In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:

```
import warnings
import random

warnings.filterwarnings('ignore')
seed = 42
```

1. Data Import

In [3]:

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

2. Data Analysis

In [4]:

```
train.head()
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl

5 rows × 81 columns

In [5]:

```
test.head()
```

Out[5]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lv
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lv
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lv
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lv
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS

5 rows × 80 columns

In [6]:

```
train.shape, test.shape
```

Out[6]:

```
((1460, 81), (1459, 80))
```

In [7]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1460 non-null   int64  
 1   MSSubClass     1460 non-null   int64  
 2   MSZoning       1460 non-null   object 
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64  
 5   Street         1460 non-null   object 
 6   Alley          91 non-null     object 
 7   LotShape       1460 non-null   object 
 8   LandContour    1460 non-null   object 
 9   Utilities      1460 non-null   object 
 10  LotConfig      1460 non-null   object 
 11  LandSlope      1460 non-null   object 
 12  Neighborhood   1460 non-null   object 
 13  Condition1     1460 non-null   object 
 14  Condition2     1460 non-null   object 
 15  BldgType       1460 non-null   object 
 16  HouseStyle     1460 non-null   object 
 17  OverallQual    1460 non-null   int64  
 18  OverallCond    1460 non-null   int64  
 19  YearBuilt      1460 non-null   int64  
 20  YearRemodAdd   1460 non-null   int64  
 21  RoofStyle      1460 non-null   object 
 22  RoofMatl       1460 non-null   object 
 23  Exterior1st    1460 non-null   object 
 24  Exterior2nd    1460 non-null   object 
 25  MasVnrType     1452 non-null   object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
 55  Functional     1460 non-null   object 
```

```
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

In [8]:

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             1459 non-null   int64  
 1   MSSubClass     1459 non-null   int64  
 2   MSZoning       1455 non-null   object 
 3   LotFrontage    1232 non-null   float64
 4   LotArea        1459 non-null   int64  
 5   Street         1459 non-null   object 
 6   Alley          107 non-null    object 
 7   LotShape       1459 non-null   object 
 8   LandContour    1459 non-null   object 
 9   Utilities      1457 non-null   object 
 10  LotConfig      1459 non-null   object 
 11  LandSlope      1459 non-null   object 
 12  Neighborhood   1459 non-null   object 
 13  Condition1     1459 non-null   object 
 14  Condition2     1459 non-null   object 
 15  BldgType       1459 non-null   object 
 16  HouseStyle     1459 non-null   object 
 17  OverallQual    1459 non-null   int64  
 18  OverallCond    1459 non-null   int64  
 19  YearBuilt      1459 non-null   int64  
 20  YearRemodAdd   1459 non-null   int64  
 21  RoofStyle      1459 non-null   object 
 22  RoofMatl       1459 non-null   object 
 23  Exterior1st    1458 non-null   object 
 24  Exterior2nd    1458 non-null   object 
 25  MasVnrType     1443 non-null   object 
 26  MasVnrArea     1444 non-null   float64
 27  ExterQual      1459 non-null   object 
 28  ExterCond      1459 non-null   object 
 29  Foundation     1459 non-null   object 
 30  BsmtQual       1415 non-null   object 
 31  BsmtCond       1414 non-null   object 
 32  BsmtExposure   1415 non-null   object 
 33  BsmtFinType1   1417 non-null   object 
 34  BsmtFinSF1     1458 non-null   float64
 35  BsmtFinType2   1417 non-null   object 
 36  BsmtFinSF2     1458 non-null   float64
 37  BsmtUnfSF      1458 non-null   float64
 38  TotalBsmtSF    1458 non-null   float64
 39  Heating        1459 non-null   object 
 40  HeatingQC      1459 non-null   object 
 41  CentralAir     1459 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1459 non-null   int64  
 44  2ndFlrSF       1459 non-null   int64  
 45  LowQualFinSF   1459 non-null   int64  
 46  GrLivArea      1459 non-null   int64  
 47  BsmtFullBath   1457 non-null   float64
 48  BsmtHalfBath   1457 non-null   float64
 49  FullBath       1459 non-null   int64  
 50  HalfBath       1459 non-null   int64  
 51  BedroomAbvGr   1459 non-null   int64  
 52  KitchenAbvGr   1459 non-null   int64  
 53  KitchenQual    1458 non-null   object 
 54  TotRmsAbvGrd   1459 non-null   int64  
 55  Functional     1457 non-null   object 
```

```
 56  Fireplaces     1459 non-null   int64
 57  FireplaceQu    729 non-null    object
 58  GarageType     1383 non-null   object
 59  GarageYrBlt    1381 non-null   float64
 60  GarageFinish   1381 non-null   object
 61  GarageCars     1458 non-null   float64
 62  GarageArea     1458 non-null   float64
 63  GarageQual     1381 non-null   object
 64  GarageCond     1381 non-null   object
 65  PavedDrive     1459 non-null   object
 66  WoodDeckSF     1459 non-null   int64
 67  OpenPorchSF    1459 non-null   int64
 68  EnclosedPorch  1459 non-null   int64
 69  3SsnPorch      1459 non-null   int64
 70  ScreenPorch    1459 non-null   int64
 71  PoolArea       1459 non-null   int64
 72  PoolQC         3 non-null      object
 73  Fence          290 non-null    object
 74  MiscFeature    51 non-null     object
 75  MiscVal        1459 non-null   int64
 76  MoSold         1459 non-null   int64
 77  YrSold         1459 non-null   int64
 78  SaleType       1458 non-null   object
 79  SaleCondition  1459 non-null   object
dtypes: float64(11), int64(26), object(43)
memory usage: 912.0+ KB
```

In [9]:

```
init_nums = train.dtypes[train.dtypes!='object'].index
init_nums
```

Out[9]:

```
Index(['Id', 'MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinS
F1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullB
ath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckS
F',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolAr
ea',
       'MiscVal', 'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

In [10]:

```
#After manually going through each columns in init_num, this is the outcome

num_countable = ['MSSubClass', 'OverallQual', 'OverallCond','BedroomAbvGr', 'KitchenAbvGr
                 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars',
                'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath']

num_cols = ['LotFrontage','LotArea','MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF'
           'LowQualFinSF', 'GrLivArea', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'Enclo
            '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal']

date_cols = ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'MoSold', 'YrSold']
```

In [11]:

```
init_cats = train.dtypes[train.dtypes == 'object'].index
init_cats
```

Out[11]:

```
Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilitie
s',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition
2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundatio
n',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinTyp
e2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQ
ual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

In [12]:

```
train[num_cols]
```

Out[12]:

	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF
0	65.0	8450	196.0	706	0	150	856
1	80.0	9600	0.0	978	0	284	1262
2	68.0	11250	162.0	486	0	434	920
3	60.0	9550	0.0	216	0	540	756
4	84.0	14260	350.0	655	0	490	1145
...	...	...	...	...	...	...	...
1455	62.0	7917	0.0	0	0	953	953
1456	85.0	13175	119.0	790	163	589	1542
1457	66.0	9042	0.0	275	0	877	1152
1458	68.0	9717	0.0	49	1029	0	1078
1459	75.0	9937	0.0	830	290	136	1256

1460 rows × 19 columns

In [13]:

```
train[num_cols].isnull().sum()
```

Out[13]:

```
LotFrontage      259
LotArea            0
MasVnrArea         8
BsmtFinSF1         0
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
GarageArea         0
WoodDeckSF         0
OpenPorchSF        0
EnclosedPorch      0
3SsnPorch          0
ScreenPorch        0
PoolArea           0
MiscVal            0
dtype: int64
```

- LotFontage has almost 18% of missing data.
- MasVnrArea has 8 only 8 rows missing which can be managed easily.

In [14]:

```
train[num_cols].isin([0]).sum()
```

Out[14]:

```
LotFrontage         0
LotArea             0
MasVnrArea        861
BsmtFinSF1        467
BsmtFinSF2       1293
BsmtUnfSF         118
TotalBsmtSF        37
1stFlrSF            0
2ndFlrSF          829
LowQualFinSF     1434
GrLivArea           0
GarageArea         81
WoodDeckSF        761
OpenPorchSF       656
EnclosedPorch    1252
3SsnPorch        1436
ScreenPorch      1344
PoolArea         1453
MiscVal          1408
dtype: int64
```

- BsmtFinSF2, LowQualFinSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal has huge amount of rows filled with zeros.

In [15]:

```
cat_cols = list(init_cats)
```

In [16]:

```
cat_cols
```

Out[16]:

```
['MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']
```

In [17]:

```
train[cat_cols]
```

Out[17]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neigh
0	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	
1	RL	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	
2	RL	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	
3	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	
4	RL	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	
...	...	...	...	...	...	...	...	...	
1455	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	
1456	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	
1457	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	
1458	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	
1459	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	

1460 rows × 43 columns

In [18]:

```
train[cat_cols].isnull().sum()
```

Out[18]:

```
MSZoning           0
Street             0
Alley           1369
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         8
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Heating            0
HeatingQC          0
CentralAir         0
Electrical         1
KitchenQual        0
Functional         0
FireplaceQu      690
GarageType        81
GarageFinish      81
GarageQual        81
GarageCond        81
PavedDrive         0
PoolQC          1453
Fence           1179
MiscFeature     1406
SaleType           0
SaleCondition      0
dtype: int64
```

- Alley, FireplaceQu, PolQC, Fence, MiscFeature has significant number of missing values.

In [19]:

```
train[num_countable]
```

Out[19]:

	MSSubClass	OverallQual	OverallCond	BedroomAbvGr	KitchenAbvGr	TotRmsAbvGrd
0	60	7	5	3	1	8
1	20	6	8	3	1	6
2	60	7	5	3	1	6
3	70	7	5	3	1	7
4	60	8	5	4	1	9
...	...	...	...	...	...	...
1455	60	6	5	3	1	7
1456	20	6	6	3	1	7
1457	70	7	9	4	1	9
1458	20	5	6	2	1	5
1459	20	5	6	3	1	6

1460 rows × 12 columns

In [20]:

```
train[num_countable].isnull().sum()
```

Out[20]:

```
MSSubClass      0
OverallQual     0
OverallCond     0
BedroomAbvGr    0
KitchenAbvGr    0
TotRmsAbvGrd    0
Fireplaces      0
GarageCars      0
BsmtFullBath    0
BsmtHalfBath    0
FullBath        0
HalfBath        0
dtype: int64
```

In [21]:

```
train[date_cols]
```

Out[21]:

	YearBuilt	YearRemodAdd	GarageYrBlt	MoSold	YrSold
0	2003	2003	2003.0	2	2008
1	1976	1976	1976.0	5	2007
2	2001	2002	2001.0	9	2008
3	1915	1970	1998.0	2	2006
4	2000	2000	2000.0	12	2008
...	...	...	...	...	...
1455	1999	2000	1999.0	8	2007
1456	1978	1988	1978.0	2	2010
1457	1941	2006	1941.0	5	2010
1458	1950	1996	1950.0	4	2010
1459	1965	1965	1965.0	6	2008

1460 rows × 5 columns

In [22]:

```
train[date_cols].isnull().sum()
```

Out[22]:

```
YearBuilt        0
YearRemodAdd     0
GarageYrBlt     81
MoSold           0
YrSold           0
dtype: int64
```

- GarageYrBlt has 81 rows of mising value, does that mean garage is not there?

Descriptive statistics of the numeric columns

In [23]:

```
train[num_cols].describe().T
```

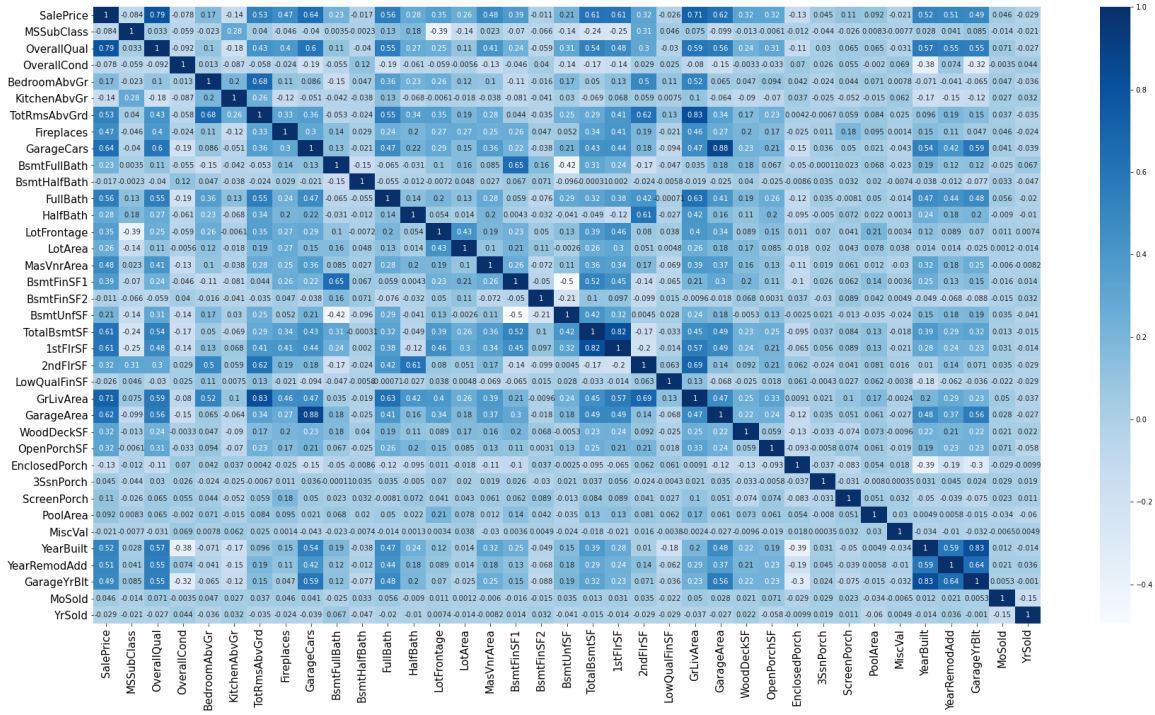
Out[23]:

	count	mean	std	min	25%	50%	75%	ma
LotFrontage	1201.0	70.049958	24.284752	21.0	59.00	69.0	80.00	313.
LotArea	1460.0	10516.828082	9981.264932	1300.0	7553.50	9478.5	11601.50	215245.
MasVnrArea	1452.0	103.685262	181.066207	0.0	0.00	0.0	166.00	1600.
BsmtFinSF1	1460.0	443.639726	456.098091	0.0	0.00	383.5	712.25	5644.
BsmtFinSF2	1460.0	46.549315	161.319273	0.0	0.00	0.0	0.00	1474.
BsmtUnfSF	1460.0	567.240411	441.866955	0.0	223.00	477.5	808.00	2336.
TotalBsmtSF	1460.0	1057.429452	438.705324	0.0	795.75	991.5	1298.25	6110.
1stFlrSF	1460.0	1162.626712	386.587738	334.0	882.00	1087.0	1391.25	4692.
2ndFlrSF	1460.0	346.992466	436.528436	0.0	0.00	0.0	728.00	2065.
LowQualFinSF	1460.0	5.844521	48.623081	0.0	0.00	0.0	0.00	572.
GrLivArea	1460.0	1515.463699	525.480383	334.0	1129.50	1464.0	1776.75	5642.
GarageArea	1460.0	472.980137	213.804841	0.0	334.50	480.0	576.00	1418.
WoodDeckSF	1460.0	94.244521	125.338794	0.0	0.00	0.0	168.00	857.
OpenPorchSF	1460.0	46.660274	66.256028	0.0	0.00	25.0	68.00	547.
EnclosedPorch	1460.0	21.954110	61.119149	0.0	0.00	0.0	0.00	552.
3SsnPorch	1460.0	3.409589	29.317331	0.0	0.00	0.0	0.00	508.
ScreenPorch	1460.0	15.060959	55.757415	0.0	0.00	0.0	0.00	480.
PoolArea	1460.0	2.758904	40.177307	0.0	0.00	0.0	0.00	738.
MiscVal	1460.0	43.489041	496.123024	0.0	0.00	0.0	0.00	15500.

Lets also have a glance of all the numeric columns and their dependencies on each other.

In [24]:

```
all_num_col = ['SalePrice'] + num_countable + num_cols + date_cols

plt.figure(figsize=(30, 16))
corr = train[all_num_col].corr()
sns.heatmap(corr, annot=True, cmap='Blues')
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



Correlation exists between different features

High Correlation

- TotRmsAbvGrd vs GrLivArea - 0.83
- GarageCars vs GarageArea - 0.88
- 1stFlrSF vs TotalBsmtSF - 0.82
- GarageYrBlt vs YearBuilt - 0.83

Moderate Correlation

- TotRomsAbvGrd vs BedroomAbvGr - 0.68
- GrLivArea vs 2ndFlrSF - 0.69
- BsmtFullBath vs BsmtFinSF1 - 0.65
- GarageYrBlt vs YearRemodAdd - 0.64
- GrLivArea vs FullBath - 0.63
- TotRmsAbvGrd vs 2ndFlrSF - 0.62
- 2ndFlrSF vs HalfBath - 0.61
- OverallQual vs GarageCars - 0.6
- GrLivArea vs OverallQual - 0.59
- GarageArea vs OverallQual - 0.59
- YearRemodAdd vs YearBuilt - 0.59
- 1stFlrSF vs GrLivArea - 0.57
- FullBath vs OverallQual - 0.55

- FullBath vs TotRmsAbvGrd - 0.55
- TotalBsmtSF vs OverallQual - 0.54

Decision

- GarageArea is sufficient to give information to give idea about Garage, hence GarageCars can be removed.
- GrLivArea can be used alone to denote living area. TotRmsAbvGrd can be removed and BedroomsAbvGr can be removed as well.
- If area of first floor depends on basement area then lets remove 1stFlrSF.

In [25]:

```
remove_cols = ['GarageCars', 'TotRmsAbvGrd', 'BedroomAbvGr', '1stFlrSF']
```

In [26]:

```
for col in remove_cols:
    if col in num_countable:
        num_countable.remove(col)
    if col in num_cols:
        num_cols.remove(col)

num_cols
```

Out[26]:

```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'PoolArea',
 'MiscVal']
```

In [27]:

```
num_countable
```

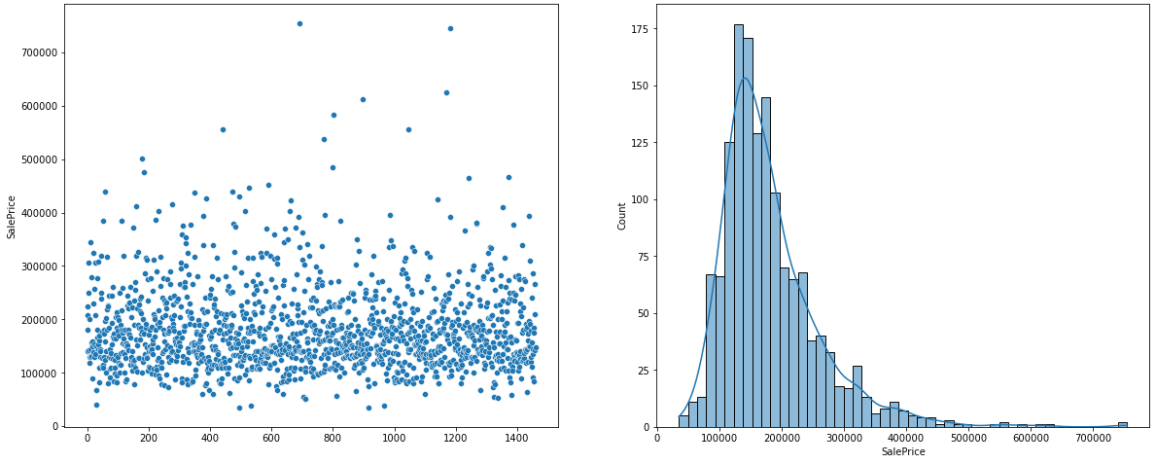
Out[27]:

```
['MSSubClass',
 'OverallQual',
 'OverallCond',
 'KitchenAbvGr',
 'Fireplaces',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath']
```

Visualizing the target

In [28]:

```
#Plotting the target

fig, ax = plt.subplots(1, 2, figsize=(20, 8))

sns.scatterplot(y = 'SalePrice', x = train.index, data=train, ax=ax[0])
sns.histplot(x='SalePrice', data=train, kde=True, ax=ax[1])
fig.show()
```



In [29]:

```
def dist_plot(df, columns, kind='scatter', label='SalePrice', basis=None):
    plt.figure(figsize=(20, 16))

    for idx, column in enumerate(columns):
        f = plt.subplot(5, 4, idx+1)
        if kind == 'scatter':
            g = sns.scatterplot(x = column, y = label, data = df, hue=basis)
        elif (kind == 'histplot'):
            g = sns.histplot(x = column, data = df, hue=basis, kde=True)
        elif (kind =='boxplot'):
            g = sns.boxplot(x = column, data=df, hue=basis)
    plt.tight_layout()

def count_plot(df, columns, label='SalePrice', kind = 'countplot', basis=None):
    plt.figure(figsize=(24, 18))

    for idx, column in enumerate(columns):
        f = plt.subplot(6, 4, idx+1)

        if kind == 'countplot':
            g= sns.countplot(x = column, data = df, hue=basis)

        if kind == 'barplot':
            g= sns.barplot(x = column, y=label, data = df, hue=basis)
    plt.tight_layout()
```

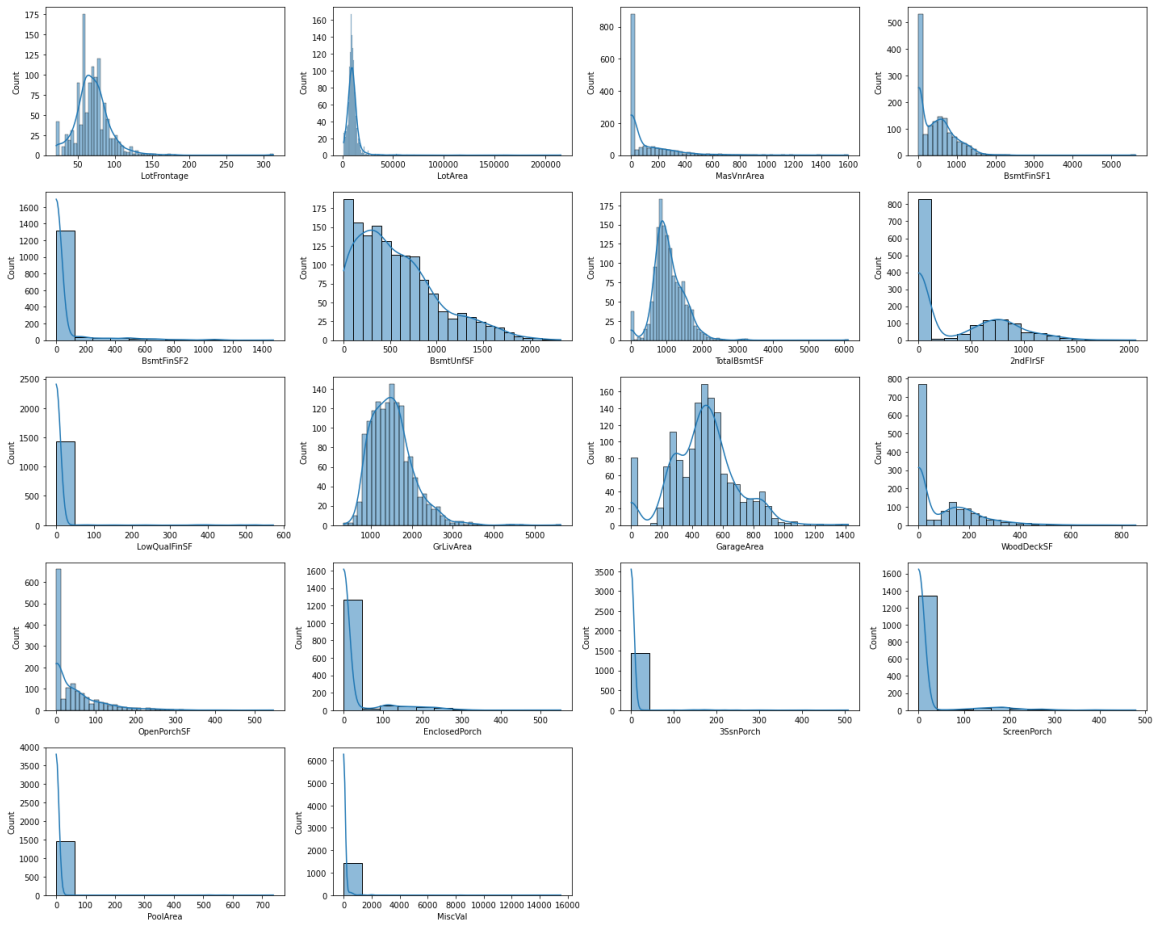
Analyzing the numeric features

In [30]:

```
dist_plot(train, num_cols)
```

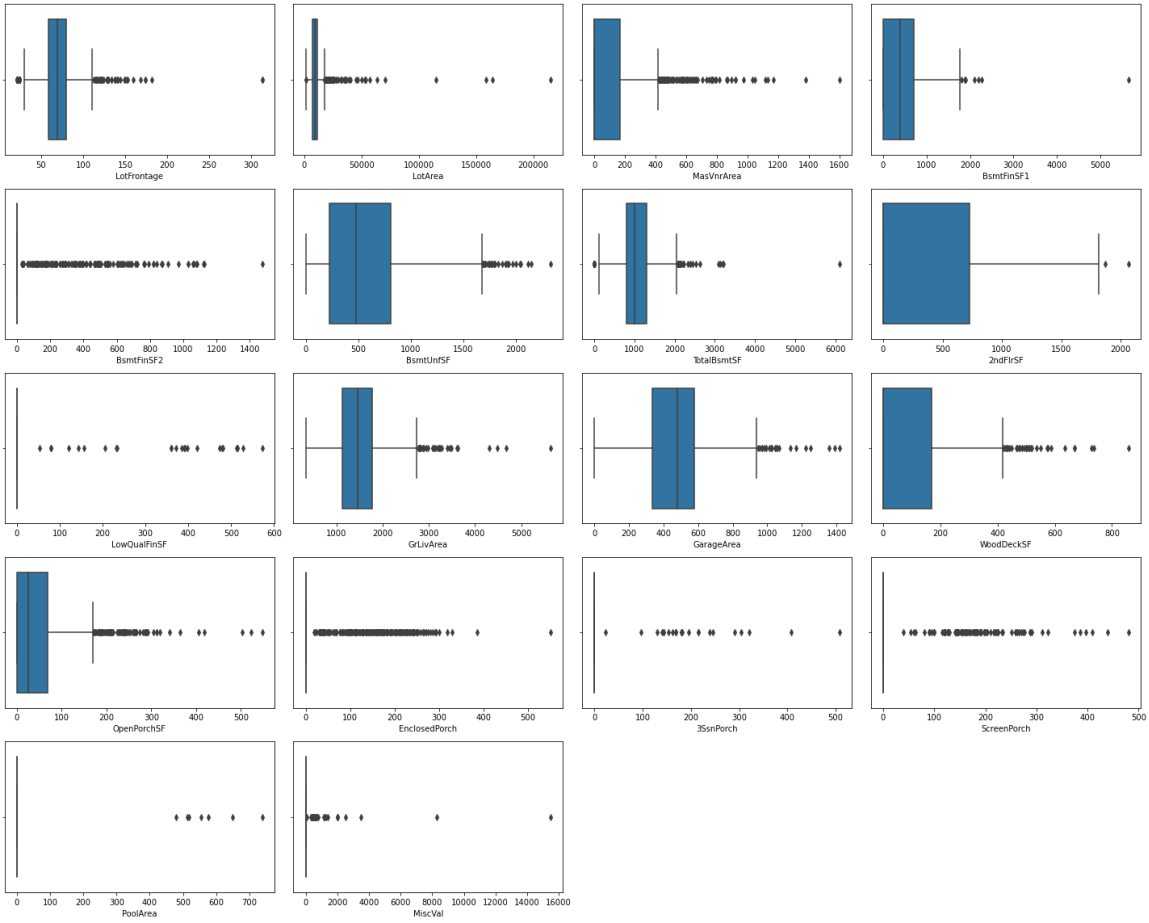
In [31]:

```
dist_plot(train, num_cols, kind='histplot')
```



In [32]:

```
dist_plot(train, num_cols, kind='boxplot')
```



Observations - Numeric Features

- MiscVal - has very few values and majority of them are zeros, do not see any specific relation with target as well. Need to decide whether we really require this column. At most, we can add a new column say 'HasSpecialFeature' and populate 0,1 while removing MiscVal.
- PoolArea - same logic as MiscVal goes for PoolArea i.e., add column HasPool & remove PoolArea
- With similar logic, lets add 3 new columns HasEnclosedPorch, Has3SsnPorch, HasScreenPorch
- OpenPorchSF - More than 600 rows contain zero, that means roughly 800 rows has value for OpenPorchSF. Maybe create a column 'HaveOpenPorch'with (0,1) values.
- Similar logic goes for 'WoodDeckSF' i.e., remove WoodDeckSF and create 'HasWoodDeck' with (0,1)
- GarageArea seems to have a linear relationship with the target. The zeros can be replaced with the median value. We will treat the outliers seperately.
- Same logic goes for GrLivArea.
- Majority of the rows are zeros for LowQualFinSF, can we delete it?.
- 2ndFlrSF also has same thing i.e., huge number of values are zeros.
- Will keep BsmtUnfSF & TotalBsmtSF as is and treat the zeros in usual manner.
- For BsmtFinSF1, BsmtFinSF2 lets create a new column HasType1Finishing & HasType2Finishing columns
- Lets do the same as previous for MasVnrArea.
- Keeping LotArea as is. Need to do further research.

Strategy

1. First we will delete the unwanted columns

2. Acordingly will add the new column set for the conditions discovered.

In [33]:

```
#remove_cols = ['GarageCars', 'TotRmsAbvGrd', 'BedroomAbvGr', '1stFlrSF']
cols_to_del = ['MiscVal', 'PoolArea', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch','OpenPo
               'LowQualFinSF', '2ndFlrSF', 'BsmtFinSF1', 'BsmtFinSF2', 'MasVnrArea']

for col in cols_to_del:
    remove_cols.append(col)

remove_cols
```

Out[33]:

```
['GarageCars',
 'TotRmsAbvGrd',
 'BedroomAbvGr',
 '1stFlrSF',
 'MiscVal',
 'PoolArea',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'OpenPorchSF',
 'WoodDeckSF',
 'LowQualFinSF',
 '2ndFlrSF',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'MasVnrArea']
```

In [34]:

```
# new_col_rpl = {'MiscVal': 'HasSpecialFeature', 'PoolArea': 'HasPoolArea', 'EnclosedPorc
#                '3SsnPorch': 'Has3SsnPorch', 'ScreenPorch': 'HasScreenPorch', 'OpenPorch'
#                'LowQualSF': 'HasLowQual', '2ndFlrSF': 'Has2ndFlr', 'BsmtFinSF1': 'HasTyp
#                 'BsmtFinSF2': 'HasType2Finishing', 'MasVnrArea': 'HasMasVnrArea'}
```

Analyzing the countable features

In [35]:

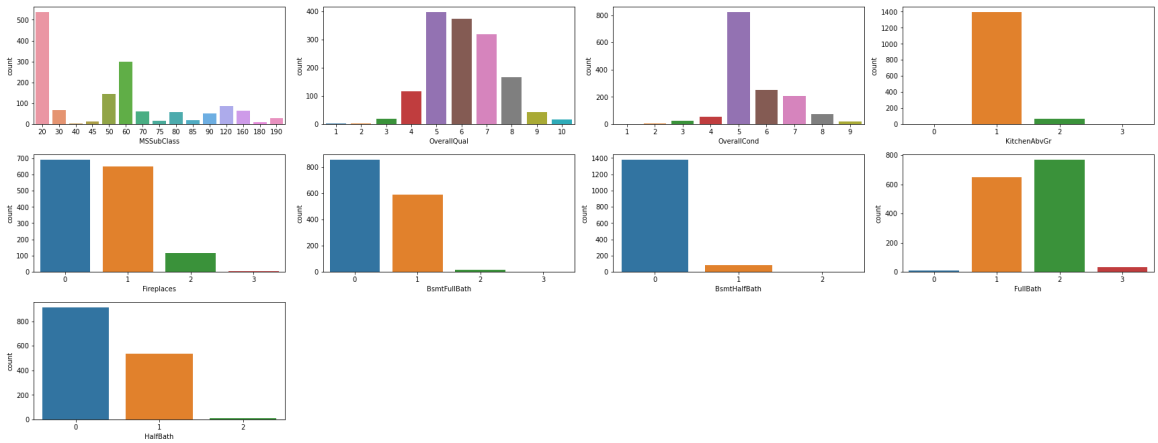
```
num_countable
```

Out[35]:

```
['MSSubClass',
 'OverallQual',
 'OverallCond',
 'KitchenAbvGr',
 'Fireplaces',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath']
```

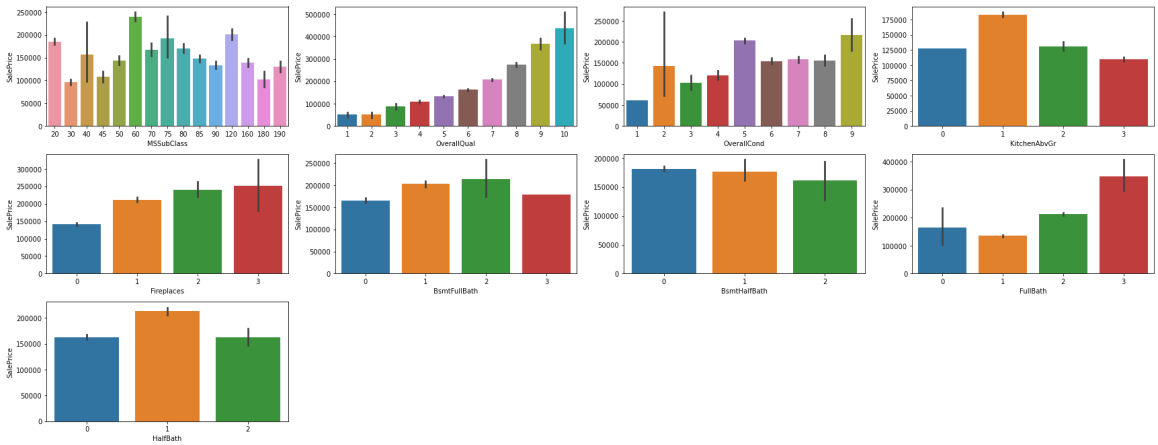
In [36]:

```
count_plot(train, num_countable)
```



In [37]:

```
count_plot(train, num_countable, kind='barplot')
```



Obesrvations - Countable Features

- Type of dwelling i.e., MSSubclass does not change SalePrice significantly.
- With OverallQual, SalePrice increases which is a general phenomena and evident here.
- Better the condition, more the price but average condition i.e., 5 has more price than (6, 7, 8) and their prices are almost almost same. Any reason?
- More no. of kitchen does not gurantee more price.
- This is interesting, more fireplaces means more price. Although number of homes with more than 1 fireplaces is very less.
- More than 1 BsmtFullBath is rare. Can we do anything about it?
- Similar thing is observed for BsmtHalfBath. More than 0 is rare and doesn't effect the price that much.
- FullBath also has similar traits. Fullbath 0 and 3 is rare but has significant price impact though.
- 1 Halfbath means more price.

Analyzing categorical features

In [38]:

```
train[cat_cols].isnull().sum()
```

Out[38]:

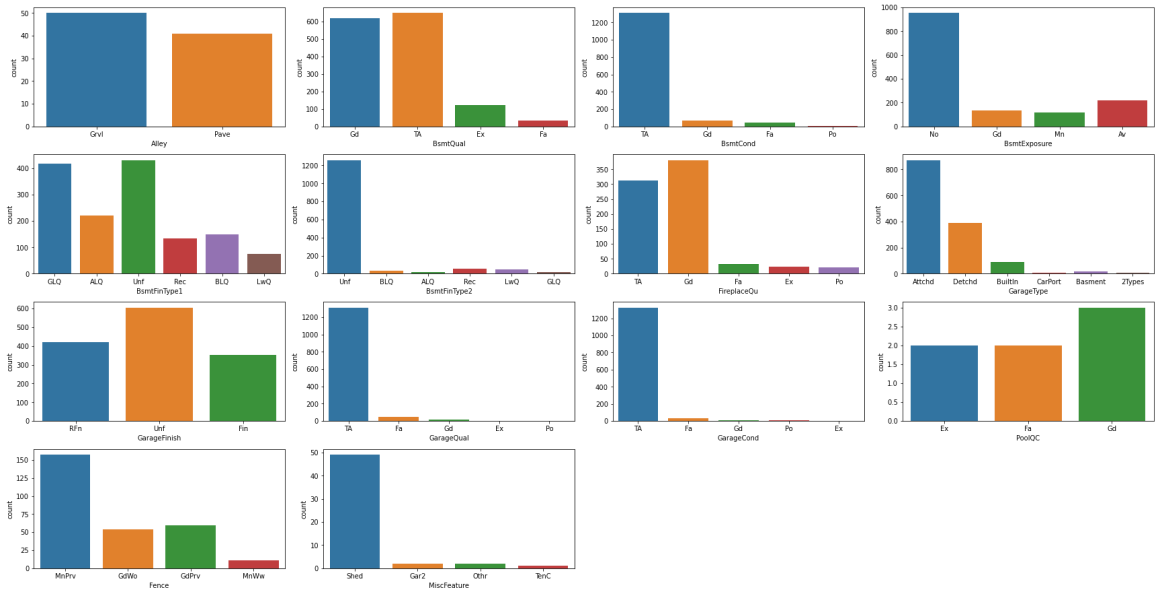
```
MSZoning           0
Street             0
Alley           1369
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType         8
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Heating            0
HeatingQC          0
CentralAir         0
Electrical         1
KitchenQual        0
Functional         0
FireplaceQu      690
GarageType        81
GarageFinish      81
GarageQual        81
GarageCond        81
PavedDrive         0
PoolQC          1453
Fence           1179
MiscFeature     1406
SaleType           0
SaleCondition      0
dtype: int64
```

In [39]:

```
cat_col_nan = ['Alley', 'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinT
               'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', '
              'Fence', 'MiscFeature']
```

In [40]:

```
#Verifying the categorical features which has null values

count_plot(train, cat_col_nan)
```



Observations

- Alley: NA here means No alley access. Replace 'NA' with 'No Alley' then OneHot can be done for this column
- FirePlaceQu: NA means 'No fireplace'. Before deciding the encoding, maybe we need to analyze the dependency on target.
- PoolQC: Same as above.
- Fence: NA means no fence. OrdinalEncoding can be done.
- MiscFeature: NA means no features. OneHotEncoding is apt.
- GarageType NA means no garage and as a result GarageFinish, GarageQual, GarageCond has NA values. Same like PoolQC.
- GarageQual and GarageCond are almost same. Deleting one of them.
- BsmtExposure, BsmtFinType1, BsmtFinType2: Ordinal encoding can be done but lets check the dependeny first.
- BsmtQual - Ordinal encoding
- BsmtCond - Ordinal Encoding

In [41]:

```
remove_cols.append('GarageCond')
remove_cols
```

Out[41]:

```
['GarageCars',
 'TotRmsAbvGrd',
 'BedroomAbvGr',
 '1stFlrSF',
 'MiscVal',
 'PoolArea',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'OpenPorchSF',
 'WoodDeckSF',
 'LowQualFinSF',
 '2ndFlrSF',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'MasVnrArea',
 'GarageCond']
```

In [42]:

```
# trans_nan_cols = {'Alley': 'No Alley', 'FireplaceQu': 'No Fireplace', 'PoolQC': 'No Poo
#                   'MiscFeature': 'No MiscFeature', 'GarageType': 'No Garage', 'GarageFin
#                    'GarageQual': 'No Garage'}

cat = train.copy()

for col in cat.columns:
    cat.loc[(cat[col].isnull()) | (cat[col] == 'None') | (cat[col] == 'No'), col] = 'No'+

cat
```

Out[42]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCon
0	1	60	RL	65.0	8450	Pave	No Alley	Reg	
1	2	20	RL	80.0	9600	Pave	No Alley	Reg	
2	3	60	RL	68.0	11250	Pave	No Alley	IR1	
3	4	70	RL	60.0	9550	Pave	No Alley	IR1	
4	5	60	RL	84.0	14260	Pave	No Alley	IR1	
...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	No Alley	Reg	
1456	1457	20	RL	85.0	13175	Pave	No Alley	Reg	
1457	1458	70	RL	66.0	9042	Pave	No Alley	Reg	
1458	1459	20	RL	68.0	9717	Pave	No Alley	Reg	
1459	1460	20	RL	75.0	9937	Pave	No Alley	Reg	

1460 rows × 81 columns

In [43]:

```
cat[cat_cols].isnull().sum()
```

Out[43]:

```
MSZoning         0
Street           0
Alley            0
LotShape         0
LandContour      0
Utilities        0
LotConfig        0
LandSlope        0
Neighborhood     0
Condition1       0
Condition2       0
BldgType         0
HouseStyle       0
RoofStyle        0
RoofMatl         0
Exterior1st      0
Exterior2nd      0
MasVnrType       0
ExterQual        0
ExterCond        0
Foundation       0
BsmtQual         0
BsmtCond         0
BsmtExposure     0
BsmtFinType1     0
BsmtFinType2     0
Heating          0
HeatingQC        0
CentralAir       0
Electrical       0
KitchenQual      0
Functional       0
FireplaceQu      0
GarageType       0
GarageFinish     0
GarageQual       0
GarageCond       0
PavedDrive       0
PoolQC           0
Fence            0
MiscFeature      0
SaleType         0
SaleCondition    0
dtype: int64
```

In [44]:

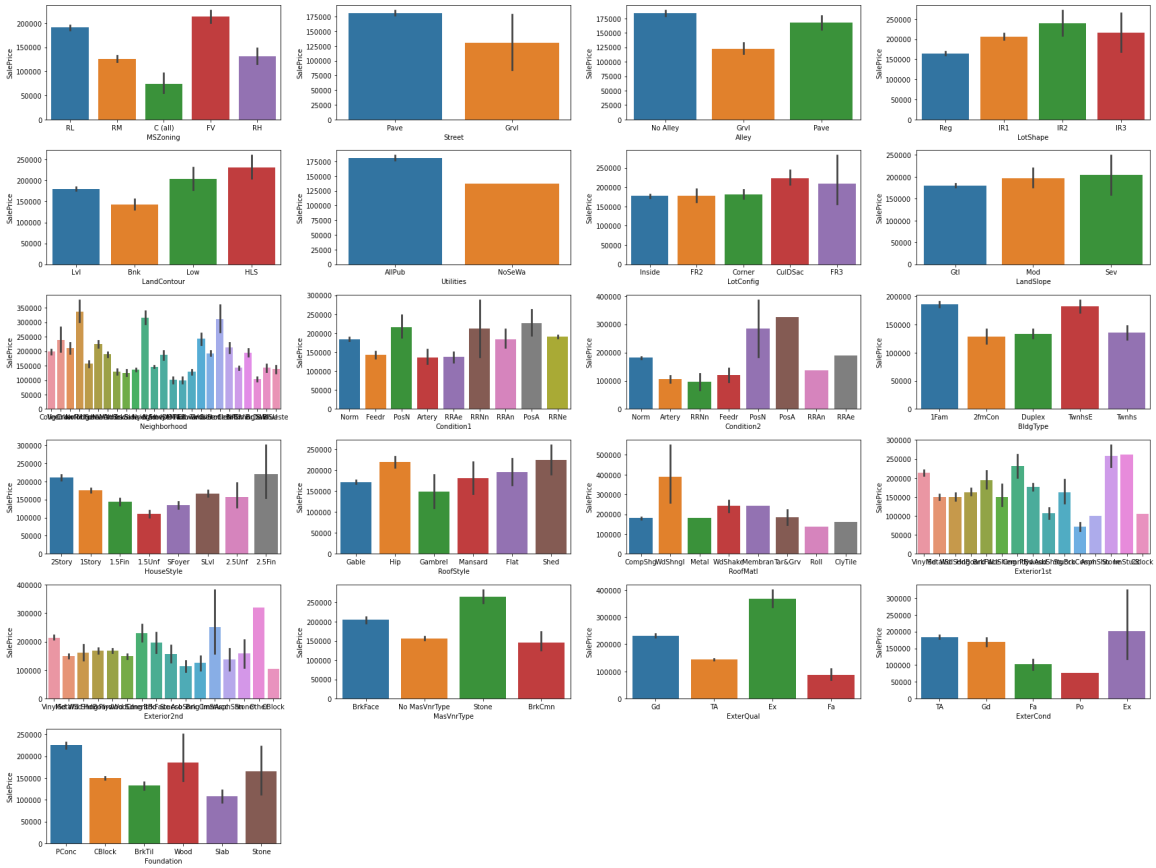
```
#Checking every categorical columns and their dependency on target
```

In [45]:

```
count_plot(cat, cat_cols[:21])
```

In [46]:

```
count_plot(cat, cat_cols[:21], kind='barplot')
```



Observation

- MSZoning: Lets make the categories as 'RL & 'Other' then one-hot encoding
- Street: Almost all type of roads are Paved. Maybe can be deleted.
- Alley catagories can be transformed as 'No Alley' and 'With Alley' then one-hot.
- LotShape: Make categories as 'Regular' and 'Irregular' then one hot.
- LandContour: 'Lvl' & 'Other' then one hot
- Utilities: Remove the column.
- LotConfig: 'Inside' & 'Other'
- LandSlope: 'Gtl' & 'Other'
- Neighbourhood: Lets analyze more
- Condition1: 'Norm', 'Feedr', 'Artery', 'Other'.
- Condition2: Lets check the dependency with condition1 and accordingly will decide.
- BldgType: One hot
- HouseStyle: Make the categories as 1Story, 2Story, 1.5Story and other
- RoofStyle: Gable, Hip, Other
- RoofMatl: Lets delete this column. Almost 95% is of a single category.
- MasVnrType: On hot encoding.
- ExternalQual: Lets do Ordinal encoding.
- ExternalCond: Lets do Ordinal encoding.
- Foundation: PConc, CBlock, BrkTil and Other.

In [47]:

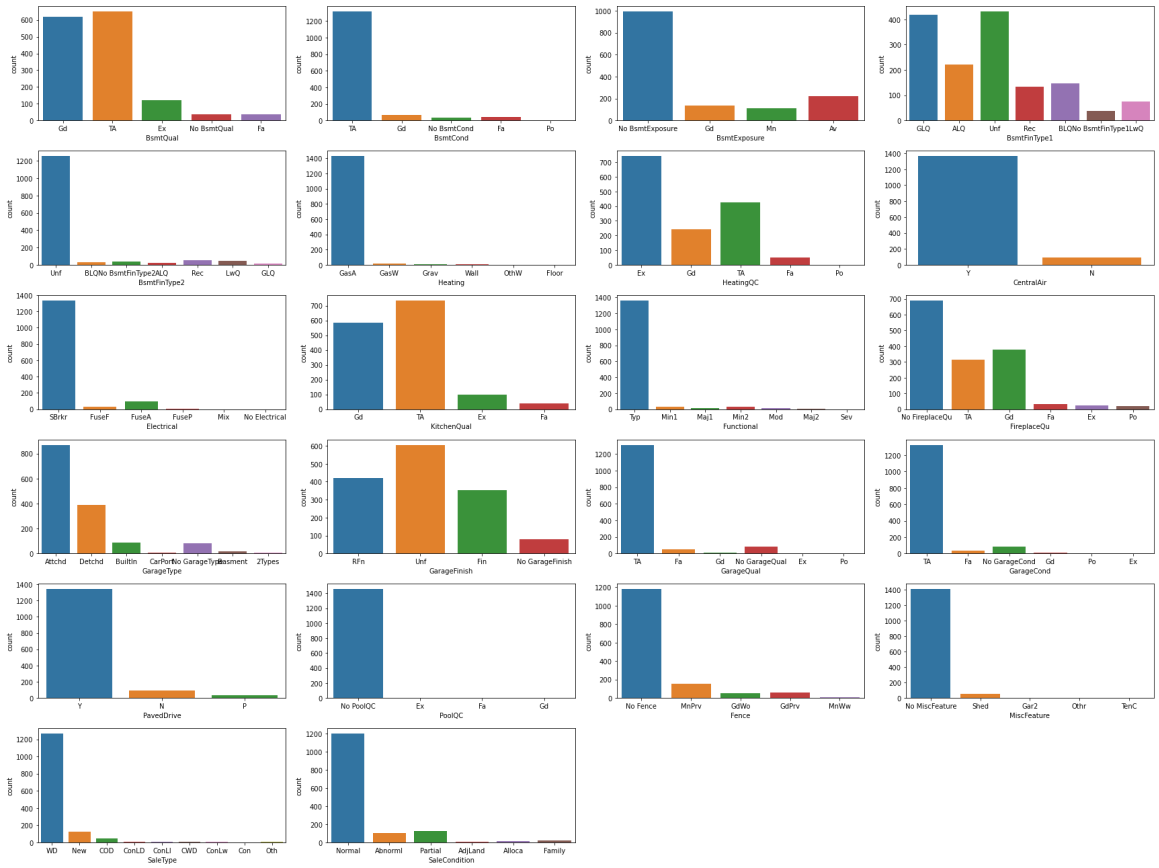
```
cat_cols
```

Out[47]:

```
['MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']
```

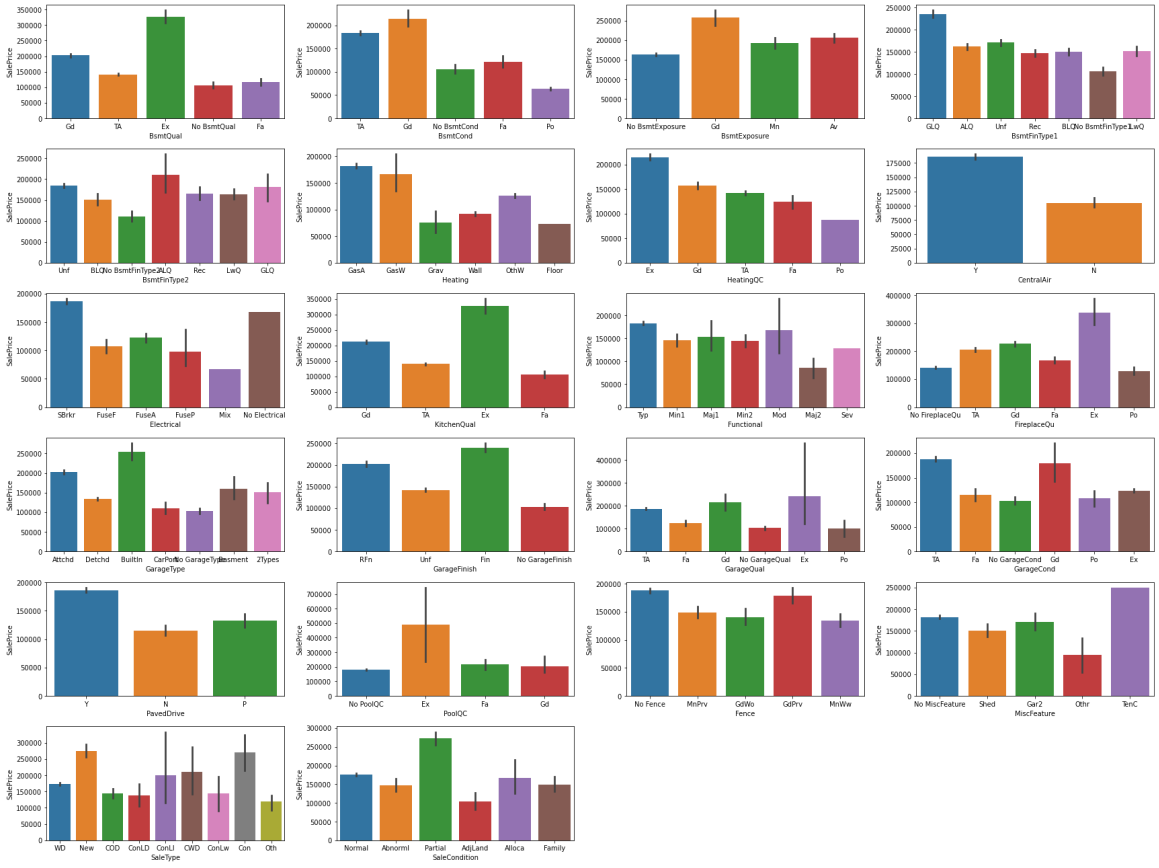
In [48]:

```
count_plot(cat, cat_cols[21:])
```



In [49]:

```
count_plot(cat, cat_cols[21:], kind='barplot')
```



Observation

- BsmtQual, BsmtCond: ordinal encoding
- BsmtExposure: Ordinal encoding
- BsmtFinType1: Ordinal Enc
- BsmtFinType2: Ordinal Enc
- Heating: Categories are 'GasA' and 'Other'; one hot
- HeatingQC: Ordinal
- CentralAir: one hot
- Electrical: 'SBrkr' and 'Other'; one hot
- KitchenQual: Ordinal
- Functional: 'Typ' & 'Other'; one hot
- FireplaceQu: ordinal
- GarageType: 'Attachd', 'Detachd' and 'Other' ; one hot
- GarageFinish: one hot
- GarageQual: 'TA', 'Other' ; one hot
- GarageCond can be deleted; seems same as the GarageQual.
- PavedDrive: 'Y' and 'N' ; ordinal
- PoolQC: 99% rows has only one valie, can be deleted.
- Fence: 'No Fence' , 'Hasfence'; ordinal
- MiscFeature: 'Yes' and 'No'; ordinal
- SaleType: 'WD', 'New', 'COD', 'Other'; one hot
- SaleCondition: 'Normal', 'Not Normal' ; ordinal

In [50]:

```
#Checking Neighbourhood column

df_nei = train.groupby(['Neighborhood']).aggregate({'SalePrice': ['mean', 'count']}).rese

df_nei.columns = ['Neighbor', 'Sale_Mean', 'Homes']

df_nei = df_nei.sort_values(by='Sale_Mean')
df_nei
```

Out[50]:

	Neighbor	Sale_Mean	Homes
10	MeadowV	98576.470588	17
9	IDOTRR	100123.783784	37
2	BrDale	104493.750000	16
3	BrkSide	124834.051724	58
7	Edwards	128219.700000	100
17	OldTown	128225.300885	113
19	Sawyer	136793.135135	74
1	Blueste	137500.000000	2
18	SWISU	142591.360000	25
13	NPkVill	142694.444444	9
12	NAmes	145847.080000	225
11	Mitchel	156270.122449	49
20	SawyerW	186555.796610	59
14	NWAmes	189050.068493	73
8	Gilbert	192854.506329	79
0	Blmngtn	194870.882353	17
5	CollgCr	197965.773333	150
6	Crawfor	210624.725490	51
4	ClearCr	212565.428571	28
21	Somerst	225379.837209	86
24	Veenker	238772.727273	11
23	Timber	242247.447368	38
22	StoneBr	310499.000000	25
16	NridgHt	316270.623377	77
15	NoRidge	335295.317073	41

In [51]:

```
#Can create probabilities of each neighbourhod

df_nei['NeighProba'] = df_nei['Homes']/train.shape[0]
df_nei
```

Out[51]:

	Neighbor	Sale_Mean	Homes	NeighProba
10	MeadowV	98576.470588	17	0.011644
9	IDOTRR	100123.783784	37	0.025342
2	BrDale	104493.750000	16	0.010959
3	BrkSide	124834.051724	58	0.039726
7	Edwards	128219.700000	100	0.068493
17	OldTown	128225.300885	113	0.077397
19	Sawyer	136793.135135	74	0.050685
1	Blueste	137500.000000	2	0.001370
18	SWISU	142591.360000	25	0.017123
13	NPkVill	142694.444444	9	0.006164
12	NAmes	145847.080000	225	0.154110
11	Mitchel	156270.122449	49	0.033562
20	SawyerW	186555.796610	59	0.040411
14	NWAmes	189050.068493	73	0.050000
8	Gilbert	192854.506329	79	0.054110
0	Blmngtn	194870.882353	17	0.011644
5	CollgCr	197965.773333	150	0.102740
6	Crawfor	210624.725490	51	0.034932
4	ClearCr	212565.428571	28	0.019178
21	Somerst	225379.837209	86	0.058904
24	Veenker	238772.727273	11	0.007534
23	Timber	242247.447368	38	0.026027
22	StoneBr	310499.000000	25	0.017123
16	NridgHt	316270.623377	77	0.052740
15	NoRidge	335295.317073	41	0.028082

In [52]:

```
df_cond = train.groupby(['Condition1', 'Condition2']).aggregate({'SalePrice': ['mean', 'c

df_cond.columns = ['Condition1', 'Condition2', 'Sale_Mean', 'Homes']

df_cond = df_cond.sort_values(by='Sale_Mean')
df_cond
```

Out[52]:

	Condition1	Condition2	Sale_Mean	Homes
3	Feedr	Feedr	85000.000000	1
7	Feedr	RRNn	96750.000000	2
0	Artery	Artery	106500.000000	2
16	RRNn	Feedr	128000.000000	1
13	RRAn	Feedr	128500.000000	4
1	Artery	Norm	132142.222222	45
6	Feedr	RRAn	136905.000000	1
12	RRAe	Norm	138400.000000	11
4	Feedr	Norm	143883.013158	76
8	Norm	Norm	184495.492063	1260
5	Feedr	RRAe	190000.000000	1
15	RRNe	Norm	190750.000000	2
14	RRAn	Norm	194559.636364	22
10	PosN	Norm	206985.294118	17
9	PosA	Norm	225875.000000	8
17	RRNn	Norm	233500.000000	4
11	PosN	PosN	284875.000000	2
2	Artery	PosA	325000.000000	1

In [53]:

```
#Can create probabilities of each neighbourhod

df_cond['CondProba'] = df_cond['Homes']/train.shape[0]
df_cond
```

Out[53]:

	Condition1	Condition2	Sale_Mean	Homes	CondProba
3	Feedr	Feedr	85000.000000	1	0.000685
7	Feedr	RRNn	96750.000000	2	0.001370
0	Artery	Artery	106500.000000	2	0.001370
16	RRNn	Feedr	128000.000000	1	0.000685
13	RRAn	Feedr	128500.000000	4	0.002740
1	Artery	Norm	132142.222222	45	0.030822
6	Feedr	RRAn	136905.000000	1	0.000685
12	RRAe	Norm	138400.000000	11	0.007534
4	Feedr	Norm	143883.013158	76	0.052055
8	Norm	Norm	184495.492063	1260	0.863014
5	Feedr	RRAe	190000.000000	1	0.000685
15	RRNe	Norm	190750.000000	2	0.001370
14	RRAn	Norm	194559.636364	22	0.015068
10	PosN	Norm	206985.294118	17	0.011644
9	PosA	Norm	225875.000000	8	0.005479
17	RRNn	Norm	233500.000000	4	0.002740
11	PosN	PosN	284875.000000	2	0.001370
2	Artery	PosA	325000.000000	1	0.000685

In [ ]:

```

```

Analyzing date columns

In [54]:

```
date_cols
```

Out[54]:

```
['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'MoSold', 'YrSold']
```

In [55]:

```
#MoSold can be deleted citing very less dependency on SalePrice
#Instead of 'YearRemodAdd' we can add a new column as Remodified with (1/0)
#Add a column 'Building_Age' as the difference between YrSold and YearBuilt
```

In [56]:

```
# trans_nan_cols = {'Alley': 'No Alley', 'FireplaceQu': 'No Fireplace', 'PoolQC': 'No Poo
#                   'MiscFeature': 'No MiscFeature', 'GarageType': 'No Garage', 'GarageFin
#                    'GarageQual': 'No Garage'}

# new_col_rpl = {'MiscVal': 'HasSpecialFeature', 'PoolArea': 'HasPoolArea', 'EnclosedPorc
#                '3SsnPorch': 'Has3SsnPorch', 'ScreenPorch': 'HasScreenPorch', 'OpenPorch'
#                'LowQualSF': 'HasLowQual', '2ndFlrSF': 'Has2ndFlr', 'BsmtFinSF1': 'HasTyp
#                 'BsmtFinSF2': 'HasType2Finishing', 'MasVnrArea': 'HasMasVnrArea', 'YearR
```

In [57]:

```
for col in date_cols:
    remove_cols.append(col)

remove_cols
```

Out[57]:

```
['GarageCars',
 'TotRmsAbvGrd',
 'BedroomAbvGr',
 '1stFlrSF',
 'MiscVal',
 'PoolArea',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'OpenPorchSF',
 'WoodDeckSF',
 'LowQualFinSF',
 '2ndFlrSF',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'MasVnrArea',
 'GarageCond',
 'YearBuilt',
 'YearRemodAdd',
 'GarageYrBlt',
 'MoSold',
 'YrSold']
```

In [58]:

```
cat_cols = ['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'BldgType',
            'RoofStyle', 'MasVnrType', 'Foundation', 'Heating', 'CentralAir', 'Electrical'
            'GarageFinish', 'SaleType', 'SaleCondition', 'FireplaceQu', 'ExterQual', 'Exte
             'KitchenQual', 'GarageQual', 'PavedDrive', 'Alley', 'Exterior1st', 'Exterior2
cat_cols
```

Out[58]:

```
['MSZoning',
 'LotShape',
 'LandContour',
 'LotConfig',
 'LandSlope',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'MasVnrType',
 'Foundation',
 'Heating',
 'CentralAir',
 'Electrical',
 'Functional',
 'GarageType',
 'GarageFinish',
 'SaleType',
 'SaleCondition',
 'FireplaceQu',
 'ExterQual',
 'ExterCond',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'HeatingQC',
 'KitchenQual',
 'GarageQual',
 'PavedDrive',
 'Alley',
 'Exterior1st',
 'Exterior2nd']
```

To-do List

1. Remove outliers from the numeric features.
2. Treat missing values of the numerical columns.
3. Treat missing values of the countable and date columns.
4. Add new columns as per new_col_rpl
5. Transform NaN values as per trans_nan_cols
6. Perform step for date columns.
7. Add Proba columns
8. Remove unwanted columns.
9. Treat Missing values of the categorical columns
10. Perform Ordinal/Onehot encoding for categorical

3. Data Preprocessing

In [59]:

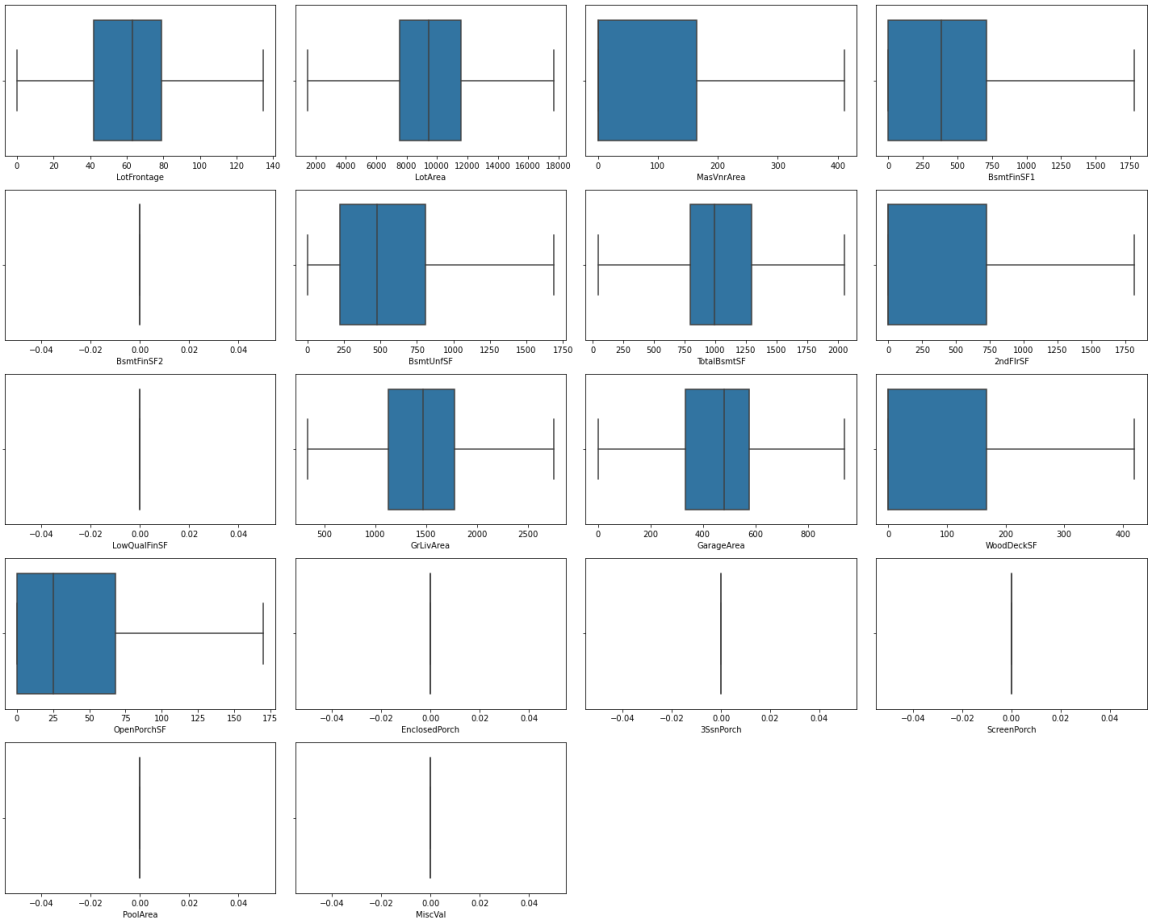
```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

In [60]:

```
def treat_outliers(X, columns):

    new_X = X.copy()
    new_X[columns] = X[columns].fillna(value=0)

    for column in columns:

        Q1 = np.percentile(new_X[column], 25.)
        Q2 = np.percentile(new_X[column], 50.)
        Q3 = np.percentile(new_X[column], 75.)

        IQR = (Q3 - Q1) * 1.5
        upper, lower = Q3 + IQR, Q1 - IQR

        new_X.loc[(new_X[column] > upper), column] = upper
        new_X.loc[(new_X[column] < lower), column] = lower

    return new_X
```

In [61]:

```
train_out = treat_outliers(train, num_cols)

dist_plot(train_out, num_cols, kind='boxplot')
```



In [62]:

```
y_train = train_out['SalePrice']
X_train = train_out.drop(['SalePrice'], axis=1)
```

In [63]:

```
class CategoryTransformer(BaseEstimator, TransformerMixin):

    def __init__(self):
        self.trans_nan_cols = {'Alley': 'No Alley', 'FireplaceQu': 'No Fireplace',
                 'GarageType': 'No Garage', 'GarageFinish': 'No Garage',
                  'GarageQual': 'No Garage', 'BsmtQual': 'No Basement', 'BsmtCond': 'No B
                        'BsmtFinType1': 'No Basement', 'BsmtFinType2': 'No Basement', 'F

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        new_X = X.copy()

        for col in self.trans_nan_cols.keys():
            new_X.loc[(new_X[col].isnull()) | (new_X[col] == 'None') | (new_X[col] == 'No

        hot_enc = OneHotEncoder(drop='first', dtype='int', max_categories=4)
        enc_X = hot_enc.fit_transform(new_X).toarray()
        return enc_X
```

In [64]:

```
cat_trans = CategoryTransformer()
cat_dt = cat_trans.transform(X_train[cat_cols])
cat_dt
```

Out[64]:

```
array([[1, 0, 0, ..., 0, 1, 0],
       [1, 0, 0, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 1, 0],
       ...,
       [1, 0, 0, ..., 0, 0, 1],
       [1, 0, 0, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 0, 0]])
```

In [65]:

```
class DataFilter(BaseEstimator, TransformerMixin):
    def __init__(self):
        self.cols_to_remove = ['Id','GarageCars', 'TotRmsAbvGrd', 'BedroomAbvGr', '1stFlr
                               'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'OpenPorchSF'
                               '2ndFlrSF', 'BsmtFinSF1', 'BsmtFinSF2', 'MasVnrArea', 'Gar
                               'GarageYrBlt', 'MoSold', 'YrSold', 'Street', 'Utilities',
                               'Neighborhood', 'Condition1', 'Condition2', 'MiscFeature',

        self.new_col_rpl = {'MiscVal': 'HasSpecialFeature', 'PoolArea': 'HasPool', 'Enclo
              '3SsnPorch': 'Has3SsnPorch', 'ScreenPorch': 'HasScreenPorch', 'OpenPorchSF'
              'LowQualFinSF': 'HasLowQualFin', '2ndFlrSF': 'Has2ndFlr', 'BsmtFinSF1': 'Ha
               'BsmtFinSF2': 'HasType2Finishing', 'MasVnrArea': 'HasMasVnrArea', 'YearRem

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):

        new_X = X.copy()

        for old, new in self.new_col_rpl.items():
            new_X[new] = new_X[old].apply(lambda x: 1 if x !=0 else 0)

        new_X['BuildingAge'] = new_X['YrSold'] - new_X['YearBuilt']

        probaSet = ['Neighborhood', 'Condition1', 'Condition2']

        for col in probaSet:
            probas = new_X[col].value_counts(normalize=True)
            prefix = col+'Proba'
            new_X[prefix] = new_X[col].apply(lambda x : probas[x])

        trans_X = new_X.drop(self.cols_to_remove, axis=1)
        return trans_X.to_numpy()
```

In [66]:

```
X_train
```

Out[66]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCon
0	1	60	RL	65.0	8450.0	Pave	NaN	Reg	
1	2	20	RL	80.0	9600.0	Pave	NaN	Reg	
2	3	60	RL	68.0	11250.0	Pave	NaN	IR1	
3	4	70	RL	60.0	9550.0	Pave	NaN	IR1	
4	5	60	RL	84.0	14260.0	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917.0	Pave	NaN	Reg	
1456	1457	20	RL	85.0	13175.0	Pave	NaN	Reg	
1457	1458	70	RL	66.0	9042.0	Pave	NaN	Reg	
1458	1459	20	RL	68.0	9717.0	Pave	NaN	Reg	
1459	1460	20	RL	75.0	9937.0	Pave	NaN	Reg	

1460 rows × 80 columns

In [68]:

```
all_cols = list(X_train.columns)
```

In [69]:

```
#Certain numeric columns needs to be replaced and those has to be removed from num_cols l

filter_cols = []

new_col_rpl = {'MiscVal': 'HasSpecialFeature', 'PoolArea': 'HasPool', 'EnclosedPorch': 'H
               '3SsnPorch': 'Has3SsnPorch', 'ScreenPorch': 'HasScreenPorch', 'OpenPorchSF'
               'LowQualFinSF': 'HasLowQualFin', '2ndFlrSF': 'Has2ndFlr', 'BsmtFinSF1': 'Ha
                'BsmtFinSF2': 'HasType2Finishing', 'MasVnrArea': 'HasMasVnrArea', 'YearRem

for col in new_col_rpl.keys():
    if col in num_cols:
        num_cols.remove(col)

for col in all_cols:
    if (col not in num_cols) and (col not in num_countable) and (col not in cat_cols):
        filter_cols.append(col)

filter_cols
```

Out[69]:

```
['Id',
 'Street',
 'Utilities',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'YearBuilt',
 'YearRemodAdd',
 'RoofMatl',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 '1stFlrSF',
 '2ndFlrSF',
 'LowQualFinSF',
 'BedroomAbvGr',
 'TotRmsAbvGrd',
 'GarageYrBlt',
 'GarageCars',
 'GarageCond',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 '3SsnPorch',
 'ScreenPorch',
 'PoolArea',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'MiscVal',
 'MoSold',
 'YrSold']
```

In [70]:

```
dt = DataFilter()

trans_data = dt.transform(X_train[filter_cols])
trans_data
```

Out[70]:

```
array([[0.        , 0.        , 0.        , ..., 0.10273973, 0.8630137 ,
        0.98972603],
       [0.        , 0.        , 0.        , ..., 0.00753425, 0.05547945,
        0.98972603],
       [0.        , 0.        , 0.        , ..., 0.10273973, 0.8630137 ,
        0.98972603],
       ...,
       [0.        , 0.        , 0.        , ..., 0.03493151, 0.8630137 ,
        0.98972603],
       [0.        , 0.        , 0.        , ..., 0.15410959, 0.8630137 ,
        0.98972603],
       [0.        , 0.        , 0.        , ..., 0.06849315, 0.8630137 ,
        0.98972603]])
```

In [71]:

```
len(trans_data[0])
```

Out[71]:

```
17
```

In [72]:

```
#Imputing Values

imp1 = SimpleImputer(strategy='median')
imp_num = imp1.fit_transform(X_train[num_cols])
imp_num
```

Out[72]:

```
array([[   65.,  8450.,   150.,   856.,  1710.,   548.],
       [   80.,  9600.,   284.,  1262.,  1262.,   460.],
       [   68., 11250.,   434.,   920.,  1786.,   608.],
       ...,
       [   66.,  9042.,   877.,  1152.,  2340.,   252.],
       [   68.,  9717.,     0.,  1078.,  1078.,   240.],
       [   75.,  9937.,   136.,  1256.,  1256.,   276.]])
```

In [73]:

```
imp2 = SimpleImputer(strategy='most_frequent')
imp_countable = imp2.fit_transform(X_train[num_countable])
imp_countable
```

Out[73]:

```
array([[60,  7,  5, ...,  0,  2,  1],
       [20,  6,  8, ...,  1,  2,  0],
       [60,  7,  5, ...,  0,  2,  1],
       ...,
       [70,  7,  9, ...,  0,  2,  0],
       [20,  5,  6, ...,  0,  1,  0],
       [20,  5,  6, ...,  0,  1,  1]], dtype=int64)
```

In [ ]:

```

```

In [74]:

```
imp3 = SimpleImputer(strategy='most_frequent')
imp_category = imp3.fit_transform(train_out[cat_cols])
imp_category
```

Out[74]:

```
array([['RL', 'Reg', 'Lvl', ..., 'Grvl', 'VinylSd', 'VinylSd'],
       ['RL', 'Reg', 'Lvl', ..., 'Grvl', 'MetalSd', 'MetalSd'],
       ['RL', 'IR1', 'Lvl', ..., 'Grvl', 'VinylSd', 'VinylSd'],
       ...,
       ['RL', 'Reg', 'Lvl', ..., 'Grvl', 'CemntBd', 'CmentBd'],
       ['RL', 'Reg', 'Lvl', ..., 'Grvl', 'MetalSd', 'MetalSd'],
       ['RL', 'Reg', 'Lvl', ..., 'Grvl', 'HdBoard', 'HdBoard']],
      dtype=object)
```

In [75]:

```
#Encoding the applicable categorical columns
```

In [76]:

```
one_hot = OneHotEncoder(drop='first', dtype='int')
hot_enc = one_hot.fit_transform(train_out[cat_cols])
hot_enc.toarray()
```

Out[76]:

```
array([[0, 0, 1, ..., 1, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [0, 0, 1, ..., 1, 0, 0],
       ...,
       [0, 0, 1, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0]])
```

In [77]:

```
#ordi =  OrdinalEncoder()
```

In [78]:

```
#Creating Data Preparation Pipeline

num_pipeline = Pipeline([('imp1', SimpleImputer(strategy='median'))])

count_pipeline = Pipeline([('imp2', SimpleImputer(strategy='most_frequent'))])

data_filter_pipeline = Pipeline([('dt', DataFilter())])

cat_pipeline = Pipeline([('cat_dt', CategoryTransformer())])

# hot_enc_pl = Pipeline([('one_hot', OneHotEncoder(drop='first', dtype='int'))])

# ord_enc_pl = Pipeline([('ordi', OrdinalEncoder(dtype='int'))])
```

In [79]:

```
# data_pipeline = ColumnTransformer([('num', num_pipeline, num_cols),
#                                    ('countable', count_pipeline, num_countable),
#                                    ('filter', data_filter_pipeline, all_cols),
#                                    ('cat', cat_pipeline, cat_cols),
#                                    ('hot_enc', hot_enc_pl, hot_cols),
#                                    ('ord_enc', ord_enc_pl, ord_cols)])

data_pipeline = ColumnTransformer([('num', num_pipeline, num_cols),
                                   ('countable', count_pipeline, num_countable),
                                   ('filter', data_filter_pipeline, filter_cols),
                                   ('cat', cat_pipeline, cat_cols)])


data_pipeline
```

Out[79]:

```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imp1',
                                                  SimpleImputer(strategy
='median'))]),
                                 ['LotFrontage', 'LotArea', 'BsmtUnfSF',
                                  'TotalBsmtSF', 'GrLivArea', 'GarageAre
a']),
                                ('countable',
                                 Pipeline(steps=[('imp2',
                                                  SimpleImputer(strategy
='most_frequent'))]),
                                 ['MSSubClass', 'OverallQual', 'OverallCon
d',
                                  'KitchenAbvGr', 'Fireplaces', 'BsmtFullB
ath',
                                  'BsmtHal...
                                  'LotConfig', 'LandSlope', 'BldgType',
                                  'HouseStyle', 'RoofStyle', 'MasVnrType',
                                  'Foundation', 'Heating', 'CentralAir',
                                  'Electrical', 'Functional', 'GarageTyp
e',
                                  'GarageFinish', 'SaleType', 'SaleConditi
on',
                                  'FireplaceQu', 'ExterQual', 'ExterCond',
                                  'BsmtQual', 'BsmtCond', 'BsmtExposure',
                                  'BsmtFinType1', 'BsmtFinType2', 'Heating
QC',
                                  'KitchenQual', 'GarageQual', 'PavedDriv
e', ...])])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [80]:

```
data_prep = data_pipeline.fit_transform(X_train)

data_prep
```

Out[80]:

```
array([[6.500e+01, 8.450e+03, 1.500e+02, ..., 0.000e+00, 1.000e+00,
        0.000e+00],
       [8.000e+01, 9.600e+03, 2.840e+02, ..., 1.000e+00, 0.000e+00,
        0.000e+00],
       [6.800e+01, 1.125e+04, 4.340e+02, ..., 0.000e+00, 1.000e+00,
        0.000e+00],
       ...,
       [6.600e+01, 9.042e+03, 8.770e+02, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [6.800e+01, 9.717e+03, 0.000e+00, ..., 1.000e+00, 0.000e+00,
        0.000e+00],
       [7.500e+01, 9.937e+03, 1.360e+02, ..., 0.000e+00, 0.000e+00,
        0.000e+00]])
```

In [81]:

```
data_prep[2]
```

Out[81]:

```
array([6.80000000e+01, 1.12500000e+04, 4.34000000e+02, 9.20000000e+02,
       1.78600000e+03, 6.08000000e+02, 6.00000000e+01, 7.00000000e+00,
       5.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
       0.00000000e+00, 2.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
       1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 1.00000000e+00,
       7.00000000e+00, 1.02739726e-01, 8.63013699e-01, 9.89726027e-01,
       1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 1.00000000e+00,
       0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
       1.00000000e+00, 0.00000000e+00])
```

In [82]:

```
len(data_prep[0])
```

Out[82]:

```
126
```

3. Model Evaluation

In [83]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

In [84]:

```
from sklearn.linear_model import ElasticNet
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

In [85]:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

In [86]:

```
def model_set():
    model_set = []

    model_set.append(('EN', ElasticNet()))
    model_set.append(('SV', SVR()))
    model_set.append(('RF', RandomForestRegressor()))
    model_set.append(('GB', GradientBoostingRegressor()))

    return model_set

def score_model(X, y, models, scale=None):

    estimators = []
    rmse = []
    r2 = []

    if scale == 'MinMax':
        scaling = MinMaxScaler()
        X = scaling.fit_transform(X)
    if scale == 'Standard':
        scaling = StandardScaler()
        X = scaling.fit_transform(X)

    for name, model in models:
        k = StratifiedKFold(n_splits=15, shuffle=True, random_state=seed)
        cv_rmse = cross_val_score(model, X, y, cv = k, scoring='neg_root_mean_squared_err
        cv_r2 = cross_val_score(model, X, y, cv = k, scoring='r2')

        cv_rmse = -(cv_rmse)

        estimators.append(model)
        rmse.append(cv_rmse.mean())
        r2.append(cv_r2.mean())

    score_matrix = list(zip(estimators, rmse, r2))
    df = pd.DataFrame(score_matrix, columns = ['Estimator', 'Mean RMSE', 'Mean R Squarred
    return df
```

Default performance of models

In [87]:

```
models = model_set()

score_model(data_prep, y_train, models)
```

Out[87]:

	Estimator	Mean RMSE	Mean R Squarred
0	ElasticNet()	32631.795283	0.821116
1	SVR()	80409.878127	-0.057101
2	RandomForestRegressor()	30407.885580	0.821581
3	GradientBoostingRegressor()	27829.950131	0.825367

Model performance with MinMax scaling

In [88]:

```
score_model(data_prep, y_train, models, scale='MinMax')
```

Out[88]:

	Estimator	Mean RMSE	Mean R Squarred
0	ElasticNet()	47117.542328	0.638871
1	SVR()	80384.066649	-0.056413
2	RandomForestRegressor()	29791.362381	0.820077
3	GradientBoostingRegressor()	27920.542734	0.828306

Model performance with Standard scaling

In [89]:

```
score_model(data_prep, y_train, models, scale='Standard')
```

Out[89]:

	Estimator	Mean RMSE	Mean R Squarred
0	ElasticNet()	32048.056066	0.824311
1	SVR()	80401.664265	-0.056879
2	RandomForestRegressor()	30008.906293	0.818411
3	GradientBoostingRegressor()	28015.771014	0.823266

With Standard scaler the R2 score is better

Also StandardScaler seems to give better R2

- Promising Algorithms
  - Random Forest
  - Gradient Boosting
  - ElasticNet

4. Model Selection

In [90]:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_predict
```

In [91]:

```
scale = StandardScaler()

X_scaled = scale.fit_transform(data_prep)
X_scaled
```

Out[91]:

```
array([[ 0.23726036, -0.33324416, -0.95879019, ..., -0.41442683,
         1.37725284, -0.76051192],
       [ 0.69281023, -0.01318852, -0.64829044, ...,  2.41297118,
        -0.72608309, -0.76051192],
       [ 0.32837033,  0.44602174, -0.30071609, ..., -0.41442683,
         1.37725284, -0.76051192],
       ...,
       [ 0.26763035, -0.16848509,  0.72578682, ..., -0.41442683,
        -0.72608309,  1.31490378],
       [ 0.32837033,  0.01937366, -1.30636454, ...,  2.41297118,
        -0.72608309, -0.76051192],
       [ 0.54096027,  0.0806017 , -0.99123046, ..., -0.41442683,
        -0.72608309, -0.76051192]])
```

In [92]:

```
promising_model_list = []

gb = GradientBoostingRegressor(random_state=seed)
promising_model_list.append(gb)

rf = RandomForestRegressor(random_state=seed)
promising_model_list.append(rf)

en = ElasticNet(random_state=seed)
promising_model_list.append(en)
```

In [93]:

```
param_grid = [
    {
        'loss' : ['squared_error', 'huber', 'quantile'],
        'n_estimators': [100, 150, 200],
        'criterion': ['friedman_mse', 'squared_error'],
        'max_features': ['sqrt', 'log2'],
        'alpha': [None, 0.8, 0.1, 0.12]
    },
    {
        'n_estimators': [90, 100, 150, 200],
        'criterion': ['friedman_mse', 'squared_error', 'poisson'],
        'max_features': ['sqrt', 'log2'],
        'bootstrap': [True, False]
    },
    {
        'alpha': [0.9, 1.0, 1.3, 1.7],
        'l1_ratio': [0.3, 0.5, 0.8],
        'selection': ['cyclic', 'random']
    }
]


model_param_map = dict(zip(promising_model_list, param_grid))
model_param_map
```

Out[93]:

```
{GradientBoostingRegressor(random_state=42): {'loss': ['squared_error',
   'huber',
   'quantile'],
  'n_estimators': [100, 150, 200],
  'criterion': ['friedman_mse', 'squared_error'],
  'max_features': ['sqrt', 'log2'],
  'alpha': [None, 0.8, 0.1, 0.12]},
 RandomForestRegressor(random_state=42): {'n_estimators': [90, 100, 150, 2
00],
  'criterion': ['friedman_mse', 'squared_error', 'poisson'],
  'max_features': ['sqrt', 'log2'],
  'bootstrap': [True, False]},
 ElasticNet(random_state=42): {'alpha': [0.9, 1.0, 1.3, 1.7],
  'l1_ratio': [0.3, 0.5, 0.8],
  'selection': ['cyclic', 'random']}}
```

In [94]:

```
best_estimator_map = {}

for model, params in model_param_map.items():
    rand = RandomizedSearchCV(model, params, n_iter=5, scoring='neg_root_mean_squared_err
    rand.fit(X_scaled, y_train)
    best_estimator_map[rand.best_estimator_] = -(rand.best_score_)

best_estimator_map
```

Out[94]:

```
{GradientBoostingRegressor(alpha=0.12, max_features='sqrt', n_estimators=1
50,
                           random_state=42): 29189.91992464586,
 RandomForestRegressor(bootstrap=False, max_features='sqrt', random_state=
42): 28854.142326816283,
 ElasticNet(l1_ratio=0.8, random_state=42, selection='random'): 31540.3735
1747742}
```

Lets create best estimators' set

Including Ensemble

In [95]:

```
best_estimator_set = []

best_gb = GradientBoostingRegressor(alpha=0.1, criterion='squared_error', loss='huber',
                           max_features='sqrt', random_state=42)

best_estimator_set.append(best_gb)

best_rf = RandomForestRegressor(criterion='friedman_mse', max_features='sqrt',
                        random_state=42)

best_estimator_set.append(best_rf)

best_en = ElasticNet(alpha=0.9, l1_ratio=0.8, random_state=42)

best_estimator_set.append(best_en)

best_vr = VotingRegressor([('best_gb', GradientBoostingRegressor(alpha=0.1, criterion='sq
                         max_features='sqrt', random_state=42)),
                             ('best_rf', RandomForestRegressor(criterion='friedman_mse', m
                        random_state=42)),
                             ('best_rn', ElasticNet(alpha=0.9, l1_ratio=0.8, random_state=

best_estimator_set.append(best_vr)

best_estimator_set
```

Out[95]:

```
[GradientBoostingRegressor(alpha=0.1, criterion='squared_error', loss='hub
er',
                           max_features='sqrt', random_state=42),
 RandomForestRegressor(criterion='friedman_mse', max_features='sqrt',
                       random_state=42),
 ElasticNet(alpha=0.9, l1_ratio=0.8, random_state=42),
 VotingRegressor(estimators=[('best_gb',
                              GradientBoostingRegressor(alpha=0.1,
                                                        criterion='squared
_error',
                                                        loss='huber',
                                                        max_features='sqr
t',
                                                        random_state=42)),
                             ('best_rf',
                              RandomForestRegressor(criterion='friedman_ms
e',
                                                    max_features='sqrt',
                                                    random_state=42)),
                             ('best_rn',
                              ElasticNet(alpha=0.9, l1_ratio=0.8,
                                         random_state=42))])]
```

In [96]:

```
def get_model(model):
    temp = str(model)
    end = temp.index('(')
    title = temp[0 : end]
    return title

def regression_score(X, y, models):
    model_list = []
    r2_scores = []
    rmse_scores = []

    for model in models:
        k = StratifiedKFold(n_splits=15, shuffle=True, random_state=seed)
        y_pred = cross_val_predict(model, X, y, cv=k)
        model_list.append(get_model(model))
        mse = mean_squared_error(y, y_pred)
        rmse = round(np.sqrt(mse), 4)
        rmse_scores.append(rmse)
        r2 = round(r2_score(y, y_pred), 4)
        r2_scores.append(r2)

    mtx = list(zip(model_list, r2_scores, rmse_scores))
    df = pd.DataFrame(mtx, columns = ['Model', 'R Squared Error', 'RMSE'])
    return df
```

In [97]:

```
regression_score(X_scaled, y_train, best_estimator_set)
```

Out[97]:

	Model	R Squared Error	RMSE
0	GradientBoostingRegressor	0.8365	32109.8476
1	RandomForestRegressor	0.8424	31527.8043
2	ElasticNet	0.8391	31853.6822
3	VotingRegressor	0.8520	30550.9968

VotingRegressor gives a better R2 and RMSE value

5. Model Implementation

In [98]:

```
final_pipeline = Pipeline([('preproc', data_pipeline),
                           ('scale', StandardScaler()),
                           ('best_vr', VotingRegressor([('best_gb', GradientBoostingRegres
                          max_features='sqrt', random_state=42)),
                              ('best_rf', RandomForestRegressor(criterion='friedman_mse', m
                         random_state=42)),
                              ('best_rn', ElasticNet(alpha=0.9, l1_ratio=0.8, random_state=

final_pipeline
```

Out[98]:

```
Pipeline(steps=[('preproc',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imp1',
                                                                   SimpleI
mputer(strategy='median'))]),
                                                  ['LotFrontage', 'LotAre
a',
                                                   'BsmtUnfSF', 'TotalBsmt
SF',
                                                   'GrLivArea', 'GarageAre
a']),
                                                 ('countable',
                                                  Pipeline(steps=[('imp2',
                                                                   SimpleI
mputer(strategy='most_frequent'))]),
                                                  ['MSSubClass', 'Overall
ual',
                                                   'OverallCond',
                                                   'KitchenAbvGr', 'Firepl
ac...
                ('scale', StandardScaler()),
                ('best_vr',
                 VotingRegressor(estimators=[('best_gb',
                                              GradientBoostingRegressor(al
pha=0.1,
                                                                        cr
iterion='squared_error',
                                                                        lo
ss='huber',
                                                                        ma
x_features='sqrt',
                                                                        ra
ndom_state=42)),
                                             ('best_rf',
                                              RandomForestRegressor(criter
ion='friedman_mse',
                                                                    max_fe
atures='sqrt',
                                                                    random
_state=42)),
                                             ('best_rn',
                                              ElasticNet(alpha=0.9,
                                                         l1_ratio=0.8,
                                                         random_state=4
2))]))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [99]:

```
# final_pipeline = Pipeline([('preproc', data_pipeline),
#                            ('scale', StandardScaler()),
#                            ('best_en', ElasticNet(alpha=0.9, l1_ratio=0.8, random_state=

# final_pipeline
```

In [100]:

```
final_pipeline.fit(X_train, y_train)
```

Out[100]:

```
Pipeline(steps=[('preproc',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imp1',
                                                                   SimpleI
mputer(strategy='median'))]),
                                                  ['LotFrontage', 'LotAre
a',
                                                   'BsmtUnfSF', 'TotalBsmt
SF',
                                                   'GrLivArea', 'GarageAre
a']),
                                                 ('countable',
                                                  Pipeline(steps=[('imp2',
                                                                   SimpleI
mputer(strategy='most_frequent'))]),
                                                  ['MSSubClass', 'Overall
ual',
                                                   'OverallCond',
                                                   'KitchenAbvGr', 'Firepl
ac...
                ('scale', StandardScaler()),
                ('best_vr',
                 VotingRegressor(estimators=[('best_gb',
                                              GradientBoostingRegressor(al
pha=0.1,
                                                                        cr
iterion='squared_error',
                                                                        lo
ss='huber',
                                                                        ma
x_features='sqrt',
                                                                        ra
ndom_state=42)),
                                             ('best_rf',
                                              RandomForestRegressor(criter
ion='friedman_mse',
                                                                    max_fe
atures='sqrt',
                                                                    random
_state=42)),
                                             ('best_rn',
                                              ElasticNet(alpha=0.9,
                                                         l1_ratio=0.8,
                                                         random_state=4
2))]))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [101]:

```
predictions = final_pipeline.predict(test)
predictions
```

Out[101]:

```
array([122118.40394472, 162889.99555077, 190684.1171349 , ...,
       167847.48999469, 120096.08135343, 219893.72646442])
```

In [102]:

```
submission = pd.DataFrame({'Id': test['Id'], 'SalePrice': predictions})
submission.head(5)
```

Out[102]:

	Id	SalePrice
0	1461	122118.403945
1	1462	162889.995551
2	1463	190684.117135
3	1464	202518.297724
4	1465	191814.608543

In [103]:

```
filename = 'house_price_prediction.csv'

submission.to_csv(filename, index=False)

print('Saved file: ' + filename)
```

```
Saved file: house_price_prediction.csv
```

In [104]:

```
sub = pd.read_csv('house_price_prediction.csv')
sub.shape
```

Out[104]:

```
(1459, 2)
```

In [ ]: