Street
5	1.654131	LotShape
6	1.582356	LandContour
7	1.346311	LotConfig
8	2.232301	LandSlope
9	1.543010	Neighborhood
10	1.341060	Condition1
11	1.485808	Condition2
12	8.534130	BldgType
13	3.634130	HouseStyle
14	5.617097	OverallQual
15	3.021828	OverallCond
16	1.604300	RoofStyle
17	1.270831	RoofMatl
18	4.987569	Exterior1st
19	5.252024	Exterior2nd
20	2.260134	MasVnrType
21	2.573780	MasVnrArea
22	4.799285	ExterQual
23	1.754237	ExterCond
24	2.848260	Foundation
25	4.574668	BsmtQual
26	1.366720	BsmtCond
27	2.184917	BsmtExposure
28	3.356137	BsmtFinType1
29	8.033539	BsmtFinSF1
30	1.628040	BsmtFinType2
31	6.749260	BsmtUnfSF

	vif_Features	Features
32	12.793646	TotalBsmtSF
33	2.144115	Heating
34	2.302395	HeatingQC
35	2.765113	CentralAir
36	1.606110	Electrical
37	26.599600	1stFlrSF
38	27.614482	2ndFlrSF
39	44.856773	GrLivArea
40	2.746759	BsmtFullBath
41	1.551028	BsmtHalfBath
42	4.005942	FullBath
43	3.324055	HalfBath
44	3.146202	BedroomAbvGr
45	3.703590	KitchenQual
46	5.737238	TotRmsAbvGrd
47	1.673681	Functional
48	3.012969	Fireplaces
49	1.824189	FireplaceQu
50	2.041550	GarageType
51	2.328987	GarageFinish
52	9.756964	GarageCars
53	7.947833	GarageArea
54	2.932611	GarageQual
55	2.705809	GarageCond
56	2.041878	PavedDrive
57	1.771421	WoodDeckSF
58	NaN	OpenPorchSF
59	1.350861	MoSold
60	1.625591	SaleType
61	1.675046	SaleCondition
62	14.830348	Year_SinceBuilt
63	4.648797	Year_SinceRemodAdded
64	1.234314	Year_SinceSold
65	5.871897	GarageAge

In [111]:
```
# Droping high VIF columns
X_1 = X_1.drop(["GrLivArea"],axis=1)
```

In [112]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X_1.values, i) for i in range(X_1.
vif["Features"]=X_1.columns
vif
```

Out[112]:

	vif_Features	Features
0	13.549671	MSSubClass
1	1.884097	MSZoning
2	2.153265	LotFrontage
3	4.420995	LotArea
4	1.523600	Street
5	1.653437	LotShape
6	1.582307	LandContour
7	1.341148	LotConfig
8	2.231504	LandSlope
9	1.543010	Neighborhood
10	1.339021	Condition1
11	1.480678	Condition2
12	8.379374	BldgType
13	3.267328	HouseStyle
14	5.561698	OverallQual
15	3.000873	OverallCond
16	1.598573	RoofStyle
17	1.265276	RoofMatl
18	4.980940	Exterior1st
19	5.198620	Exterior2nd
20	2.256127	MasVnrType
21	2.572250	MasVnrArea
22	4.793886	ExterQual
23	1.751759	ExterCond
24	2.806363	Foundation
25	4.572268	BsmtQual
26	1.366627	BsmtCond
27	2.184031	BsmtExposure
28	3.338550	BsmtFinType1
29	8.027101	BsmtFinSF1
30	1.621135	BsmtFinType2
31	6.671438	BsmtUnfSF

	vif_Features	Features
32	12.412036	TotalBsmtSF
33	2.109692	Heating
34	2.300893	HeatingQC
35	2.764548	CentralAir
36	1.605655	Electrical
37	8.679532	1stFlrSF
38	7.818566	2ndFlrSF
39	2.739138	BsmtFullBath
40	1.550745	BsmtHalfBath
41	3.830519	FullBath
42	3.203928	HalfBath
43	3.090992	BedroomAbvGr
44	3.700177	KitchenQual
45	5.481700	TotRmsAbvGrd
46	1.653628	Functional
47	2.959369	Fireplaces
48	1.804014	FireplaceQu
49	2.038678	GarageType
50	2.327933	GarageFinish
51	9.731516	GarageCars
52	7.905036	GarageArea
53	2.890187	GarageQual
54	2.689422	GarageCond
55	2.038886	PavedDrive
56	1.769715	WoodDeckSF
57	NaN	OpenPorchSF
58	1.342317	MoSold
59	1.620613	SaleType
60	1.654558	SaleCondition
61	14.535096	Year_SinceBuilt
62	4.637267	Year_SinceRemodAdded
63	1.233109	Year_SinceSold
64	5.871092	GarageAge

Our data is ready now

Lets find best random staet

In [113]:
```
# importing necessary libraries
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [114]:
```
from sklearn.ensemble import RandomForestRegressor
maxAccu=0
maxRS=0
for i in range(1,200):
    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.30, random_st
    mod = RandomForestRegressor()
    mod.fit(X_train, y_train)
    pred = mod.predict(X_test)
    acc=r2_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.8982288790064739  on Random_state  50
```

We found our best random state which is 50 and we are getting 89% accuracy with this random satet

In [115]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.30,random_state=50)
```

In [117]:
```
# importing necessary libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingRegressor
from sklearn import metrics
```

RandomForestRegressor

In [118]:
```
RFR=RandomForestRegressor()
RFR.fit(X_train,y_train)
pred=RFR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(RFR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 90.48869060743056
mean_squared_error: 572490073.5776772
mean_absolute_error: 16489.206752136753
root_mean_squared_error: 23926.764795468636

Cross validation score : 83.13980165249832

R2_Score - Cross Validation Score : 7.348888954932235
```

ExtraTreesRegressor

In [119]:
```
ETR=ExtraTreesRegressor()
ETR.fit(X_train,y_train)
pred=ETR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(ETR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 88.74712101172136
mean_squared_error: 677315946.1086572
mean_absolute_error: 17181.10772079772
root_mean_squared_error: 26025.294352007957

Cross validation score : 84.08078742343974

R2_Score - Cross Validation Score : 4.666333588281617
```

GradientBoostingRegressor

In [120]:
```
GBR=GradientBoostingRegressor()
GBR.fit(X_train,y_train)
pred=GBR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(GBR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 91.3707297413273
mean_squared_error: 519399733.66533804
mean_absolute_error: 15394.542524123563
root_mean_squared_error: 22790.342991393045

Cross validation score : 82.46158952618515

R2_Score - Cross Validation Score : 8.909140215142145
```

DecisionTreeRegressor

In [121]:
```
DTR=DecisionTreeRegressor()
DTR.fit(X_train,y_train)
pred=DTR.predict(X_test)
R2_score = r2_score(y_test,pred)*100
print('R2_score:',R2_score)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print('root_mean_squared_error:',np.sqrt(metrics.mean_squared_error(y_test,pred)))

#cross validation score
scores = cross_val_score(DTR, X, y, cv = 10).mean()*100
print("\nCross validation score :", scores)

#difference of accuracy and cv score
diff = R2_score - scores
print("\nR2_Score - Cross Validation Score :", diff)
```

```
R2_score: 67.25296105975163
mean_squared_error: 1971059289.3760684
mean_absolute_error: 29408.91452991453
root_mean_squared_error: 44396.61348995065

Cross validation score : 62.644519590708235

R2_Score - Cross Validation Score : 4.608441469043392
```

Hyper parameter tunning for best model which is ExtraTreesRegressor as there is low difference between accuracy score and corss validation

Hyper parameter tunning

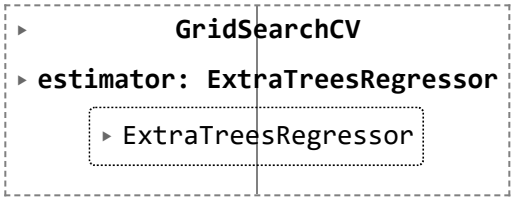
In [122]:
```
# importing necessary libraries
from sklearn.model_selection import GridSearchCV
```

In [123]:
```
parameter = {'n_estimators':[10,100,1000],
             'criterion':['squared_error','mse','absolute_error','mae'],
             'min_samples_split': [1,2,3,4],
             'max_features':['auto','sqrt','log2'],
             'n_jobs':[-2,-1,1,2]}
```

In [124]:
```
GCV=GridSearchCV(ExtraTreesRegressor(),parameter,cv=5)
```

In [125]:
```
GCV.fit(X_train,y_train)
```

Out[125]:



In [126]:
```
GCV.best_params_
```

Out[126]:
```
{'criterion': 'mse',
 'max_features': 'sqrt',
 'min_samples_split': 2,
 'n_estimators': 100,
 'n_jobs': 1}
```

In [127]:
```
Best_mod=ExtraTreesRegressor(criterion='mae',max_features='sqrt',min_samples_spli
Best_mod.fit(X_train,y_train)
pred=Best_mod.predict(X_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print("RMSE value:",np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

```
R2_Score: 88.67339875622513
mean_squared_error: 681753322.4709828
mean_absolute_error: 17045.901994301992
RMSE value: 26110.406401873235
```

Saving the model:

In [128]:
```
# Saving the model using .pkl
import joblib
joblib.dump(Best_mod,"House_PricePrediction.pkl")
```

Out[128]:
```
['House_PricePrediction.pkl']
```

Predicating Price for test dataset using best model saved

In [129]:
```
# Loading the saved model
model=joblib.load("House_PricePrediction.pkl")

#Prediction
prediction = model.predict(X_test)
prediction
```

Out[129]:
```
array([136762.16, 180206.06, 115878.11, 227109.9 , 137573.17,  99927.35,
        97545.  , 376514.9 , 263212.85, 208886.05, 285506.56, 136302.93,
       198890.91, 220415.  , 165489.  , 209925.42, 170518.99, 210229.66,
       159718.07, 155660.61, 167612.15, 362974.11, 204513.  , 231421.05,
       118977.99, 136077.  , 164544.87, 230247.81, 124772.58, 140107.5 ,
       320529.12, 187246.63, 132959.4 , 201898.36, 102538.51, 200526.45,
       159979.58,  91372.59, 158848.37, 205102.74, 220486.83, 214418.24,
       148490.6 , 181768.  , 189592.32, 201345.07, 261276.76, 203457.8 ,
       183636.32, 177230.92, 168117.5 ,  72616.65, 176339.  , 121736.48,
       120991.63, 263350.6 , 291619.13, 150209.  ,  97190.05, 221170.44,
       104375.63,  94957.64, 181442.59, 371601.37, 153976.  , 210377.61,
       306943.38,  85756.76, 152401.8 , 164386.47, 196855.47, 202666.86,
        97852.67, 209645.15, 170595.66, 245307.66, 136324.25, 202498.53,
       256123.95, 140057.05, 467782.45, 125627.28, 199173.34, 196921.29,
       198220.65, 203660.75, 194601.24, 279057.54, 168923.85, 101218.73,
       126446.16, 183634.22, 170380.79, 135187.26, 141997.76, 109575.41,
       124827.06, 138138.65, 240371.99, 123421.75, 149203.56, 300635.17,
       224116.77, 191626.27, 309260.89, 272764.41, 198897.  , 209662.82,
       116062.42, 114424.1 , 143324.87, 213174.35, 203610.43, 191245.11,
       186353.5 , 147473.99, 140403.16, 271313.12, 213667.44, 220370.21,
       145094.75, 142392.77, 214055.56, 141797.3 , 194927.8 , 243042.31,
       216457.85, 168381.09, 149979.41, 295236.72, 186802.73, 360380.93,
       237271.27, 313734.09,  89132.56, 199855.62, 207456.99, 182912.37,
       181770.35, 134479.71, 180738.07, 375105.74, 221806.  , 164161.73,
       205124.24, 102755.01, 119750.46, 245797.18, 103802.9 , 195314.9 ,
       159281.09, 144265.4 , 122604.66, 231538.61, 128168.04, 133403.5 ,
       147485.35, 283604.58, 151946.5 , 137881.94, 200638.1 , 228085.54,
       115866.5 , 295966.65, 121331.01, 184134.89,  97099.  , 120237.21,
       316362.33, 175569.88, 135969.66, 142472.54, 129142.03, 130349.75,
       236367.63, 124397.3 , 179157.  , 221165.25, 149502.36, 250201.53,
       134454.62, 118368.85, 101950.5 , 127596.24, 131103.26, 217827.7 ,
       169824.14, 142558.37, 129848.71, 128419.99, 199048.12, 112316.84,
       120410.18, 187575.64, 232723.2 , 133901.5 , 112470.5 , 125424.  ,
       120321.1 , 389870.15, 260836.09, 155143.39, 130304.47, 118834.3 ,
       151591.52, 178954.64, 173971.7 , 149614.87, 117216.18, 195663.67,
       113027.54, 186880.33, 127191.39, 116098.09, 257009.02, 129513.11,
       151098.  , 121651.  , 142568.71, 146012.52, 143486.73, 205282.03,
       260487.74, 135062.  , 200584.23, 282346.58, 209624.45, 296667.93,
       148687.78, 147041.1 , 132695.68, 395839.51, 188749.05, 132391.35,
       118130.19, 147096.25, 287852.64, 242621.77, 336453.39, 190845.59,
       212794.58, 190406.34, 123782.46, 195393.01, 145804.65, 205185.39,
       122905.37, 143806.74, 172467.53, 156573.07, 167252.69, 130938.  ,
       163781.32, 126219.99, 164324.  , 330414.56, 158540.24, 157222.  ,
       134206.11, 201043.52, 125696.83, 128040.78, 285794.  , 203215.85,
       119689.49, 181052.47, 120987.83, 247154.4 , 192185.9 , 118257.85,
       175905.73, 122844.69, 161906.72, 161328.19, 386323.25, 218364.89,
       158507.12, 139981.25, 244417.27, 191994.15, 252324.16, 310227.08,
       276676.67, 199634.75, 259441.22, 204456.86, 122152.38, 143942.74,
       137347.35, 182793.99, 358048.46, 193631.01, 282462.56, 276243.65,
       141687.48, 218452.24, 124144.74, 177748.74, 131297.69, 149889.74,
```

```
151436.84, 126304.  , 206995.92, 172143.87, 264066.4 , 325236.25,
162162.57, 148707.25, 157650.11, 433860.01, 196843.  , 295637.11,
181405.43,  83156.01, 381260.63, 178023.46, 154621.82, 207511.4 ,
160880.14, 221052.01, 104369.5 , 184292.37, 175803.35, 169258.44,
120502.76, 241475.02, 144136.5 , 140244.62, 298410.78, 247781.73,
393870.3 , 151865.93, 240233.52, 137086.44, 173755.87, 200248.77,
141513.14, 141342.4 , 138866.51, 223217.9 , 135987.5 , 110843.44,
116481.3 , 151843.23, 139413.52, 120108.32, 126327.05, 154111.76,
179229.64, 120082.58, 339508.27])
```

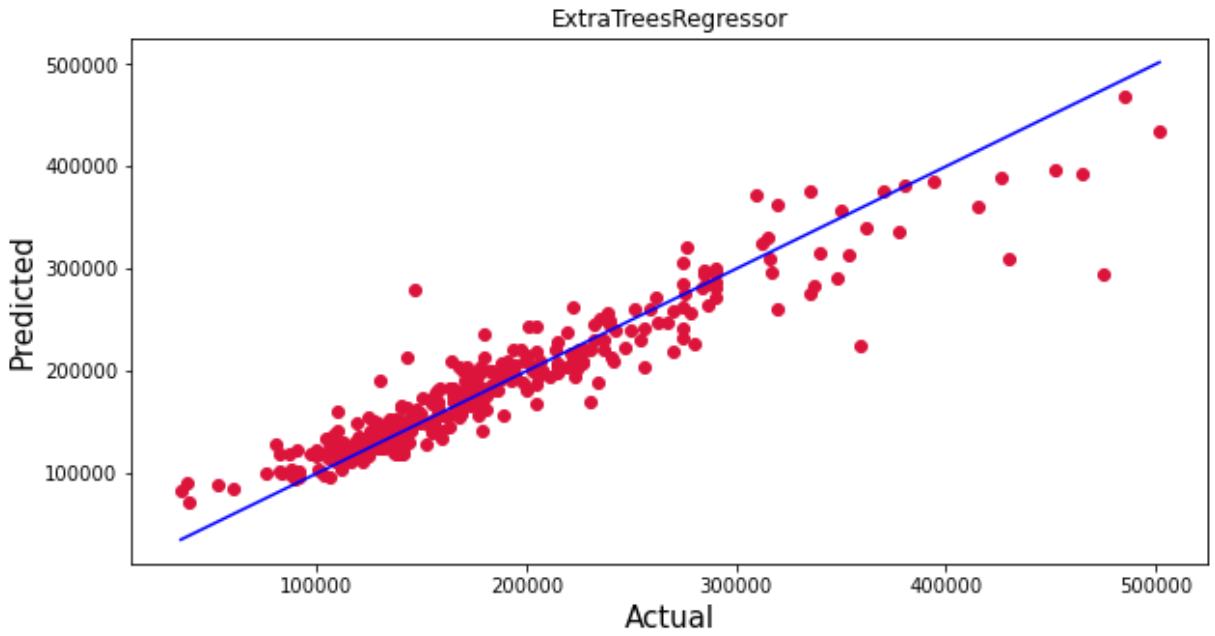
In [130]:
```
pd.DataFrame([model.predict(X_test)[:],y_test[:]],index=["Predicted","Actual"])
```

Out[130]:

	0	1	2	3	4	5	6	7	
Predicted	136762.16	180206.06	115878.11	227109.9	137573.17	99927.35	97545.0	376514.9	2632
Actual	137000.00	168500.00	115000.00	280000.0	140000.00	76000.00	88000.0	335000.0	2220

As we can see comparision between actual and predicated values they are very close

In [131]:
```
plt.figure(figsize=(10,5))
plt.scatter(y_test, prediction, c='crimson')
p1 = max(max(prediction), max(y_test))
p2 = min(min(prediction), min(y_test))
plt.plot([p1, p2], [p1, p2], 'b-')
plt.xlabel('Actual', fontsize=15)
plt.ylabel('Predicted', fontsize=15)
plt.title("ExtraTreesRegressor")
plt.show()
```



We can see pink dots they are very close to our line

Perdiction house price for separatly provided test data set

In [132]:
```
Predicted_Sale_Price=model.predict(X_1)
Predicted_Sale_Price
```

Out[132]:
```
array([332375.55, 210515.57, 244111.5 , 170349.72, 264228.96,  91175.  ,
       153622.8 , 328121.36, 236679.95, 165715.9 ,  94208.22, 142926.87,
       123363.06, 187514.81, 282856.63, 134896.67, 119423.09, 135284.5 ,
       175886.52, 208061.28, 144458.23, 164303.31, 157639.24, 101926.  ,
       114056.71, 132900.84, 179590.23, 150975.58, 185619.26, 106623.45,
       138505.16, 203683.51, 229936.11, 158121.  , 127753.04, 181072.22,
       204193.43, 124580.04, 161294.37, 154530.5 , 123046.89, 276113.44,
       200641.89, 200605.49, 150195.8 , 128344.5 , 129353.75, 112496.37,
       211840.01, 343669.98, 148008.83, 214957.77, 115263.26, 104656.67,
       274758.19, 137184.74, 152091.87, 182005.75, 118347.74, 256812.62,
        99696.87, 195395.74, 137580.9 , 154910.86, 212907.84,  98534.5 ,
       153506.  , 205562.74, 150331.5 , 158771.5 , 268638.68, 175085.97,
       164537.13, 156126.06, 147674.89, 220835.86, 312666.74, 193096.07,
       288047.96, 149886.  , 214812.77, 138758.55, 147166.74, 154542.69,
       197703.41, 211606.48, 127258.81, 335848.16, 152783.7 , 181927.01,
       248157.13, 142012.93, 139184.51, 139490.27, 194850.91, 157983.75,
       247038.06, 176569.17, 347803.84, 131058.27, 244789.33, 120214.79,
       138725.64, 179821.39, 188379.45, 149046.68, 265709.01, 156854.37,
       199158.09, 200106.44, 214927.12, 169270.  , 166245.42, 239360.76,
       130740.5 , 107791.07, 139093.43, 191195.11, 146423.58, 117635.94,
       107717.75, 202742.02, 208256.66, 138236.84, 135819.37, 182404.5 ,
       128651.58, 158444.2 , 102901.78, 124143.01, 155975.33, 220191.04,
       146042.25, 173748.5 , 155344.61, 324323.82, 215530.99, 121515.13,
       261311.92, 138397.65, 144215.51, 371429.5 , 108333.86, 331660.46,
       166314.52, 216020.82, 160279.98, 119740.85, 112707.43, 204569.79,
       177654.49, 145849.25, 199811.5 , 122459.48, 101750.35, 169088.85,
       182610.11, 186981.38, 137015.84, 182392.4 , 206430.76, 147309.9 ,
       200357.25, 128912.41, 116405.98, 273399.92, 185030.24, 204989.72,
       132036.25, 210579.3 , 173775.34, 134169.09, 148002.64, 269173.26,
       135493.5 , 358131.98, 129196.  , 116720.53, 162501.5 , 151980.91,
       196696.63, 172257.05, 267324.05, 170973.5 , 417166.32, 342506.42,
       192620.46, 124078.26, 169461.39, 156457.96, 113310.38, 207976.29,
       204756.71, 104878.  , 138465.69,  92303.  , 167025.72, 204435.85,
       126977.77, 177124.96, 154893.96, 116733.51, 248785.79, 282969.37,
       132996.29, 140814.  , 256338.42, 147807.  , 130801.71, 137428.95,
       113451.34, 152670.  , 139527.5 , 177600.95, 101036.17, 138444.58,
       151294.4 , 135857.58, 121584.25, 181674.27, 210043.79, 226892.24,
       286206.02, 134583.5 , 203211.82, 329605.83, 222074.92, 113466.3 ,
       364024.83, 186123.62, 115624.53, 166554.5 , 116402.8 , 138537.66,
       127515.52, 216325.11, 181601.87, 161753.42, 247982.51, 201547.1 ,
       231274.89, 149663.59, 265505.82, 102579.24, 150341.63, 250291.98,
       204451.69, 266589.02, 134596.85, 149579.26, 158759.1 , 231934.68,
       150474.24, 120094.46, 122904.71, 194394.78, 116477.1 , 412639.13,
       188499.39, 121000.6 , 188976.62, 185840.24, 111811.09, 139784.11,
       211489.85, 184670.75, 115469.62, 173545.11, 247803.05, 234883.03,
       151754.54, 137764.82, 350889.43, 224722.24, 289870.71, 210641.56,
       180401.12, 133751.71, 190484.8 , 384617.21, 133390.2 , 332994.83,
       143134.74, 317966.3 , 150809.61, 132096.5 , 172273.74, 246916.3 ,
       146488.58, 163312.48, 168632.91, 100646.4 ])
```

In [133]:
```
# Making dataframe for predicted SalePrice
House_Price_Predictions=pd.DataFrame()
House_Price_Predictions["SalePrice"]=Predicted_Sale_Price
House_Price_Predictions.head(10)
```

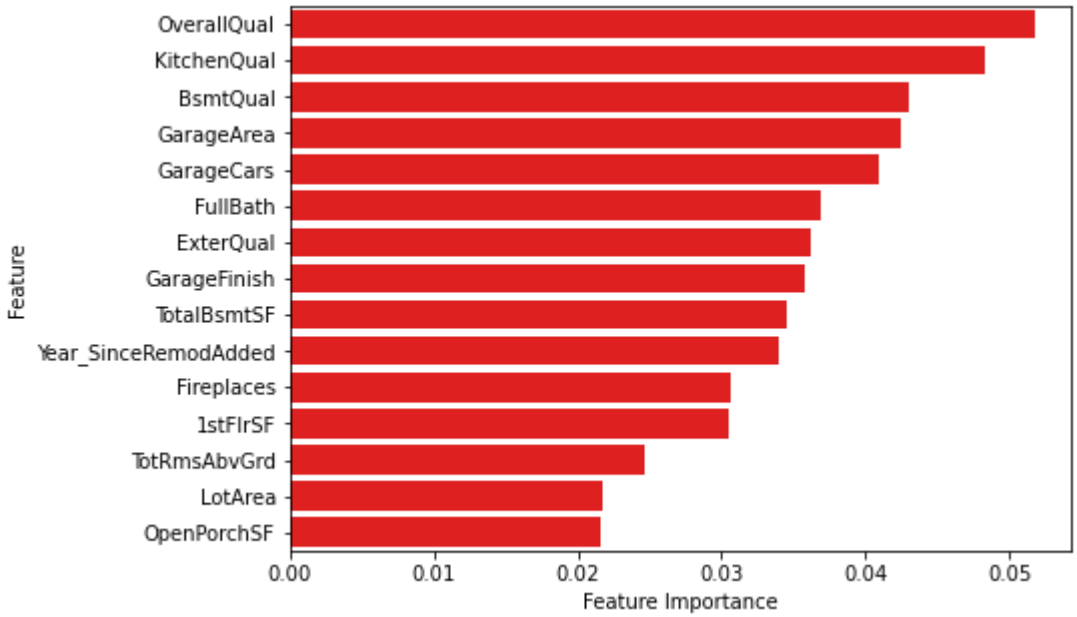
Out[133]:

	SalePrice
0	332375.55
1	210515.57
2	244111.50
3	170349.72
4	264228.96
5	91175.00
6	153622.80
7	328121.36
8	236679.95
9	165715.90

In [134]:
```
# lets save the predictions to csv
House_Price_Predictions.to_csv("House_Price_Predictions.csv",index=False)
```

In [136]:
```
Extr_feature_importances = Best_mod.feature_importances_
Extr_feature_importances = pd.Series(Extr_feature_importances,
                                     index=X_train.columns.values).sort_values(asc

fig, ax = plt.subplots(figsize=(7,5))
sns.barplot(x=Extr_feature_importances, y=Extr_feature_importances.index, color='
plt.xlabel('Feature Importance');
plt.ylabel('Feature');
```



Thank you

In [ ]: