PRCP-1020-House Price Prediction using Advanced Regression

Parimal #Problem Statement Task 1:- Prepare a complete data analysis report on the given data. Task 2: a) Create a robust machine learning algorithm to accurately predict the price of the house given the various factors across the market. b) Determine the relationship between the house features and how the price varies based on this. Task3:- Come up with suggestions for the customer to buy the house according to the area, price and other requirements.

Importing important libraires

In [ ]:
```
# Importing libraries
import numpy as np
import pandas as pd
# Visaulize the plot
import matplotlib.pyplot as plt
#%matplotlibinline
import seaborn as sns
#remove warniongs
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
```

Read data set -

In [4]:
```
# load dataset :
df=pd.read_csv("data.csv")
```

In [4]:
```
df # dataframe of 1460 rows and 81 columns
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	A
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	A
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	A
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	A
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	A
...	...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	A
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	A
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	A
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	A
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	A

1460 rows × 81 columns

In [2]:
```
df.shape # number of columns are 1460 and columns are 81
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_8708\900203781.py in <module>
----> 1 df.shape # number of columns are 1460 and columns are 81

NameError: name 'df' is not defined
```

In [4]:
```
df.columns
```

Out[4]:
```
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 [7]:
```
df.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 [5]:
```
# To show the all columns
pd.set_option("display.max_columns", 2000)
pd.set_option("display.max_rows", 85)
```

In [6]:
```
df
```

Out[6]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Ut
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	A
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	A
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	A
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	A
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	A
...	...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl	A
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl	A
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl	A
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl	A
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl	A

1460 rows × 81 columns

In [6]:
```
# Set index as Id
df = df.set_index("Id")
```

In [8]:
```
df.head(8) #first 8rows
```

Out[8]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lo
1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	
2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	
3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	
4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	
5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	
6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl	AllPub	
7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl	AllPub	
8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl	AllPub	

Id column set as index column so no need to remove it # Not necessary step because we set index as id column in df # OR you can drop unimportant column ID : df.drop(df['Id'], axis = 1, inplace = True)

In [10]:
```
df.describe() # shows stastical of numerical features
```

Out[10]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAd
count	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.0000
mean	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.8657
std	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.6454
min	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.0000
25%	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.0000
50%	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.0000
75%	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.0000
max	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.0000

In [8]:
```
df.describe().T #T -CHANGES ROWS AND COLUMNS POSITION
```

Out[8]:

	count	mean	std	min	25%	50%	75%	ma
MSSubClass	1460.0	56.897260	42.300571	20.0	20.00	50.0	70.00	190.
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.
OverallQual	1460.0	6.099315	1.382997	1.0	5.00	6.0	7.00	10.
OverallCond	1460.0	5.575342	1.112799	1.0	5.00	5.0	6.00	9.
YearBuilt	1460.0	1971.267808	30.202904	1872.0	1954.00	1973.0	2000.00	2010.
YearRemodAdd	1460.0	1984.865753	20.645407	1950.0	1967.00	1994.0	2004.00	2010.
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.
BsmtFullBath	1460.0	0.425342	0.518911	0.0	0.00	0.0	1.00	3.
BsmtHalfBath	1460.0	0.057534	0.238753	0.0	0.00	0.0	0.00	2.
FullBath	1460.0	1.565068	0.550916	0.0	1.00	2.0	2.00	3.
HalfBath	1460.0	0.382877	0.502885	0.0	0.00	0.0	1.00	2.
BedroomAbvGr	1460.0	2.866438	0.815778	0.0	2.00	3.0	3.00	8.
KitchenAbvGr	1460.0	1.046575	0.220338	0.0	1.00	1.0	1.00	3.
TotRmsAbvGrd	1460.0	6.517808	1.625393	2.0	5.00	6.0	7.00	14.
Fireplaces	1460.0	0.613014	0.644666	0.0	0.00	1.0	1.00	3.
GarageYrBlt	1379.0	1978.506164	24.689725	1900.0	1961.00	1980.0	2002.00	2010.
GarageCars	1460.0	1.767123	0.747315	0.0	1.00	2.0	2.00	4.
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.

	count	mean	std	min	25%	50%	75%	ma
MiscVal	1460.0	43.489041	496.123024	0.0	0.00	0.0	0.00	15500.
MoSold	1460.0	6.321918	2.703626	1.0	5.00	6.0	8.00	12.
YrSold	1460.0	2007.815753	1.328095	2006.0	2007.00	2008.0	2009.00	2010.
SalePrice	1460.0	180921.195890	79442.502883	34900.0	129975.00	163000.0	214000.00	755000.

INSIGHTS- FROM describe() function:gives Statistics Summary- -Statistics summary gives a high-level idea to identify whether the data has any outliers, data entry error, distribution of data such as the data is normally distributed or left/right skewness. - 1. SalePrice ,MiscVal, YrSold , GarageYrBlt, GrLivArea ,LotArea,YearBuilt,YearRemodAdd,LotArea, LotFrontage, all basement features ,1stFlrS,2ndtFlrS has large scale range ,so need to apply scaling on dataset. 2. Some fetures -lotarea ,MasVnrArea, MiscVal large outliers are present and also in basement features 3. all years features shows large gap in years yrbuilt and year sold large differences so we have to convert into same format. 4. large scale parameters should apply scaling and also use log tranformation for normall distributution. 5. Large outliers columns should we can remove or drop some outliers. 6. Drop not important columns having zeros values, or same constant value,it is not affecting the output such columns are Utilities, Alley etc.

In [9]:
```
df.describe(include='O')# categorical columns
```

Out[9]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborh
count	1460	1460	91	1460	1460	1460	1460	1460	1
unique	5	2	2	4	4	2	5	3	
top	RL	Pave	Grvl	Reg	Lvl	AllPub	Inside	Gtl	NA
freq	1151	1454	50	925	1311	1459	1052	1382	

In [13]:
```
df.dtypes # data types of each column
```

Out[13]:

```
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
```

from above information need to convert all features into numerical and same datatype format.

In [16]:
```
# value counts of each categorical variables , with thier unique values ,most frequent
print(df.describe(include='O').value_counts)
```

```
<bound method DataFrame.value_counts of       MSZoning Street Alley LotShape LandCon
tour Utilities LotConfig  \
count      1460   1460    91     1460        1460      1460      1460
unique        5      2     2        4           4         2         5
top          RL   Pave  Grvl      Reg         Lvl    AllPub    Inside
freq       1151   1454    50      925        1311      1459      1052

       LandSlope Neighborhood Condition1 Condition2 BldgType HouseStyle  \
count       1460         1460       1460       1460     1460       1460
unique         3           25          9          8        5          8
top          Gtl        NAmes       Norm       Norm     1Fam     1Story
freq        1382          225       1260       1445     1220        726

       RoofStyle RoofMatl Exterior1st Exterior2nd MasVnrType ExterQual  \
count       1460     1460        1460        1460       1452      1460
unique         6        8          15          16          4         4
top        Gable  CompShg     VinylSd     VinylSd       None        TA
freq        1141     1434         515         504        864       906

       ExterCond Foundation BsmtQual BsmtCond BsmtExposure BsmtFinType1  \
count       1460       1460     1423     1423         1422         1423
unique         5          6        4        4            4            6
top           TA      PConc       TA       TA           No          Unf
freq        1282        647      649     1311          953          430

       BsmtFinType2 Heating HeatingQC CentralAir Electrical KitchenQual  \
count          1422    1460      1460       1460       1459        1460
unique            6       6         5          2          5           4
top             Unf    GasA        Ex          Y      SBrkr          TA
freq           1256    1428       741       1365       1334         735

       Functional FireplaceQu GarageType GarageFinish GarageQual GarageCond  \
count        1460         770       1379         1379       1379       1379
unique          7           5          6            3          5          5
top           Typ          Gd     Attchd          Unf         TA         TA
freq         1360         380        870          605       1311       1326

       PavedDrive PoolQC  Fence MiscFeature SaleType SaleCondition
count        1460      7    281          54     1460          1460
unique          3      3      4           4        9             6
top             Y     Gd  MnPrv        Shed       WD        Normal
freq         1340      3    157          49     1267          1198  >
```

In [10]:
```
df.shape # columns becomes 80
```

Out[10]:
```
(1460, 80)
```

In [7]:
```
# Drop duplicates
'''the more rows of values and columns have the same values or are duplicates.
    Therefore its very important for you to remove duplicates from the dataset to mair
    and to avoid misleading statistics. '''
df = df.drop_duplicates()
print( df.shape )
# no duplicates in given datasets
```

```
(1460, 80)
```

In [12]:
```
# find unique value
print(df.nunique().sort_values(ascending=False))
```

```
LotArea          1073
GrLivArea         861
BsmtUnfSF         780
1stFlrSF          753
TotalBsmtSF       721
SalePrice         663
BsmtFinSF1        637
GarageArea        441
2ndFlrSF          417
MasVnrArea        327
WoodDeckSF        274
OpenPorchSF       202
BsmtFinSF2        144
EnclosedPorch     120
YearBuilt         112
LotFrontage       110
GarageYrBlt        97
ScreenPorch        76
YearRemodAdd       61
Neighborhood       25
LowQualFinSF       24
MiscVal            21
3SsnPorch          20
Exterior2nd        16
MSSubClass         15
Exterior1st        15
MoSold             12
TotRmsAbvGrd       12
OverallQual        10
SaleType            9
Condition1          9
OverallCond         9
HouseStyle          8
PoolArea            8
Condition2          8
BedroomAbvGr        8
RoofMatl            8
Functional          7
BsmtFinType2        6
Heating             6
GarageType          6
BsmtFinType1        6
SaleCondition       6
Foundation          6
RoofStyle           6
FireplaceQu         5
LotConfig           5
GarageCond          5
GarageCars          5
BldgType            5
GarageQual          5
Electrical          5
MSZoning            5
HeatingQC           5
YrSold              5
ExterCond           5
Fence               4
MiscFeature         4
LotShape            4
LandContour         4
```

```
MasVnrType          4
KitchenQual         4
FullBath            4
BsmtQual            4
Fireplaces          4
BsmtCond            4
ExterQual           4
KitchenAbvGr        4
BsmtExposure        4
BsmtFullBath        4
LandSlope           3
BsmtHalfBath        3
PoolQC              3
GarageFinish        3
HalfBath            3
PavedDrive          3
Street              2
Alley               2
Utilities           2
CentralAir          2
dtype: int64
```

2. EXPLORATORY DATA ANALYSIS -with data analysis

-Using EDA :Visualize fesatures, insight /observation from the data

- Missing Values
- All The Numerical Variables
- Distribution of the Numerical Variables
- Categorical Variables
- Cardinality of Categorical Variables
- Outliers

In [8]:
```
# Seperate numerical & categorical columns
cat_cols=df.select_dtypes(include=['object']).columns # categorical columns
num_cols = df.select_dtypes(include=np.number).columns.tolist() # numerical columns


print("Categorical Variables:", len(cat_cols))
print(cat_cols)
```

```
Categorical Variables: 43
Index(['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'],
      dtype='object')
```

In [9]:
```
#Numerical variables:
print("Numerical Variables:", len(num_cols))
print(num_cols)
```

```
Numerical Variables: 37
['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF',
'1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath',
'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorc
h', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SalePric
e']
```

In [10]:
```
df.select_dtypes(include=['int64', 'float64']).columns # shows integer ,floating numer
```

Out[10]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
```

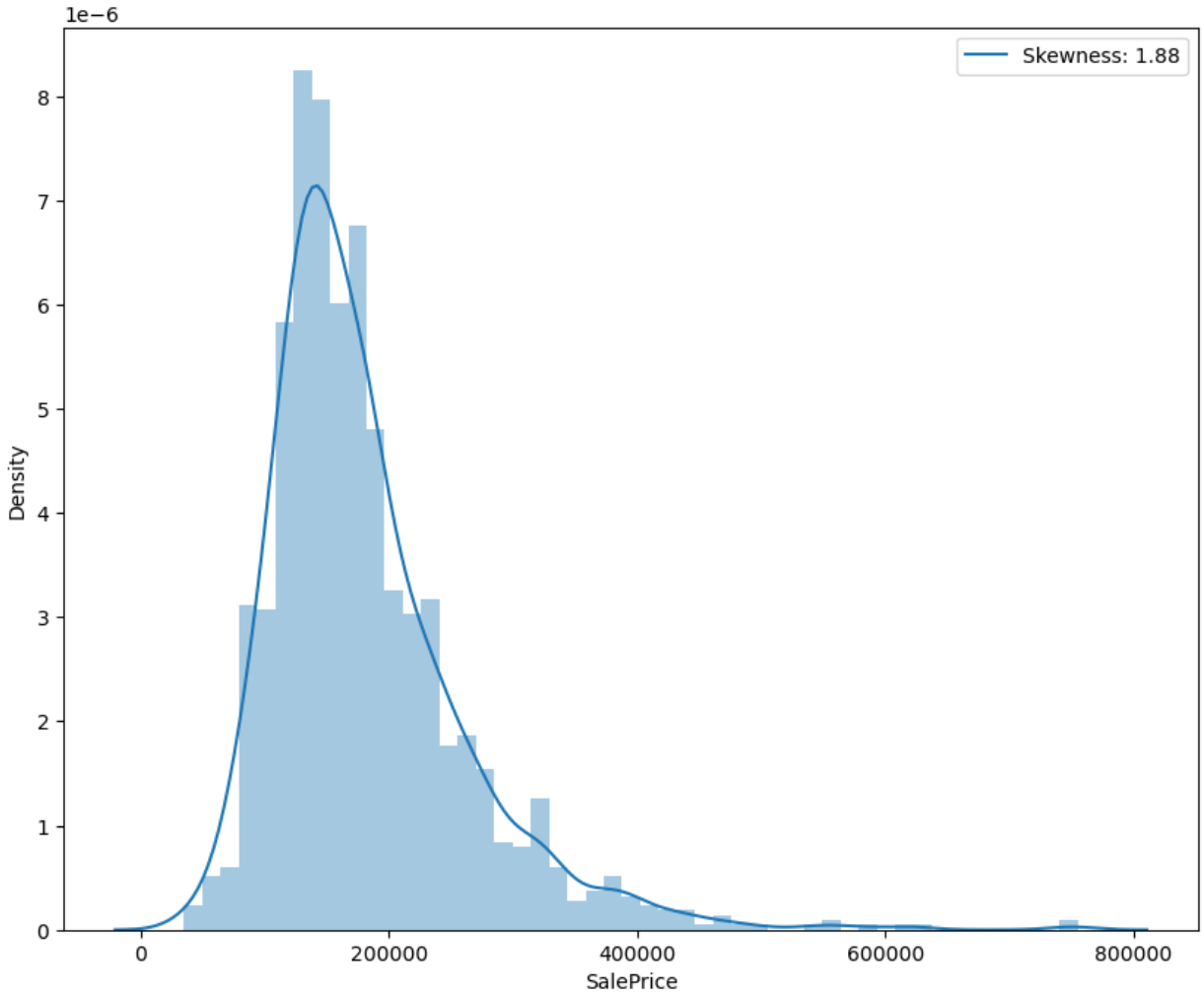
Check the distribution of target column -

- log transformation due to right positive skewness:
- It should be normal distribution

In [11]:
```
# check the distrubution of target varible i.e SalePrcie
df["SalePrice"].describe()
```

Out[11]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

In [12]:
```
# Plot the distplot of target
plt.figure(figsize=(10,8))
bar = sns.distplot(df["SalePrice"],kde=True)
bar.legend(["Skewness: {:.2f}".format(df['SalePrice'].skew())])
plt.show()
```



here sales price normaly distrubeted with right skewness

log transformation on sale price -right skewness

this indicates that output variable Sale price is right skewed so i applied log transoformation.

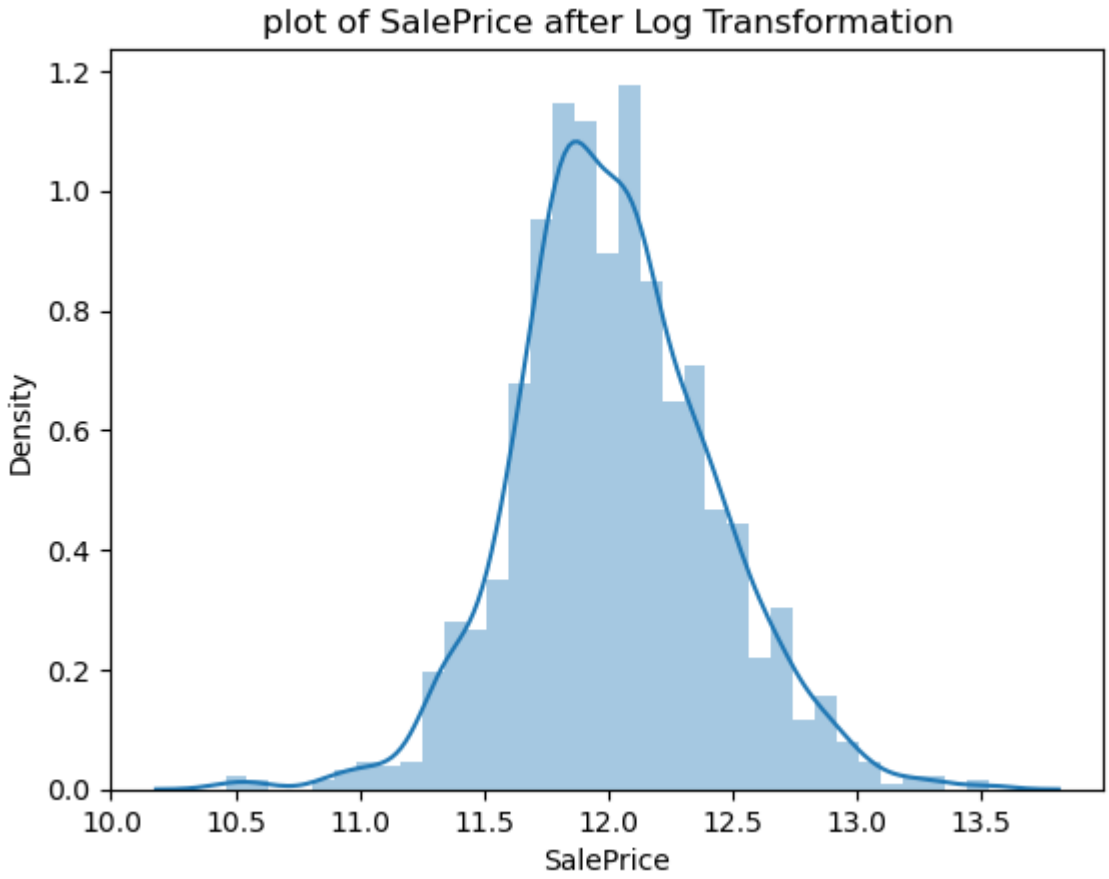
In [13]:
```
# Positive Skeweness:
df['SalePrice'].skew()
```

Out[13]:
```
1.8828757597682129
```

In [14]:
```
df["SalePrice"] = np.log1p(df["SalePrice"]) # Log transoformation
```

In [15]:
```
 #SalePrice after Log-transformation
sns.distplot(df["SalePrice"])
plt.title("plot of SalePrice after Log Transformation")
```

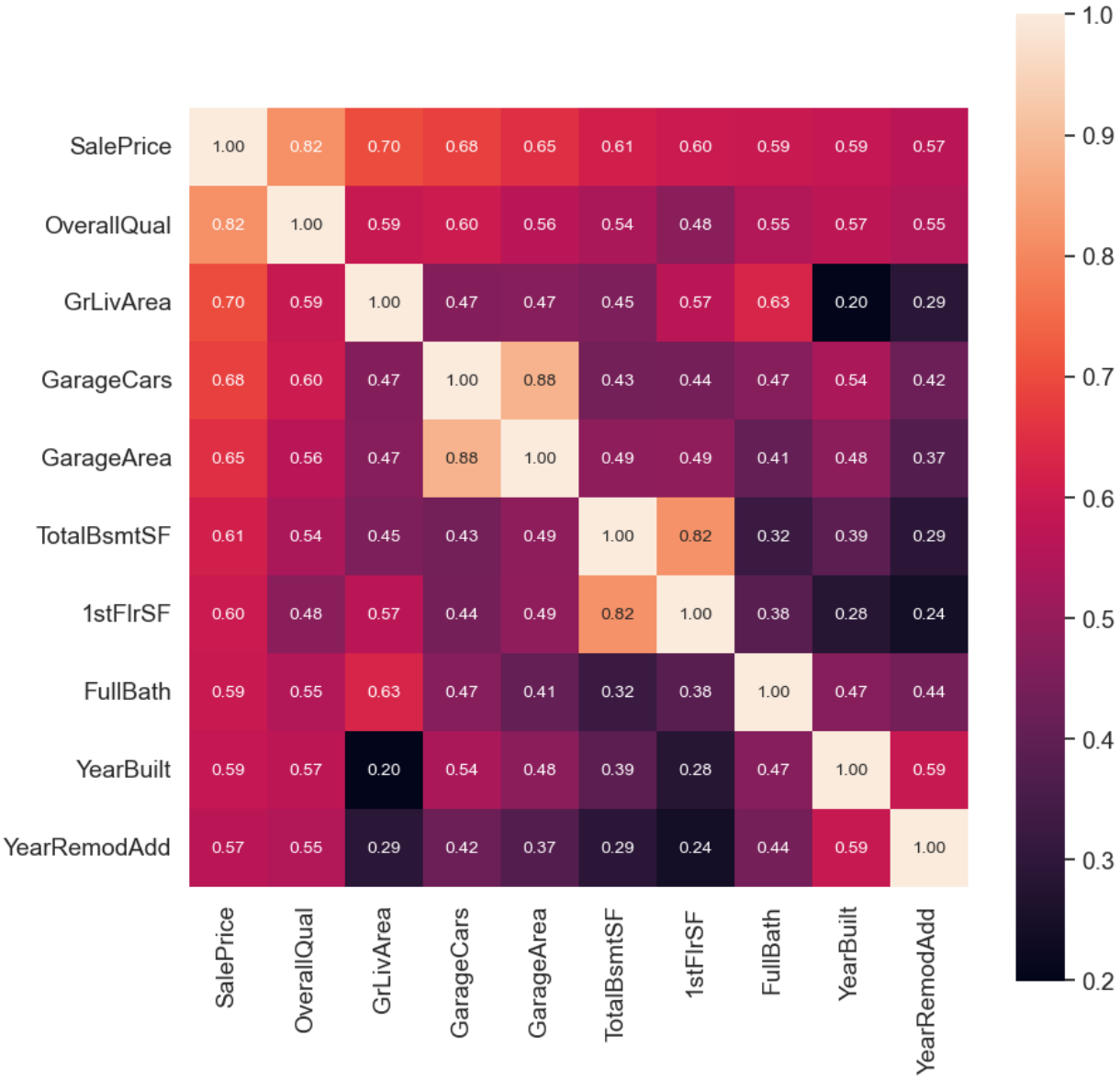
Out[15]:
```
Text(0.5, 1.0, 'plot of SalePrice after Log Transformation')
```



In [16]:
```
# again check skewness :
# Positive Skeweness:
df['SalePrice'].skew() # very less value of skewness .
```

Out[16]:
```
0.12134661989685333
```

In [19]:
```
# Before going to data prerocessing  find out top variables affecting sale price:
#saleprice correlation matrix of most top 10 varaibles
corrmat = df.corr()
k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
cm = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1.25)
plt.figure(figsize=(10,10))
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size'
plt.show()
print("affecting target varaible most : ", cols, len(cols))
```




```
affecting target varaible most :  Index(['SalePrice', 'OverallQual', 'GrLivArea', 'Ga
rageCars', 'GarageArea',
       'TotalBsmtSF', '1stFlrSF', 'FullBath', 'YearBuilt', 'YearRemodAdd'],
      dtype='object') 10
```

In [17]:
```
#values of correlation : highly positively correlated features
abs(df.corr()['SalePrice']).nlargest(10)
```

Out[17]:
```
SalePrice       1.000000
OverallQual     0.817185
GrLivArea       0.700927
GarageCars      0.680625
GarageArea      0.650888
TotalBsmtSF     0.612134
1stFlrSF        0.596981
FullBath        0.594771
YearBuilt       0.586570
YearRemodAdd    0.565608
Name: SalePrice, dtype: float64
```

from above heatmap these are highly 10 affecting on sale price :see above result 1. columns : 'OverallQual', 'GrLivArea', 'GarageCars', 'GarageArea', 'TotalBsmtSF', '1stFlrSF', 'FullBath', 'YearBuilt', 'YearRemodAdd 2.But garage cars and garage area some how same feture affecting so we keep garagearea ,drop garage cars. 3.Also reduce

some columns with adding some features into one column so we can reducing number of columns. 4. transfoming large value range into log transoformation . check describe() function thier scale.

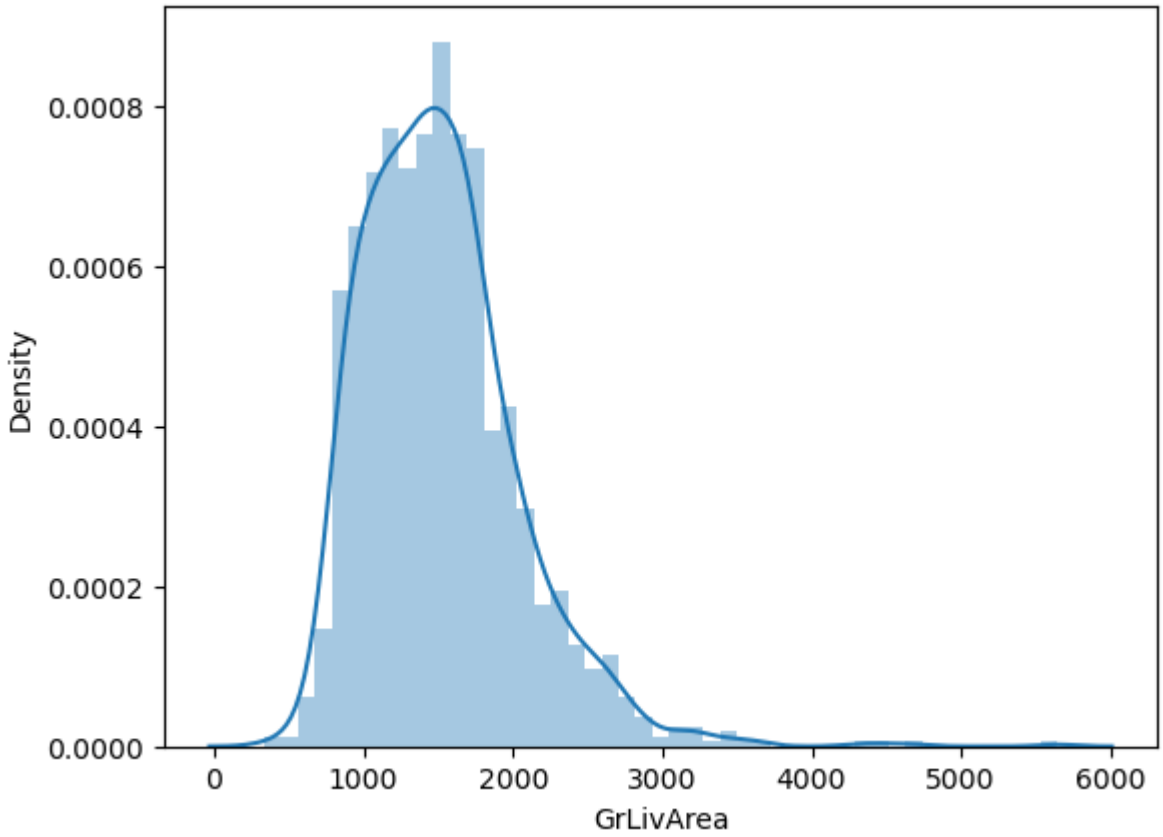
In [ ]:
```
# Transforming the target variable, GrLivArea and OverallQualto log values so that the
```

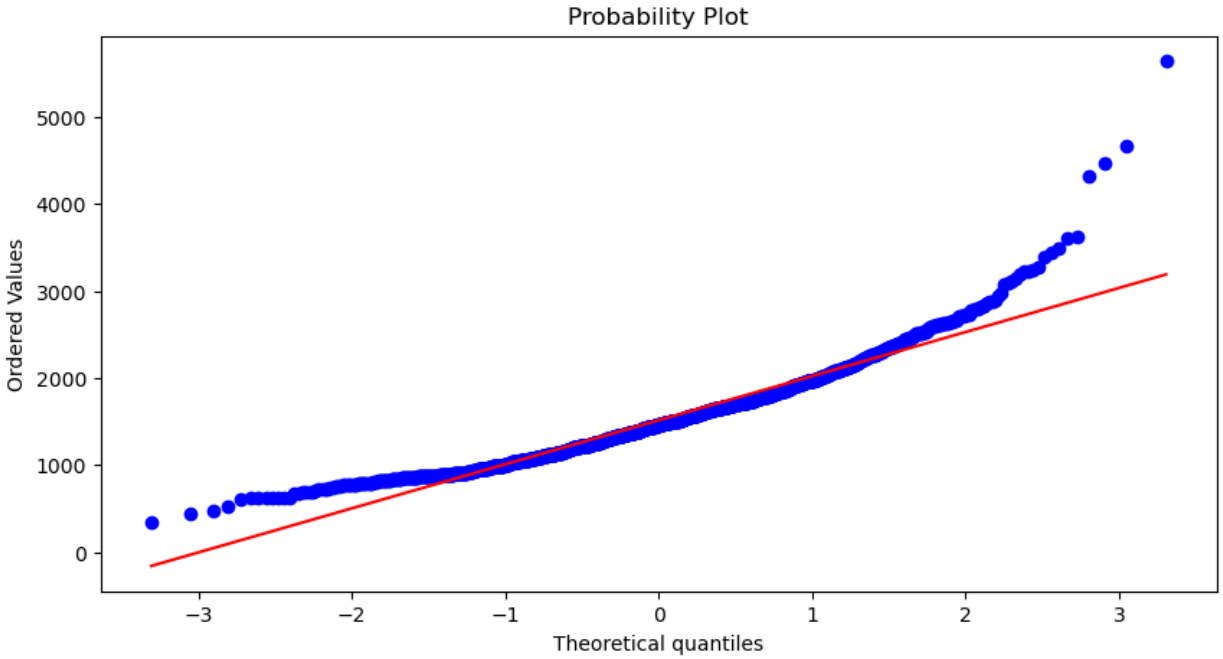
In [18]:
```
print(df['GrLivArea'].skew(),df['TotalBsmtSF'].skew(), df['1stFlrSF'].skew(),
      df['GarageArea'].skew())
#check skew  greater than 1
```

```
1.3665603560164552 1.5242545490627664 1.3767566220336365 0.17998090674623907
```

In [19]:
```
from scipy import stats  # from scipy package import stats
```

In [20]:
```
sns.distplot(df['GrLivArea']);
fig_GrLivArea = plt.figure(figsize=(10,5))
result1 = stats.probplot(df['GrLivArea'],plot = plt)
plt.show()
```





In [20]:
```
df['GrLivArea'] = np.log1p(df['GrLivArea'])
print("AFTER LOG TRANSFORMATION  GrLivArea SKEWNESS :", df['GrLivArea'].skew())
```

```
AFTER LOG TRANSFORMATION  GrLivArea SKEWNESS : -0.006140253486287281
```

print("after handling skewness") print(df['GrLivArea'].skew(),df['TotalBsmtSF'].skew(), df['1stFlrSF'].skew(), df['GarageArea'].skew()) # check later of #df['OverallQual'].skew(),df['SalePrice'].skew(),df['TotalBsmtSF'].skew(), df['1stFlrSF'].skew(), df['GarageArea'].skew()) # we also do later before model creation.

3. Datapreprocessing

- Data preprocessing- is the process of cleaning and preparing the raw data to enable feature engineering.
- Feature Engineering covers various data engineering techniques such as adding/removing relevant features, handling missing data, encoding the data, handling categorical variables, etc
- handling missing values
- handling outliers
- drop duplicates
- handling categorical varaibles
- scaling

In [23]:
```
#sum of missing data
df.isnull().sum().sort_values(ascending=False)
```

Out[23]:

PoolQC	1453
MiscFeature	1406
Alley	1369
Fence	1179
FireplaceQu	690
LotFrontage	259
GarageYrBlt	81
GarageCond	81
GarageType	81
GarageFinish	81
GarageQual	81
BsmtExposure	38
BsmtFinType2	38
BsmtCond	37
BsmtQual	37
BsmtFinType1	37
MasVnrArea	8
MasVnrType	8
Electrical	1
MSSubClass	0
Fireplaces	0
Functional	0
KitchenQual	0
KitchenAbvGr	0
BedroomAbvGr	0
HalfBath	0
FullBath	0
BsmtHalfBath	0
TotRmsAbvGrd	0
GarageCars	0
GrLivArea	0
GarageArea	0
PavedDrive	0
WoodDeckSF	0
OpenPorchSF	0
EnclosedPorch	0
3SsnPorch	0
ScreenPorch	0
PoolArea	0
MiscVal	0
MoSold	0
YrSold	0
SaleType	0
SaleCondition	0
BsmtFullBath	0
CentralAir	0
LowQualFinSF	0
Neighborhood	0
OverallCond	0
OverallQual	0
HouseStyle	0
BldgType	0
Condition2	0
Condition1	0
LandSlope	0
2ndFlrSF	0
LotConfig	0
Utilities	0
LandContour	0
LotShape	0

```
Street             0
LotArea            0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
ExterQual          0
ExterCond          0
Foundation         0
BsmtFinSF1         0
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
MSZoning           0
1stFlrSF           0
SalePrice          0
dtype: int64
```

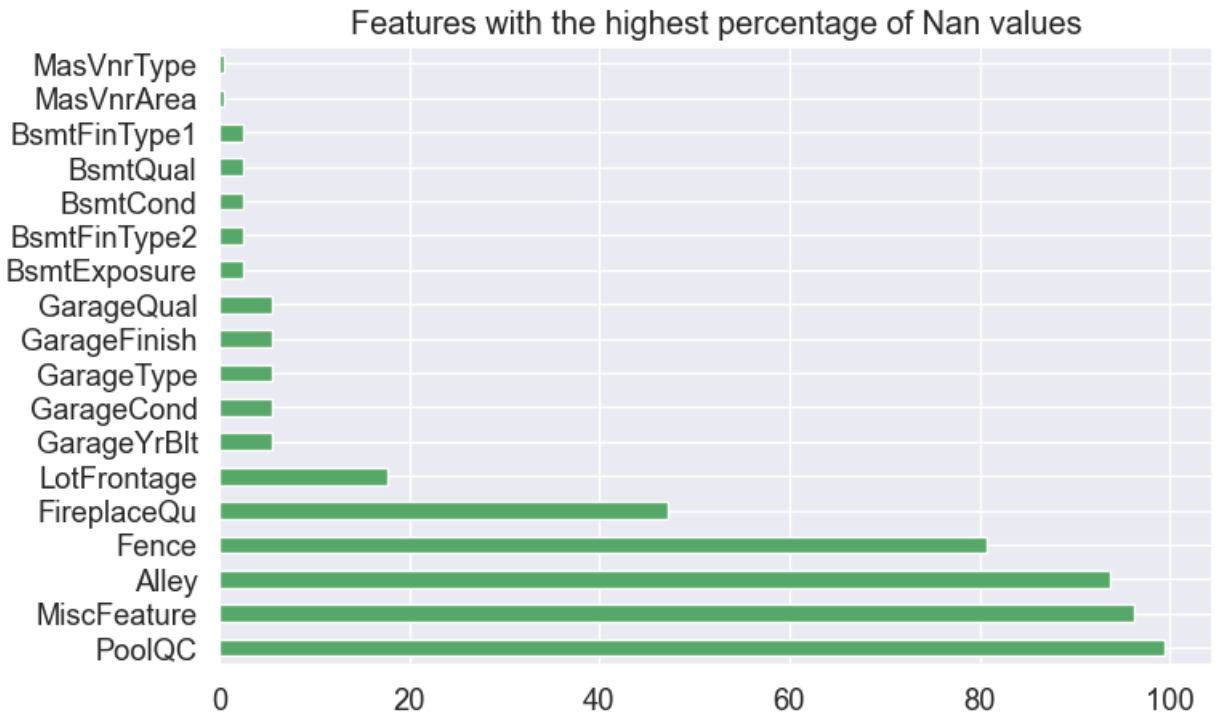
In [21]:
```
# Get the percentages of null values of each column
null_percent = (df.isnull().sum()/df.shape[0]*100).sort_values(ascending=False)
null_percent
```

Out[21]:

PoolQC	99.520548
MiscFeature	96.301370
Alley	93.767123
Fence	80.753425
FireplaceQu	47.260274
LotFrontage	17.739726
GarageYrBlt	5.547945
GarageCond	5.547945
GarageType	5.547945
GarageFinish	5.547945
GarageQual	5.547945
BsmtExposure	2.602740
BsmtFinType2	2.602740
BsmtCond	2.534247
BsmtQual	2.534247
BsmtFinType1	2.534247
MasVnrArea	0.547945
MasVnrType	0.547945
Electrical	0.068493
MSSubClass	0.000000
Fireplaces	0.000000
Functional	0.000000
KitchenQual	0.000000
KitchenAbvGr	0.000000
BedroomAbvGr	0.000000
HalfBath	0.000000
FullBath	0.000000
BsmtHalfBath	0.000000
TotRmsAbvGrd	0.000000
GarageCars	0.000000
GrLivArea	0.000000
GarageArea	0.000000
PavedDrive	0.000000
WoodDeckSF	0.000000
OpenPorchSF	0.000000
EnclosedPorch	0.000000
3SsnPorch	0.000000
ScreenPorch	0.000000
PoolArea	0.000000
MiscVal	0.000000
MoSold	0.000000
YrSold	0.000000
SaleType	0.000000
SaleCondition	0.000000
BsmtFullBath	0.000000
CentralAir	0.000000
LowQualFinSF	0.000000
Neighborhood	0.000000
OverallCond	0.000000
OverallQual	0.000000
HouseStyle	0.000000
BldgType	0.000000
Condition2	0.000000
Condition1	0.000000
LandSlope	0.000000
2ndFlrSF	0.000000
LotConfig	0.000000
Utilities	0.000000
LandContour	0.000000
LotShape	0.000000

```
Street           0.000000
LotArea          0.000000
YearBuilt        0.000000
YearRemodAdd     0.000000
RoofStyle        0.000000
RoofMatl         0.000000
Exterior1st      0.000000
Exterior2nd      0.000000
ExterQual        0.000000
ExterCond        0.000000
Foundation       0.000000
BsmtFinSF1       0.000000
BsmtFinSF2       0.000000
BsmtUnfSF        0.000000
TotalBsmtSF      0.000000
Heating          0.000000
HeatingQC        0.000000
MSZoning         0.000000
1stFlrSF         0.000000
SalePrice        0.000000
dtype: float64
```

In [42]:
```
# Shows null values bar graph
plt.figure(figsize=(8, 5))

sns.set(font_scale=1.2)
null_percent[null_percent > 0.1].plot(kind = "barh",color='g') # horizontal bar
plt.title("Features with the highest percentage of Nan values")
plt.show()
```



In [22]:
```
# We have to drop some columns which contains large number of null values and more tha
#  DROP THESE 5 features  for model
drop_variables = ['Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature']
df.drop(drop_variables, axis = 1, inplace=True)

# Now, both the columns with more than 90% missing values and their respective rows ar
```

In [23]:
```
df.dropna(subset=drop_variables, inplace=True) # corresponding null rows also delete.
df
# # 5 columns get droped
# most null values columns are
- ALLEY ,PoolQC  ,MiscFeature,  Fence, FireplaceQu
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_924\891033548.py in <module>
----> 1 df.dropna(subset=drop_variables, inplace=True) # corresponding null rows also
delete.
      2 df
      3 # # 5 columns get droped
      4 # most null values columns are
      5 - ALLEY ,PoolQC  ,MiscFeature,  Fence, FireplaceQu

~\anaconda3\lib\site-packages\pandas\util\_decorators.py in wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~\anaconda3\lib\site-packages\pandas\core\frame.py in dropna(self, axis, how, thresh,
subset, inplace)
   6009             check = indices == -1
   6010             if check.any():
-> 6011                 raise KeyError(np.array(subset)[check].tolist())
   6012             agg_obj = self.take(indices, axis=agg_axis)
   6013

KeyError: ['Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature']
```

In [24]:
```
df.shape
```

Out[24]:
```
(1460, 75)
```

In [25]:
```
# some columns are missing values  missing values .greater than 10%

col_nan = df.isna().sum() / df.shape[0]
```

In [26]:
```
col_nan.sort_values(ascending=False)
```

Out[26]:

LotFrontage	0.177397
GarageType	0.055479
GarageYrBlt	0.055479
GarageFinish	0.055479
GarageQual	0.055479
GarageCond	0.055479
BsmtFinType2	0.026027
BsmtExposure	0.026027
BsmtQual	0.025342
BsmtCond	0.025342
BsmtFinType1	0.025342
MasVnrArea	0.005479
MasVnrType	0.005479
Electrical	0.000685
KitchenAbvGr	0.000000
BedroomAbvGr	0.000000
HalfBath	0.000000
FullBath	0.000000
BsmtHalfBath	0.000000
BsmtFullBath	0.000000
KitchenQual	0.000000
GrLivArea	0.000000
TotRmsAbvGrd	0.000000
Functional	0.000000
MSSubClass	0.000000
Fireplaces	0.000000
ScreenPorch	0.000000
SaleCondition	0.000000
SaleType	0.000000
YrSold	0.000000
MoSold	0.000000
MiscVal	0.000000
PoolArea	0.000000
3SsnPorch	0.000000
2ndFlrSF	0.000000
EnclosedPorch	0.000000
OpenPorchSF	0.000000
WoodDeckSF	0.000000
PavedDrive	0.000000
GarageArea	0.000000
GarageCars	0.000000
LowQualFinSF	0.000000
Heating	0.000000
1stFlrSF	0.000000
CentralAir	0.000000
LotArea	0.000000
Street	0.000000
LotShape	0.000000
LandContour	0.000000
Utilities	0.000000
LotConfig	0.000000
LandSlope	0.000000
Neighborhood	0.000000
Condition1	0.000000
Condition2	0.000000
BldgType	0.000000
HouseStyle	0.000000
OverallQual	0.000000
OverallCond	0.000000
YearBuilt	0.000000

```
YearRemodAdd     0.000000
RoofStyle        0.000000
RoofMatl         0.000000
Exterior1st      0.000000
Exterior2nd      0.000000
ExterQual        0.000000
ExterCond        0.000000
Foundation       0.000000
BsmtFinSF1       0.000000
BsmtFinSF2       0.000000
BsmtUnfSF        0.000000
TotalBsmtSF      0.000000
MSZoning         0.000000
HeatingQC        0.000000
SalePrice        0.000000
dtype: float64
```

In [27]:
```
df.columns[df.isnull().any()] # these columns having missing values
```

Out[27]:
```
Index(['LotFrontage', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical',
       'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual',
       'GarageCond'],
      dtype='object')
```

In [28]:
```
categorical_cols = df.select_dtypes(include=['object']).columns
numerical_cols = df.select_dtypes(include=[np.number]).columns

print("categorical_cols", categorical_cols,len(categorical_cols))
print("numerical_cols: ", numerical_cols, len(numerical_cols))

# after droping some columns :
#categorical_cols=38 and  numerical_cols :37
```

```
categorical_cols Index(['MSZoning', 'Street', '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', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
       'PavedDrive', 'SaleType', 'SaleCondition'],
      dtype='object') 38
numerical_cols:  Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'Overa
llCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object') 37
```

In [29]:
```
#missing categorical as well as numerical column list :

categorical_cols_with_missing = df[categorical_cols].columns[df[categorical_cols].isnu

numerical_cols_with_missing = df[numerical_cols].columns[df[numerical_cols].isnull().a
```

```
print("categorical_cols_with_missing are :",len(categorical_cols_with_missing))

print("numerical_cols_with_missing are :",len(numerical_cols_with_missing))
```

```
categorical_cols_with_missing are : 11
numerical_cols_with_missing are : 3
```

HANDLING MISSING VALUES MEDIAN AND MODE USING FOR LOOP[:]

In [30]:
```
# handle with mean ,mode median of missing value using for loop :
for col in numerical_cols_with_missing:
    df[col].fillna(df[col].median(), inplace=True) # change with median value



for col in categorical_cols_with_missing:
    mode_val = stats.mode(df[col]).mode[0] # replace categorical columns by mode value
    df[col].fillna(mode_val, inplace=True)
print(df.isnull().sum().sum()) # no null values are present
```

```
0
```

CONVERT COLUMNS INTO SAME DATATYPE

- Need to be - all datatype features is convert into same datatype , so convert float into int 64

In [31]:
```
# Need to all datatype into same datatype so convert float into int 64
# LotFrontage,  MasVnrArea - float 64 type into int 64
# convert sale price into int 64 -SalePrice
df['LotFrontage'] = df['LotFrontage'].astype(np.int64)
df['MasVnrArea'] = df['MasVnrArea'].astype(np.int64)

df['SalePrice'] = df['SalePrice'].astype(np.int64)
df['GarageYrBlt']=df['GarageYrBlt'].astype(np.int64)
```

OUTLIER DETECTION AND REMOVAL : MOST IMP

Removing outliers is important step in data analysis. # However, while removing outliers in ML we should be careful, because we do not know if there are not any outliers in test set. # checked the outliers then decide to drop outliers or handle the outliers.

Detecting outliers check skewness and kurtosis of columns as well ,

so we will get is it normaly distributed or not ?

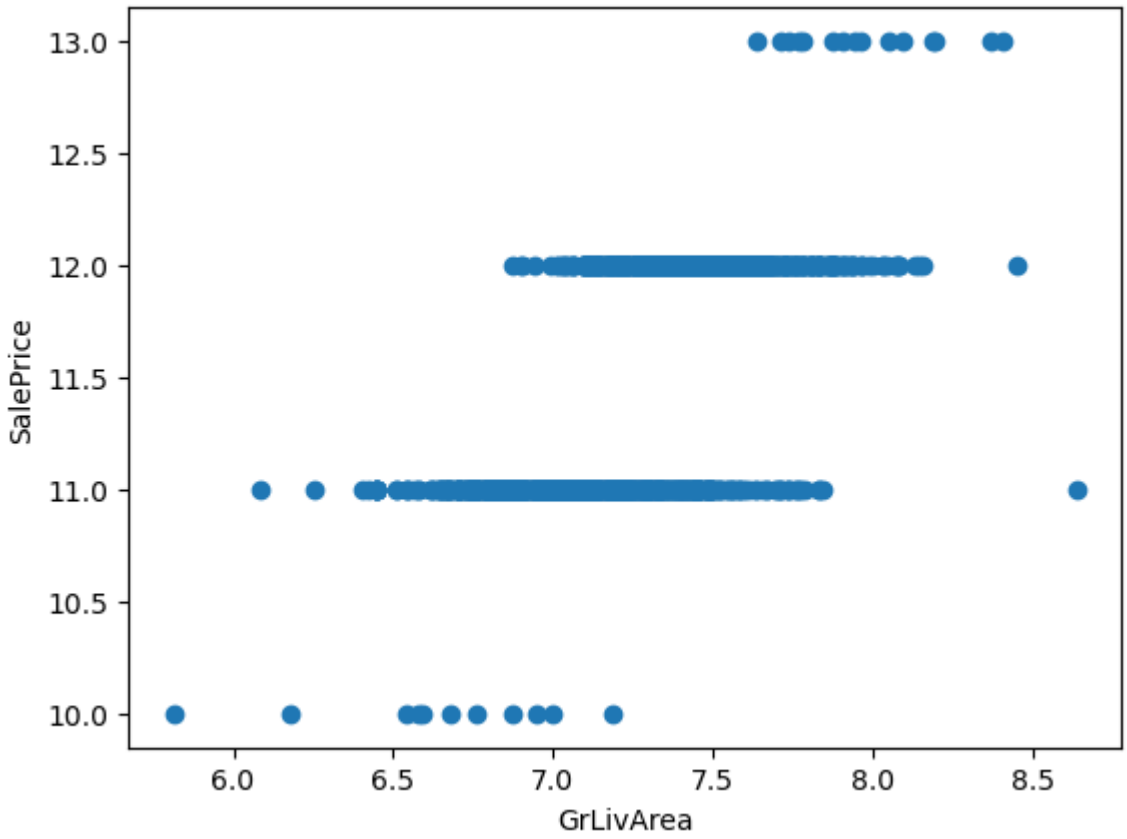
In [32]:
```
#NORMALLY DISTRIBUTED OR NOT ?
from scipy.stats import skew, kurtosis

numerical_cols = df.select_dtypes(include=[np.number]).columns
columns = numerical_cols # Replace with the actual column names

# Check normality for each column-

for column in columns:
    data = df[column]
```

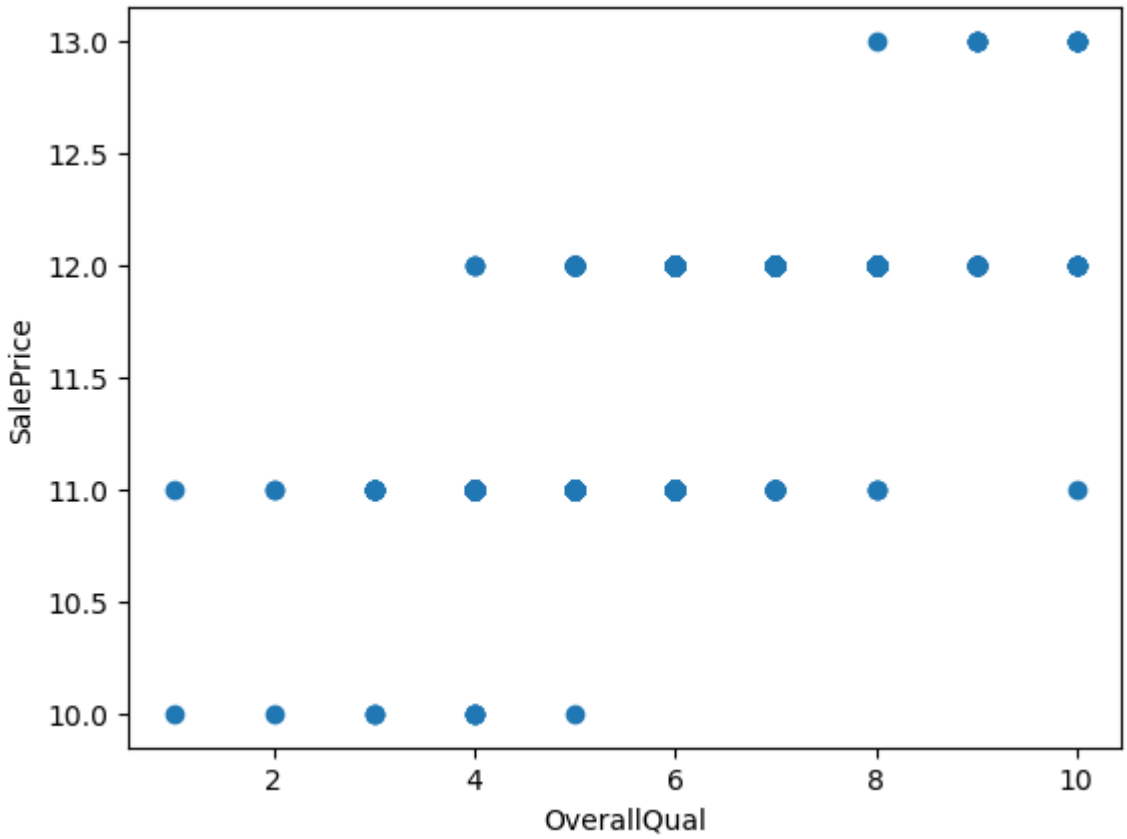
```
    # Calculate skewness and kurtosis
    skewness = skew(data)
    kurt = kurtosis(data)

    # Assess normality:
    if abs(skewness) <= 1 and abs(kurt) <= 3:  # skewness =1 and Kurtsosi= 3
        print(f"Column:{column} is normally distributed")
    else:
        print(f"Column:{column} is not normally distributed" )
```

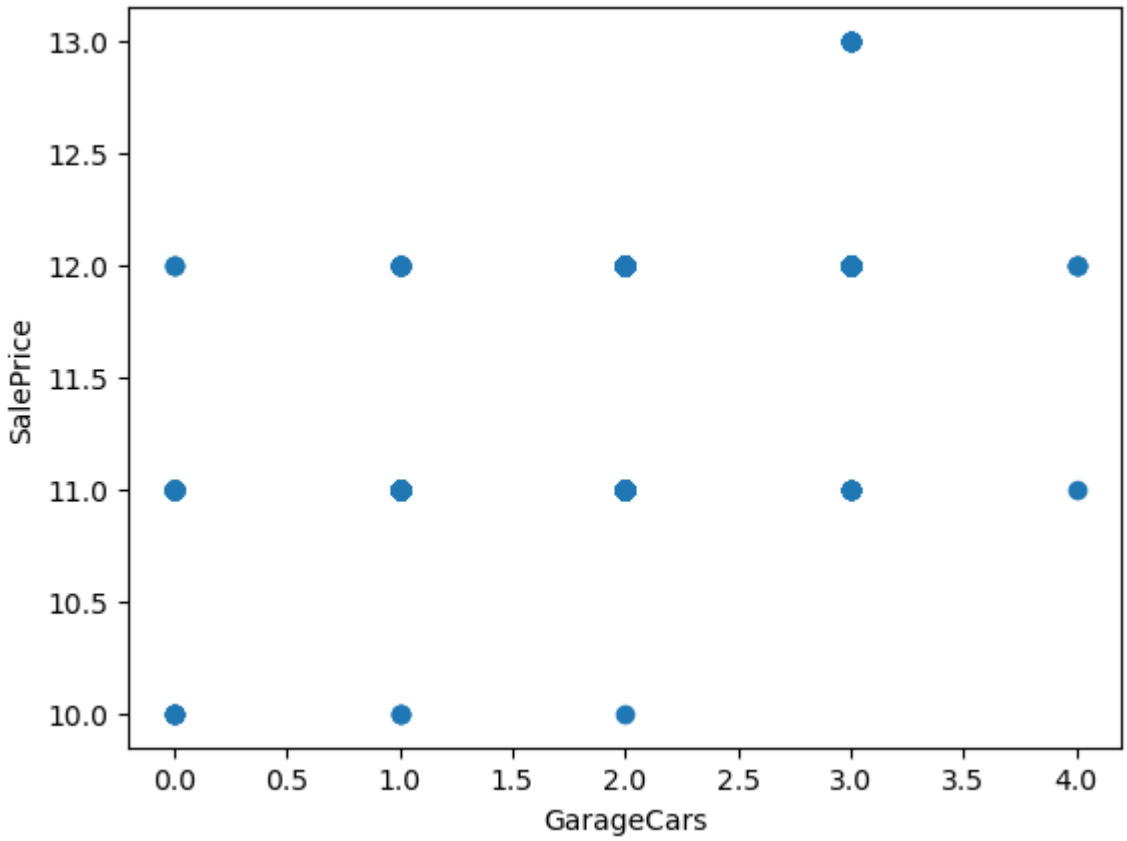
```
Column:MSSubClass is not normally distributed
Column:LotFrontage is not normally distributed
Column:LotArea is not normally distributed
Column:OverallQual is normally distributed
Column:OverallCond is normally distributed
Column:YearBuilt is normally distributed
Column:YearRemodAdd is normally distributed
Column:MasVnrArea is not normally distributed
Column:BsmtFinSF1 is not normally distributed
Column:BsmtFinSF2 is not normally distributed
Column:BsmtUnfSF is normally distributed
Column:TotalBsmtSF is not normally distributed
Column:1stFlrSF is not normally distributed
Column:2ndFlrSF is normally distributed
Column:LowQualFinSF is not normally distributed
Column:GrLivArea is normally distributed
Column:BsmtFullBath is normally distributed
Column:BsmtHalfBath is not normally distributed
Column:FullBath is normally distributed
Column:HalfBath is normally distributed
Column:BedroomAbvGr is normally distributed
Column:KitchenAbvGr is not normally distributed
Column:TotRmsAbvGrd is normally distributed
Column:Fireplaces is normally distributed
Column:GarageYrBlt is normally distributed
Column:GarageCars is normally distributed
Column:GarageArea is normally distributed
Column:WoodDeckSF is not normally distributed
Column:OpenPorchSF is not normally distributed
Column:EnclosedPorch is not normally distributed
Column:3SsnPorch is not normally distributed
Column:ScreenPorch is not normally distributed
Column:PoolArea is not normally distributed
Column:MiscVal is not normally distributed
Column:MoSold is normally distributed
Column:YrSold is normally distributed
Column:SalePrice is normally distributed
```

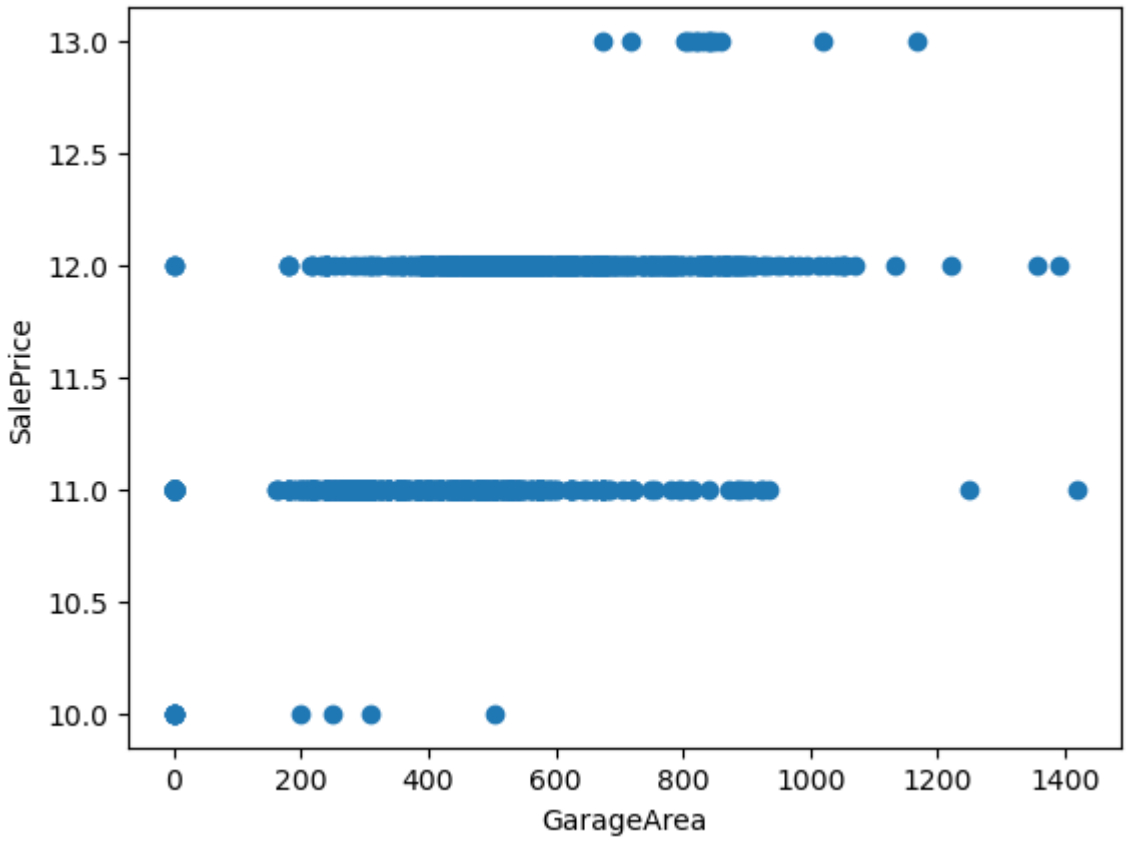
INSIGHTS OF NUMERICAL COLUMNS FROM ABOVE GRAPH: MOST COLUMNS ARE RIGHT SKEWED OR SOME has only constant value columns, some are normaly distributes. MSSubClass is not normally distributed LotFrontage is not normally distributed MasVnrArea is not normally distributed BsmtFinSF1 is not normally distributed BsmtFinSF2 is not normally distributed TotalBsmtSF is not normally distributed 1stFlrSF is not normally distributed LowQualFinSF is not normally distributed BsmtHalfBath is not normally distributed KitchenAbvGr is not normally distributed WoodDeckSF is not normally distributed OpenPorchSF is not normally distributed EnclosedPorch is not normally distributed 3SsnPorch is not normally distributed ScreenPorch is not normally distributed PoolArea is not normally distributed MiscVal is not normally distributed # 17 columns are not normally disriuuted . # but only 9 columns affecting the sale price. we think later .some outlier columns we cant remove because it will be affecting output.

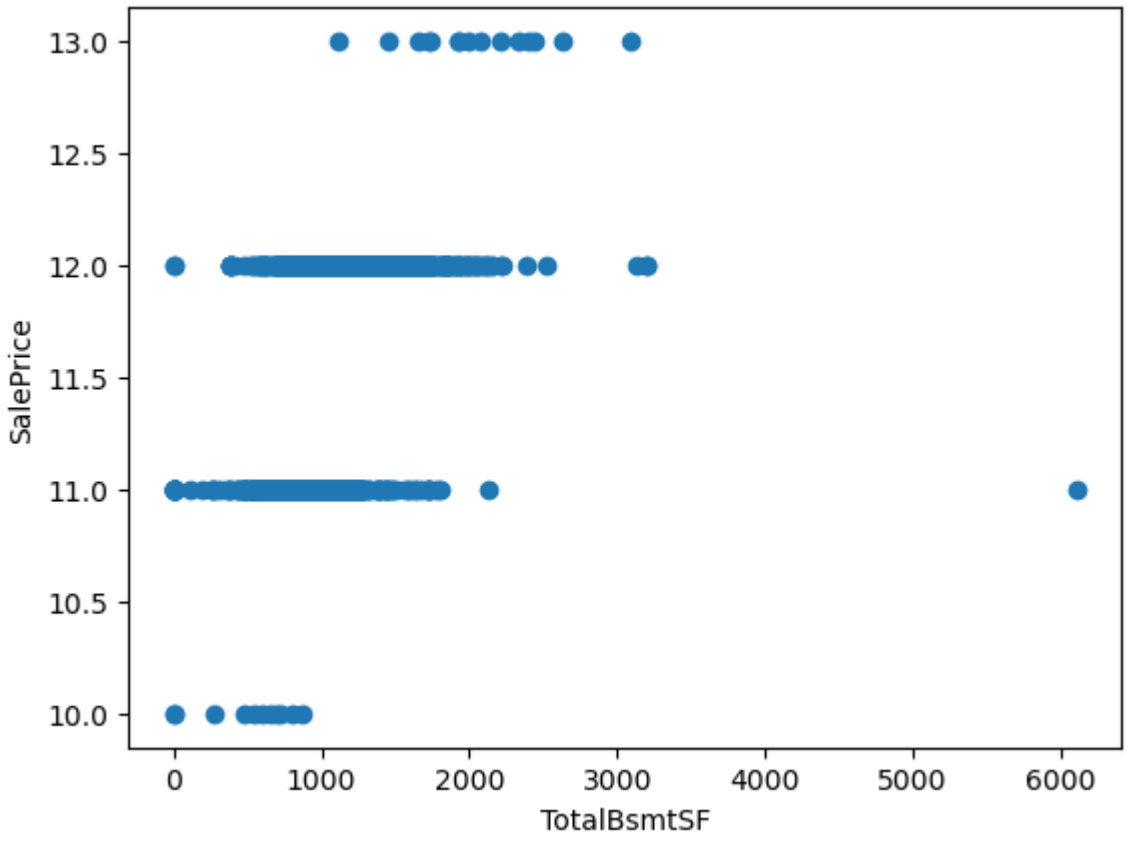
In [33]:
```
# plot outliers columns using scatter plots
# we are going detect outliers in whole dataset

fig = plt.subplots()
plt.scatter(x = df['GrLivArea'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=10)
plt.xlabel('GrLivArea', fontsize=10)
plt.show()

fig1= plt.subplots()
plt.scatter(x = df['OverallQual'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=10)
plt.xlabel('OverallQual', fontsize=10)
plt.show()

fig2= plt.subplots()
plt.scatter(x = df['GarageCars'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=10)
plt.xlabel('GarageCars', fontsize=10)
plt.show()

fig3= plt.subplots()
plt.scatter(x = df['GarageArea'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=10)
plt.xlabel('GarageArea', fontsize=10)
plt.show()

fig4= plt.subplots()
plt.scatter(x = df['TotalBsmtSF'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=10)
plt.xlabel('TotalBsmtSF', fontsize=10)
plt.show()

fig5= plt.subplots()
plt.scatter(x = df['1stFlrSF'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=10)
plt.xlabel('1stFlrSF', fontsize=10)
plt.show()

fig6= plt.subplots()
plt.scatter(x = df['FullBath'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=11)
plt.xlabel('FullBath', fontsize=11)
plt.show()

fig7= plt.subplots()
plt.scatter(x = df['TotRmsAbvGrd'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=11)
plt.xlabel('TotRmsAbvGrd', fontsize=11)
plt.show()

fig8= plt.subplots()
plt.scatter(x = df['YearBuilt'], y = df['SalePrice'])
plt.ylabel('SalePrice', fontsize=11)
plt.xlabel('YearBuilt', fontsize=11)
plt.show()
```

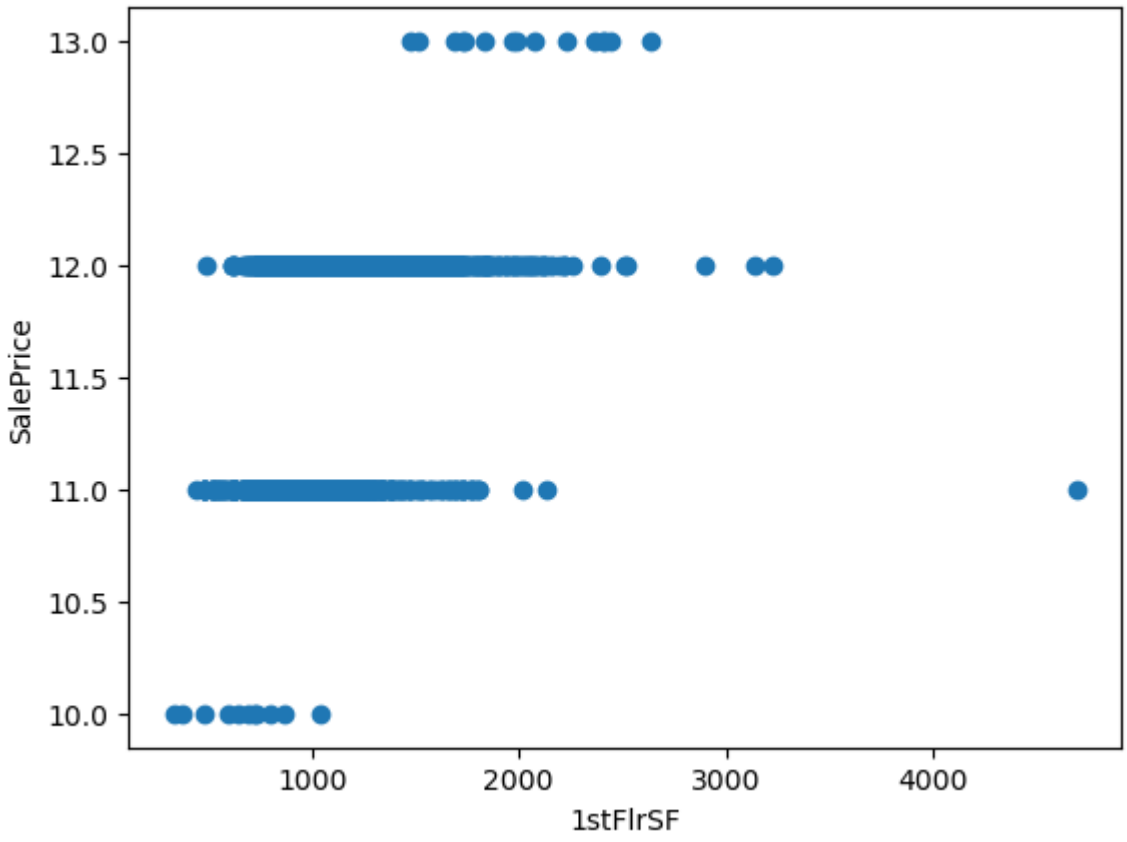

13.0
12.5
12.0
11.5
11.0
10.5
10.0
SalePrice
6.0
6.5
7.0
7.5
8.0
8.5
GrLivArea

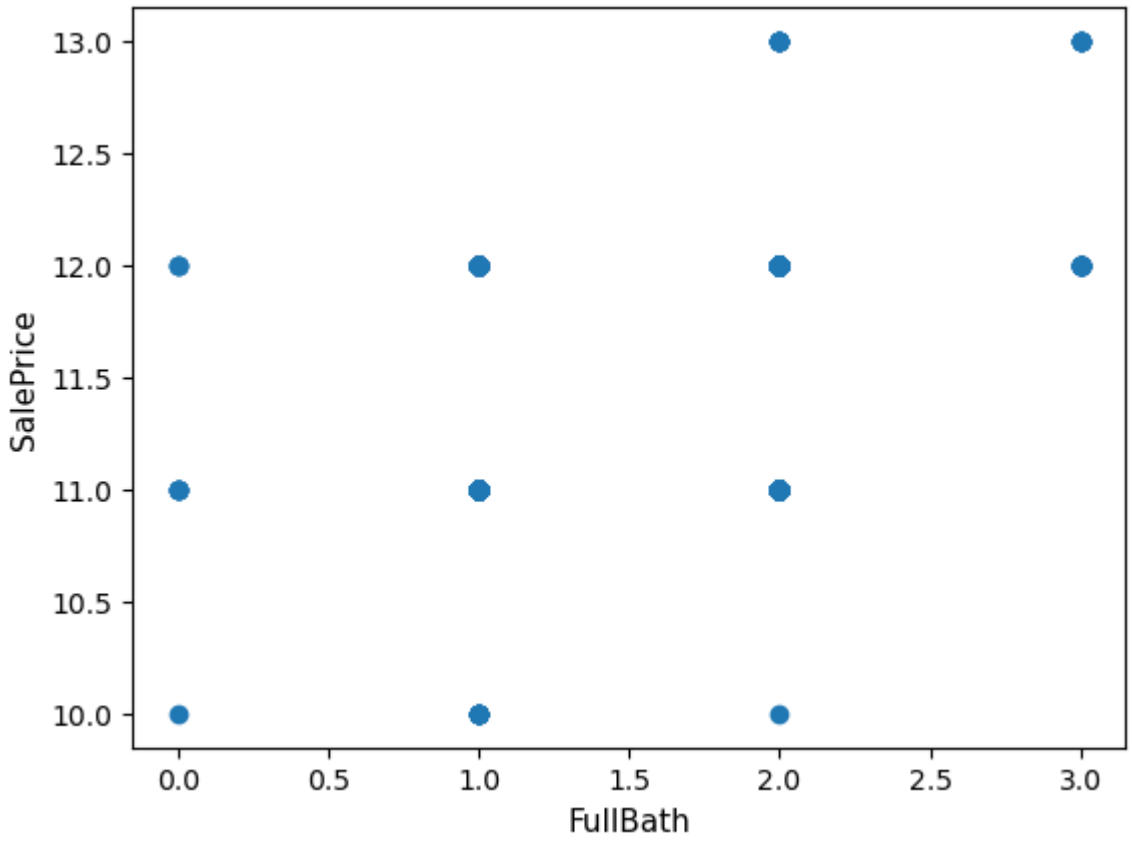

13.0
12.5
12.0
11.5
11.0
10.5
10.0
SalePrice
2
4
6
8
10
OverallQual

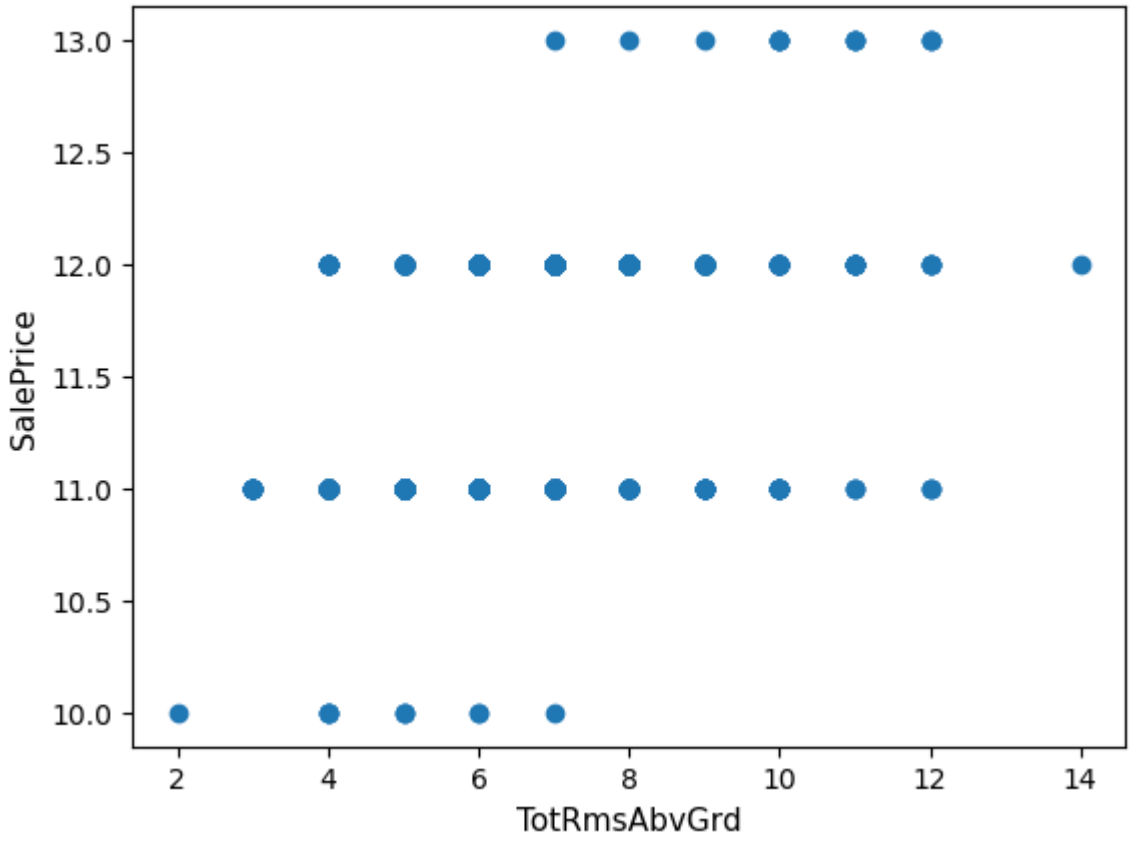

SalePrice
GarageCars
13.0
12.5
12.0
11.5
11.0
10.5
10.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

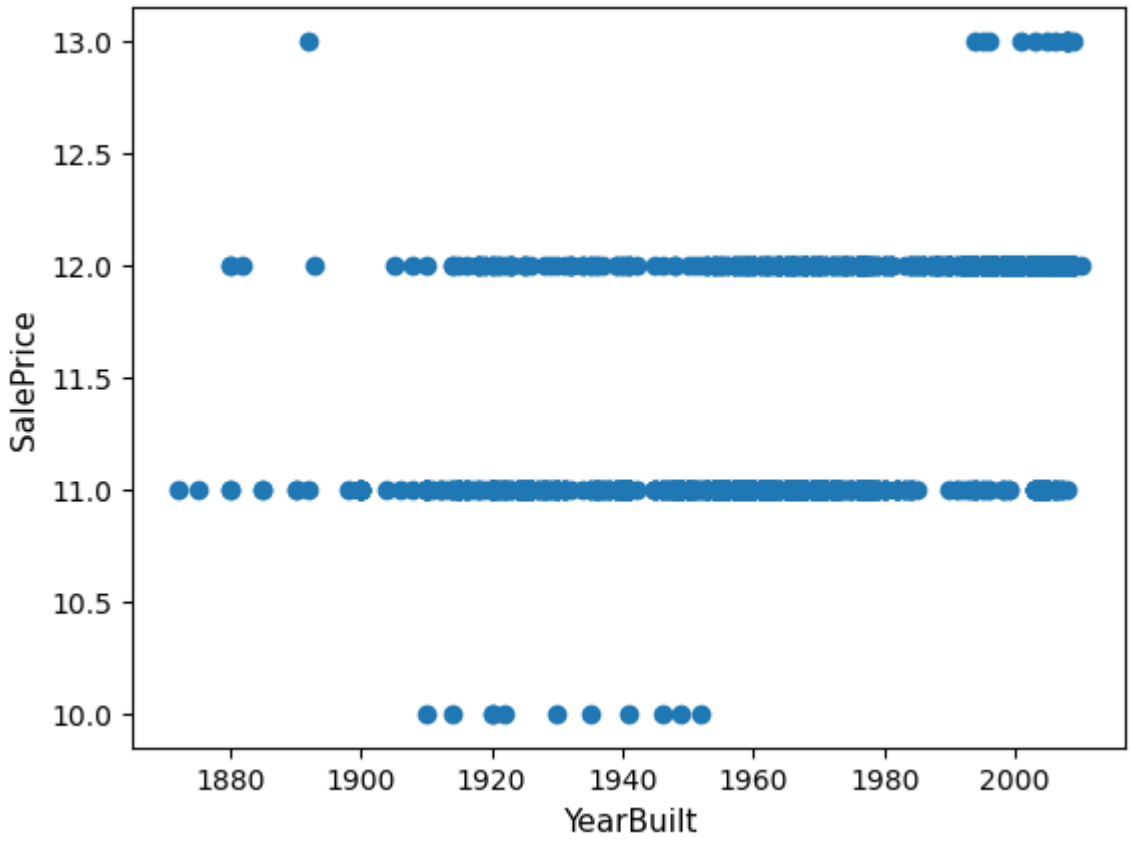

SalePrice
GarageArea
13.0
12.5
12.0
11.5
11.0
10.5
10.0
0
200
400
600
800
1000
1200
1400


13.0
12.5
12.0
11.5
11.0
10.5
10.0
SalePrice
0
1000
2000
3000
4000
5000
6000
TotalBsmtSF


13.0
12.5
12.0
11.5
11.0
10.5
10.0
SalePrice
1000
2000
3000
4000
1stFlrSF


13.0
12.5
12.0
11.5
11.0
10.5
10.0
SalePrice
0.0
0.5
1.0
1.5
2.0
2.5
3.0
FullBath

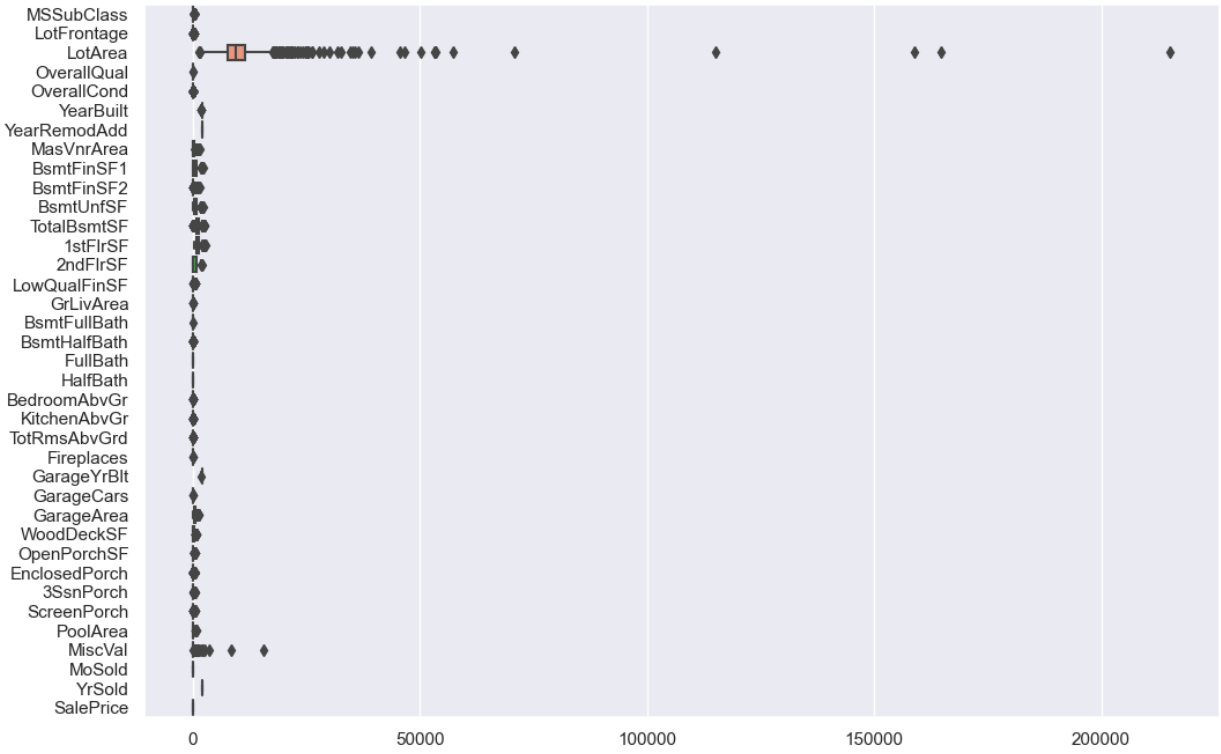

13.0
12.5
12.0
11.5
11.0
10.5
10.0
SalePrice
2
4
6
8
10
12
14
TotRmsAbvGrd



In [39]:
```
# df[['1stFlrSF','SalePrice']].describe()
```

In [34]:
```
# above all highly correlated  feature with output so i check thier outliers .
# we drop some outliers from specfic range
df = df.drop(df[(df['GrLivArea']>8.6) & (df['SalePrice']<13.5)].index)
df = df.drop(df[(df['GarageArea']>1300) & (df['SalePrice']<13.5)].index)
df = df.drop(df[(df['TotalBsmtSF']>3000) & (df['SalePrice']<13.5)].index)
df = df.drop(df[(df['1stFlrSF']>3000) & (df['1stFlrSF']<13.5)].index)
#you will get an idea of which points are outliers. and below code is used to delete c
```

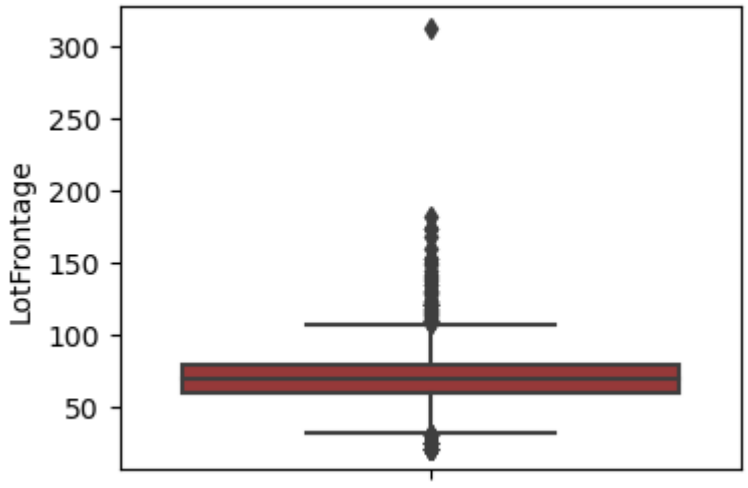
In [55]:
```
#check outliers
plt.figure(figsize=(12, 8))
sns.set_theme(style="darkgrid")
sns.set_palette("GnBu")
ax = sns.boxplot(data=df, orient="h")
plt.show()
```



In [37]:
```
df.LotFrontage.describe()
```

Out[37]:
```
count    1453.000000
mean       69.602202
std        21.041185
min        21.000000
25%        60.000000
50%        69.000000
75%        79.000000
max       313.000000
Name: LotFrontage, dtype: float64
```

In [38]:
```
# box plot of numeric column only
plt.figure(figsize=(4,3 ))

sns.boxplot(y=df['LotFrontage'],orient="h",color="brown")

plt.show()
```



In [35]:
```
# Check skenwess then handle the outliers

print('skewness value of : ',df['LotFrontage'].skew()) # no skew found
print('skewness value of : ',df['LotArea'].skew())
# skew value between -1 to 1 so i rpleace with median value
# LotFrontage area increse sale prize also increase so need to remove LotFrontage outl
```

```
skewness value of :  1.729525073719968
skewness value of :  12.620650803812744
```

decided to replace LotFrontage house - impute value with median only

In [36]:
```
# using detection percentile choose 10 % and 90%
print(df['LotFrontage'].quantile(0.10)) # 1
print(df['LotFrontage'].quantile(0.90))


# we will do the flooring (e.g., the 10th percentile) for the lower values and
'''capping (e.g., the 90th percentile) for the higher values '''
```

```
49.0
92.0
```

Out[36]:
```
'capping (e.g., the 90th percentile) for the higher values '
```

In [37]:
```
print(df['LotArea'].quantile(0.10)) # 2
print(df['LotArea'].quantile(0.90))
```

```
5000.0
14294.399999999998
```

In [38]:
```
# remove outliers using where df["LotFrontage"]
df["LotFrontage"] = np.where(df["LotFrontage"] <49.0, 49.0,df['LotFrontage'])
df["LotFrontage"] = np.where(df["LotFrontage"] >91.0, 91.0,df['LotFrontage'])
print(df['LotFrontage'].skew()) # normal distribution
# it removes some outliers
```

```
0.1229750086400216
```

In [39]:
```
#remove outliers using where : df ["LotArea"]
df["LotArea"] = np.where(df["LotArea"] <5000.0, 5000.0,df['LotArea'])
df["LotArea"] = np.where(df["LotArea"] >14226.400000000001, 14226.400000000001,df['Lot
print(df['LotArea'].skew())
# it removes some outliers
```

```
0.06904789634513771
```

3. EDA - DATA VISUALIZATION

- I.UNIVARIATE ANALYSIS -Categorical variables can be visualized using a Count plot, Bar Chart, Pie Plot, etc.
- Numerical Variables can be visualized using Histogram, Box Plot, Density Plot,
- II. BIVARIATE ANALYSIS:
- Bivariate Analysis helps to understand how variables are related to each other and the relationship between dependent and independent variables present in the dataset. -For Numerical variables, Pair plots and Scatter plots are widely been used to do Bivariate Analysis.

-A Stacked bar chart can be used for categorical variables if the output variable is a classifier. Bar plots can be used if the output variable is continuous

- III. Multivariate Analysis -A heat map is widely been used for Multivariate Analysis

-Heat Map gives the correlation between the variables, whether it has a positive or negative correlation.

- Univariate analysis can be done for both Categorical and Numerical variables.

In [40]:
```
## identifying numeric variables
numeric = df.select_dtypes(include=['float64','int64'])
numeric = numeric.columns
```

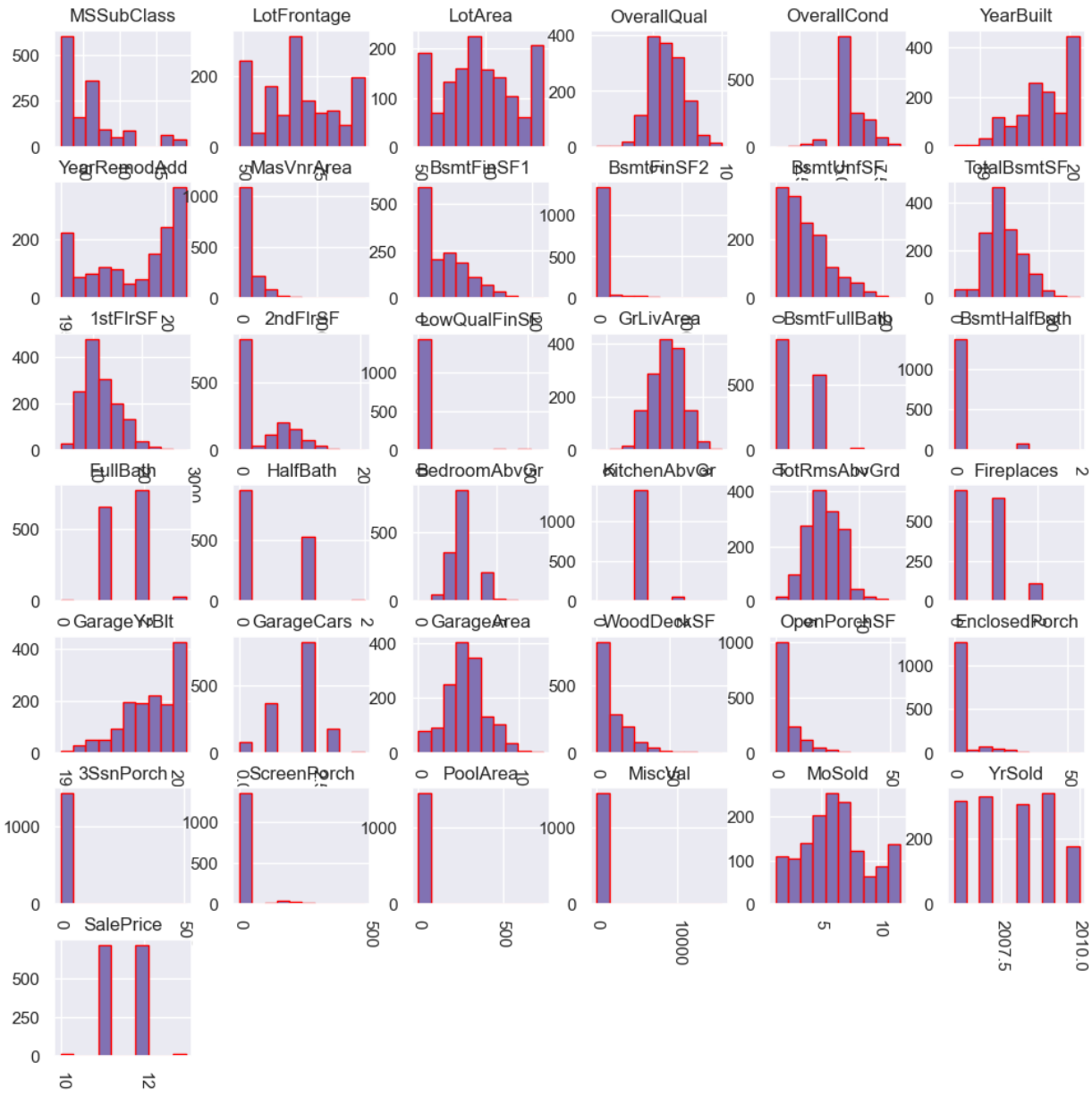
In [41]:
```
print(numeric, len(numeric)) # allready i checked distribution of numerical data with
'''please go and check histogram plots of numerical columns only. '''
```

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object') 37
```

Out[41]:
```
'please go and check histogram plots of numerical columns only. '
```

- DISTRIBUTIUTION OF NUMERICAL FEATURES : Numerical columns plot:

In [66]:
```
# DISTRIBUTIUTION OF NUMERICAL FEATURES : Numerical columns plot:
df.hist(figsize=(12,12), xrot=-90, bins=10, color = "m",ec="red") ## Display the label
plt.show()
```



Plot categorical columns :using countplot(),piechart()

In [42]:
```
# plot categorical columns
##identifying categorical variables
categorical = df.select_dtypes(include=['object'])
categorical = categorical.columns
```

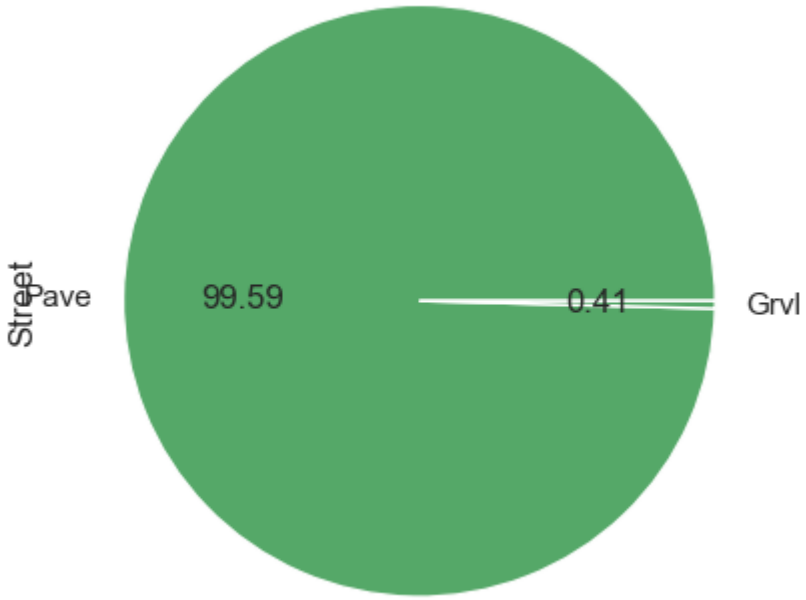
In [43]:
```
print("categorical columns :", categorical,"length is ",len(categorical))
```

```
categorical columns : Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilit
ies',
       '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', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
       'PavedDrive', 'SaleType', 'SaleCondition'],
      dtype='object') length is  38
```

- 1.pie chart : for categorical columns:

In [73]:
```
# pie chart for categorical variable .ieStreet
#colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99']
df["Street"].value_counts().plot(kind='pie',autopct='%.2f',colors='g')
```

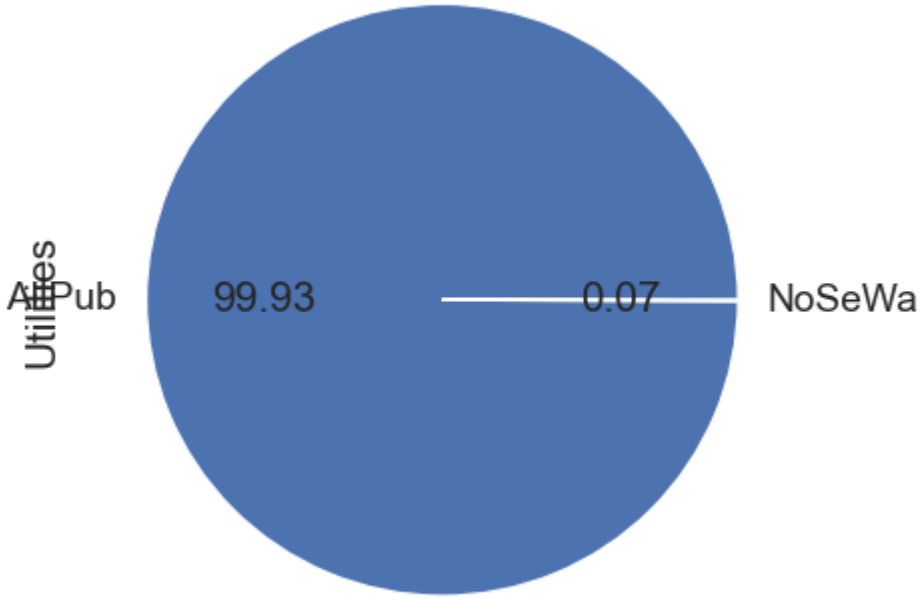
Out[73]:
```
<AxesSubplot:ylabel='Street'>
```



- street column show only one pave constant value so we drop in future.

In [94]:
```
df["Utilities"].value_counts().plot(kind='pie',autopct='%.2f')
```

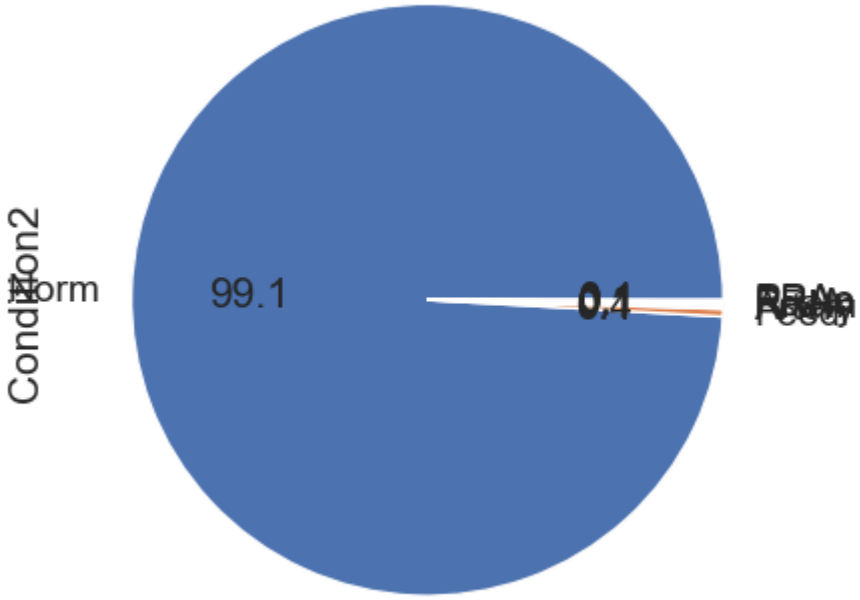
Out[94]:
```
<AxesSubplot:ylabel='Utilities'>
```



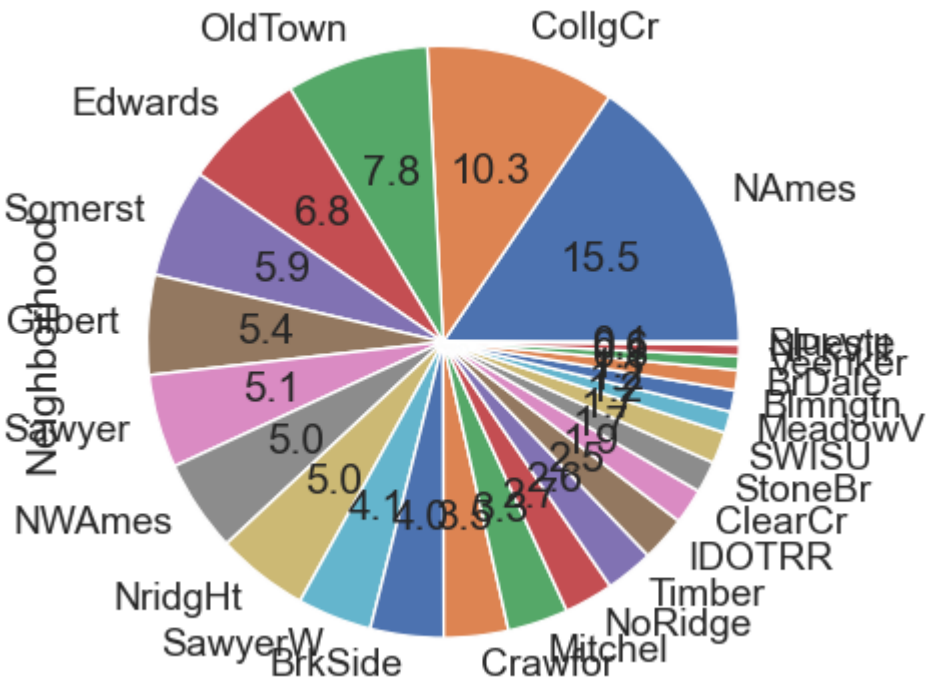
- same as street column show only Utilities - one ALL PUB - constant value so we can drop in future.

In [101…

```
# Condition1
df["Condition2"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
# same as above two columns we can drop in future - Condition2
```

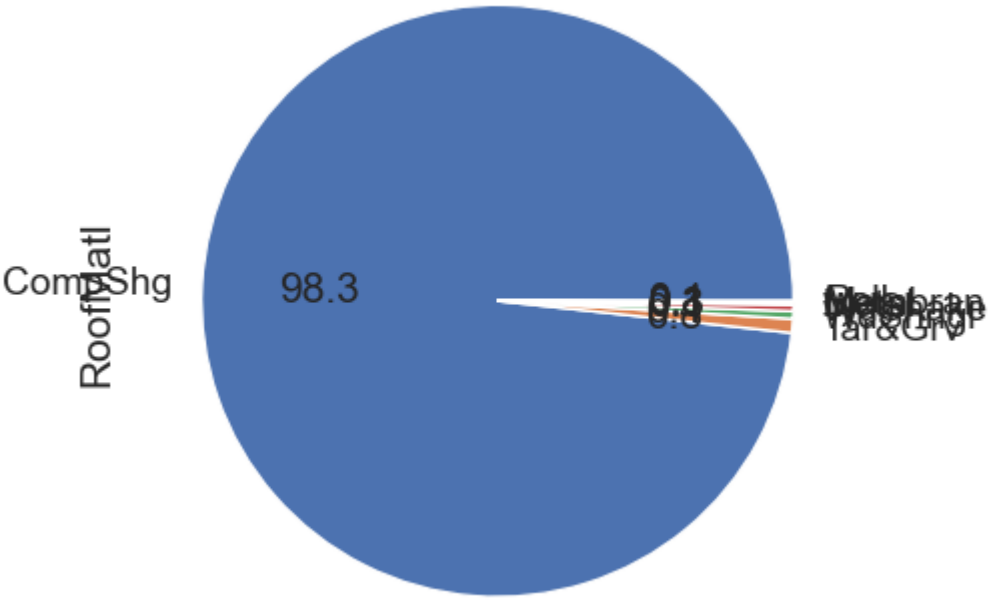


In [103…
```
# Neighborhood
df["Neighborhood"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
```



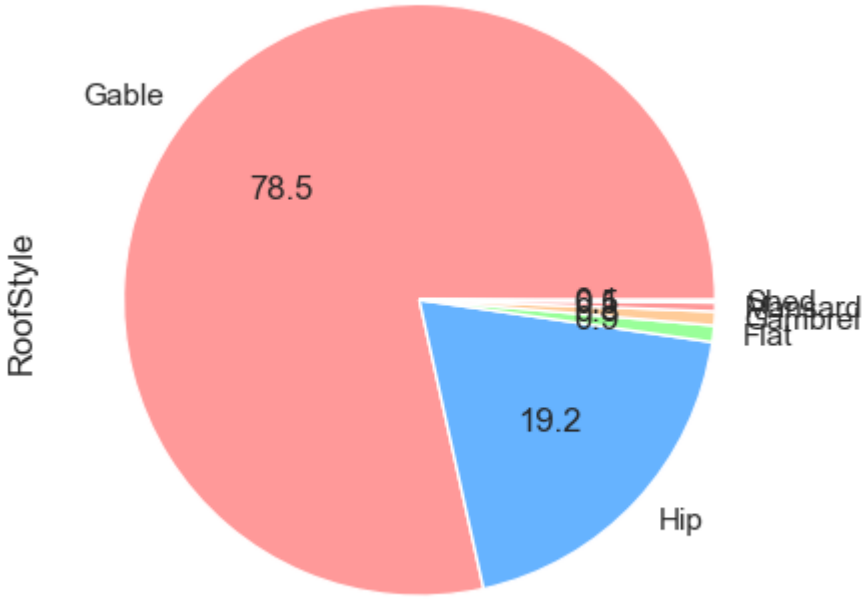
Neighborhood column - NORTH AMERICAN side more houses ,means crowded area, then Collgcr then Oldtown then so on .it is important column .According to diiferent regions have different sale price.

In [106…
```
# RoofMatl
df["RoofMatl"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
```

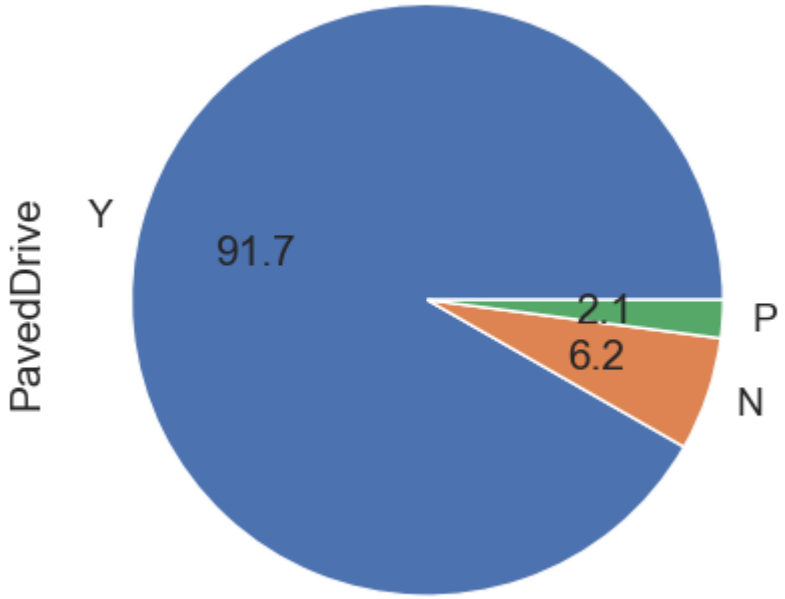


In [ ]:
```
#RoofMatl -insight one material constant value we can drop .it is not affecting the sa
```

In [69]:
```
# RoofStyle: Gabel type roofstyling most houses are there.
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99']
df["RoofStyle"].value_counts().plot(kind='pie',autopct='%.1f',colors=colors)
plt.show()
```

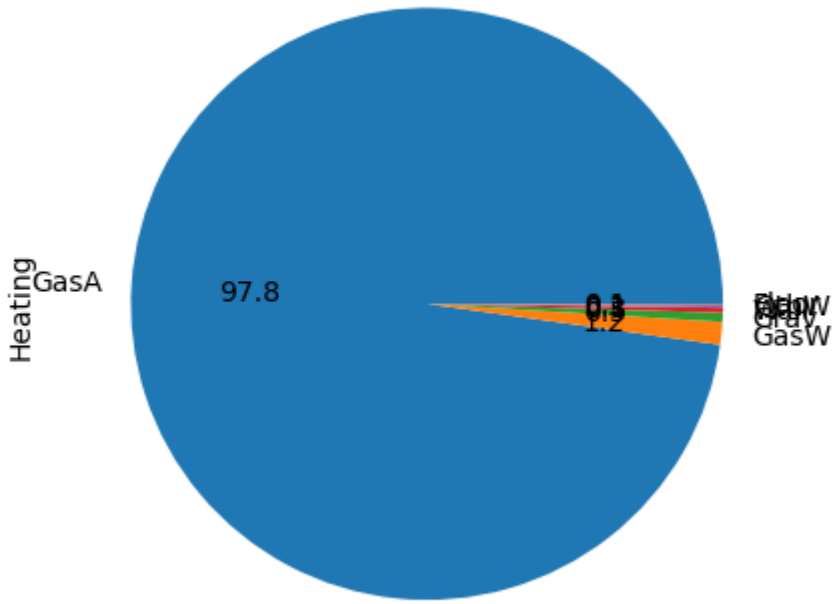


In [109…
```
#PavedDrive
df["PavedDrive"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
```



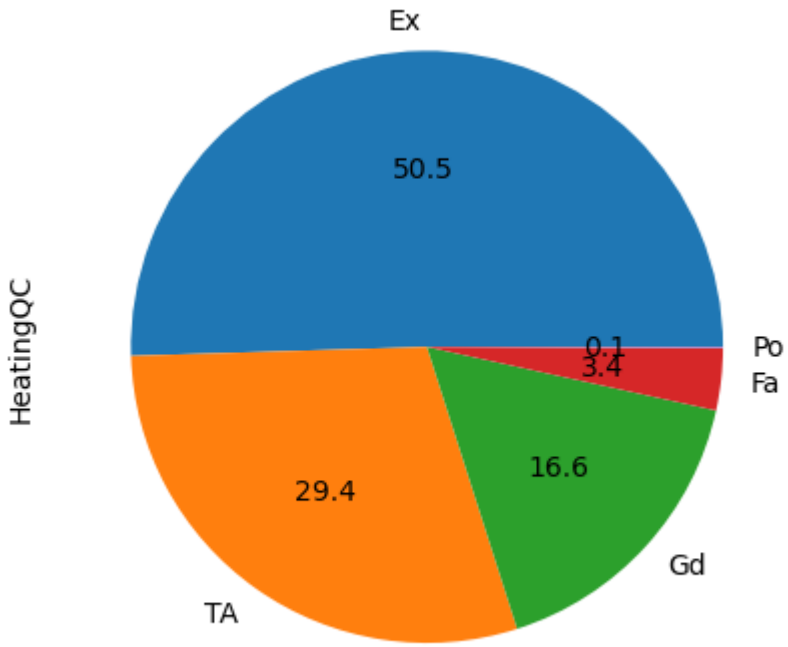
- PavedDrive -insight is most houses are pavedrive area 91.7% it important feature when buy house.

In [53]:
```
df["Heating"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
```



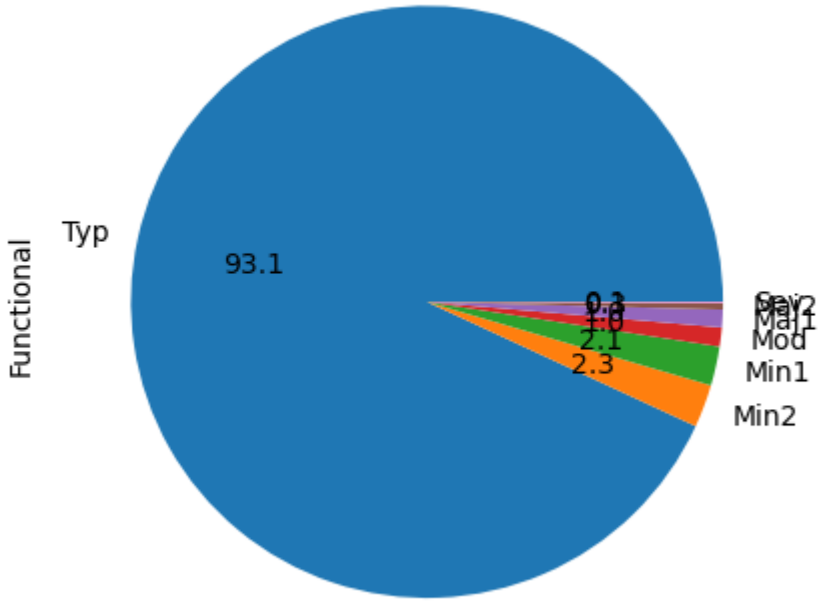
- Heating - In all houses heating system are provided almost 97.8 %

In [54]:
```
df["HeatingQC"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
```



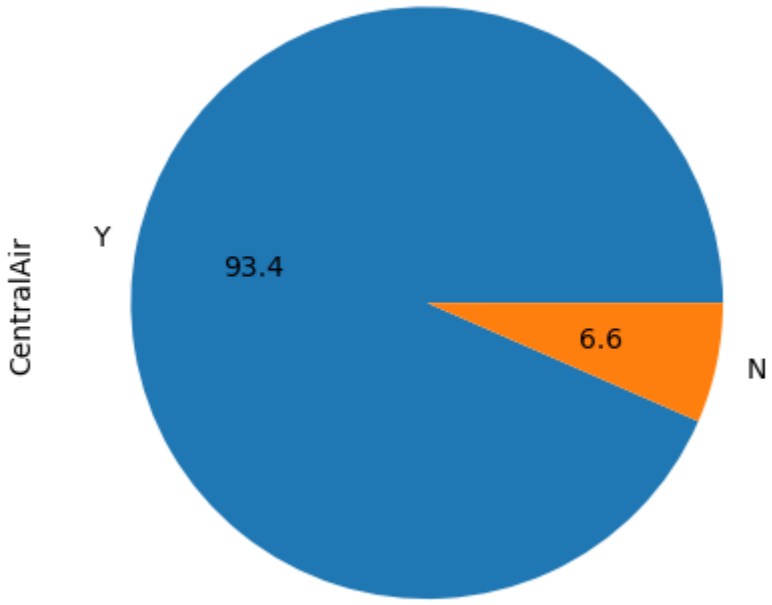
- HeatingQC and heating both variables showing heating service ,so we can drop Heating column because already heating quality is -provided by our dataset so in future we drop heating column

In [55]:
```
#Functional-
df["Functional"].value_counts().plot(kind='pie',autopct='%.1f')
plt.show()
```



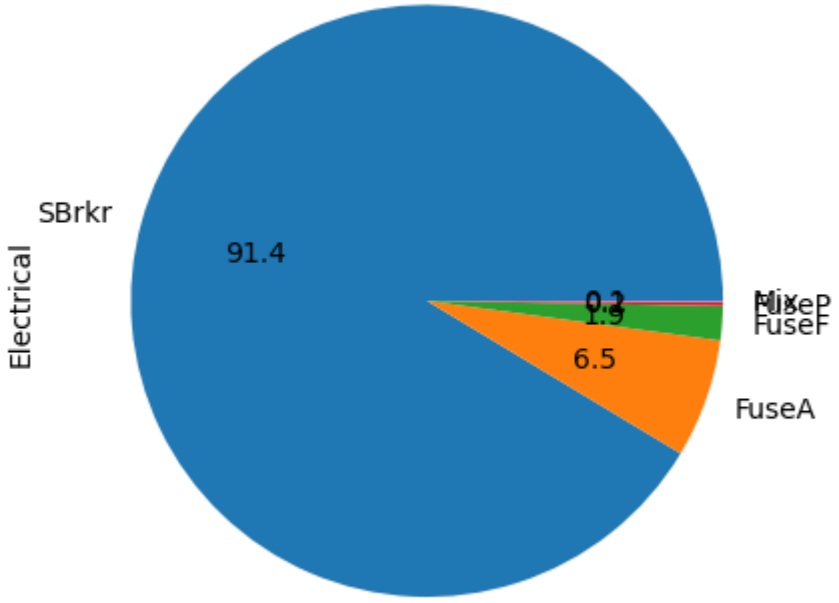
- Functional- Home functionality rating is type means typical upto 93.1% so have to take this column and drop other related columns such as Electrical

In [56]:
```
# CentralAir', 'Electrical', 'KitchenQual',
df["CentralAir"].value_counts().plot(kind='pie',autopct='%.1f')

plt.show()
```



In [57]:
```
df["Electrical"].value_counts().plot(kind='pie',autopct='%.1f')
```

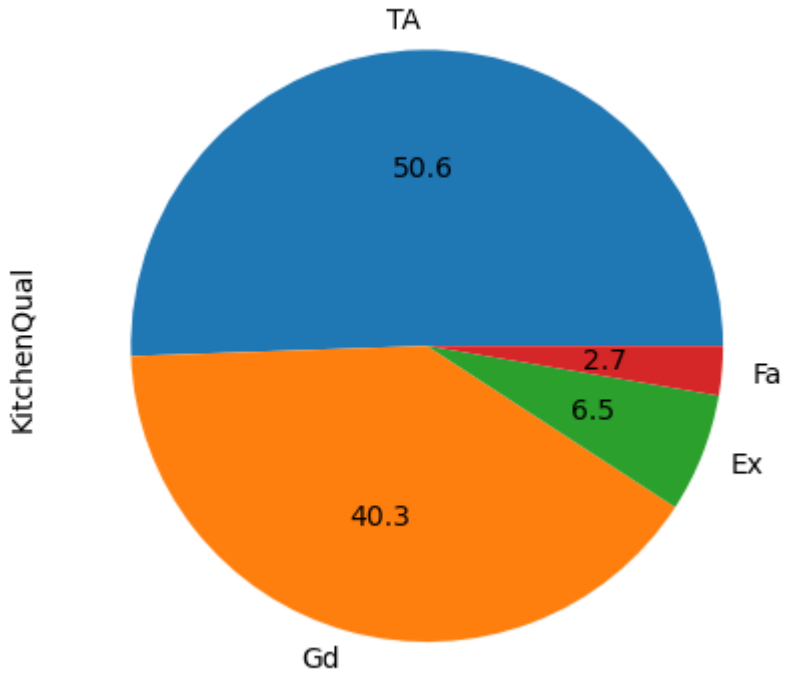
Out[57]: <AxesSubplot:ylabel='Electrical'>



-CentralAir -almost all houses have CentralAir systyem -93.4 % -df["Electrical"] -almost all houses provided SBrkr electrical system 91.4%

In [58]:
```
# KitchenQual-
df["KitchenQual"].value_counts().plot(kind='pie',autopct='%.1f')
```

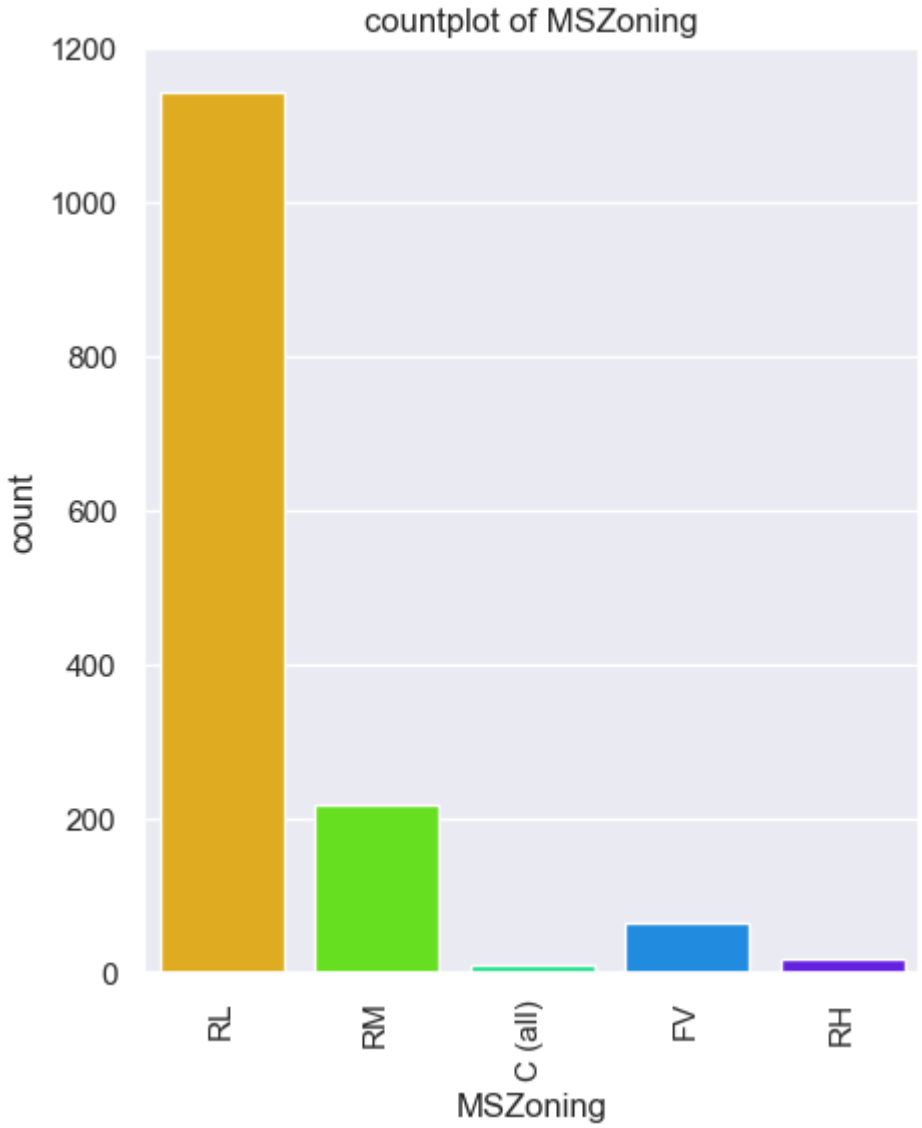
Out[58]: <AxesSubplot:ylabel='KitchenQual'>

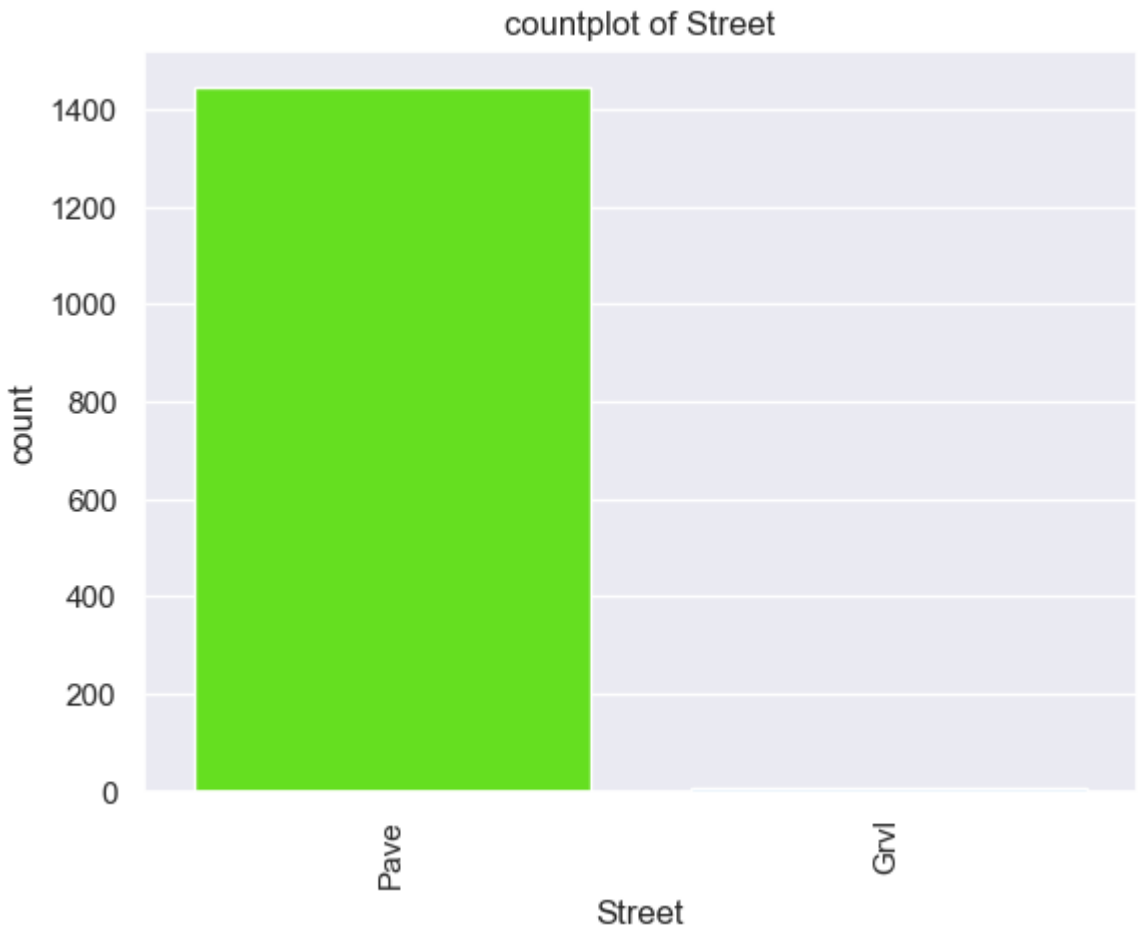


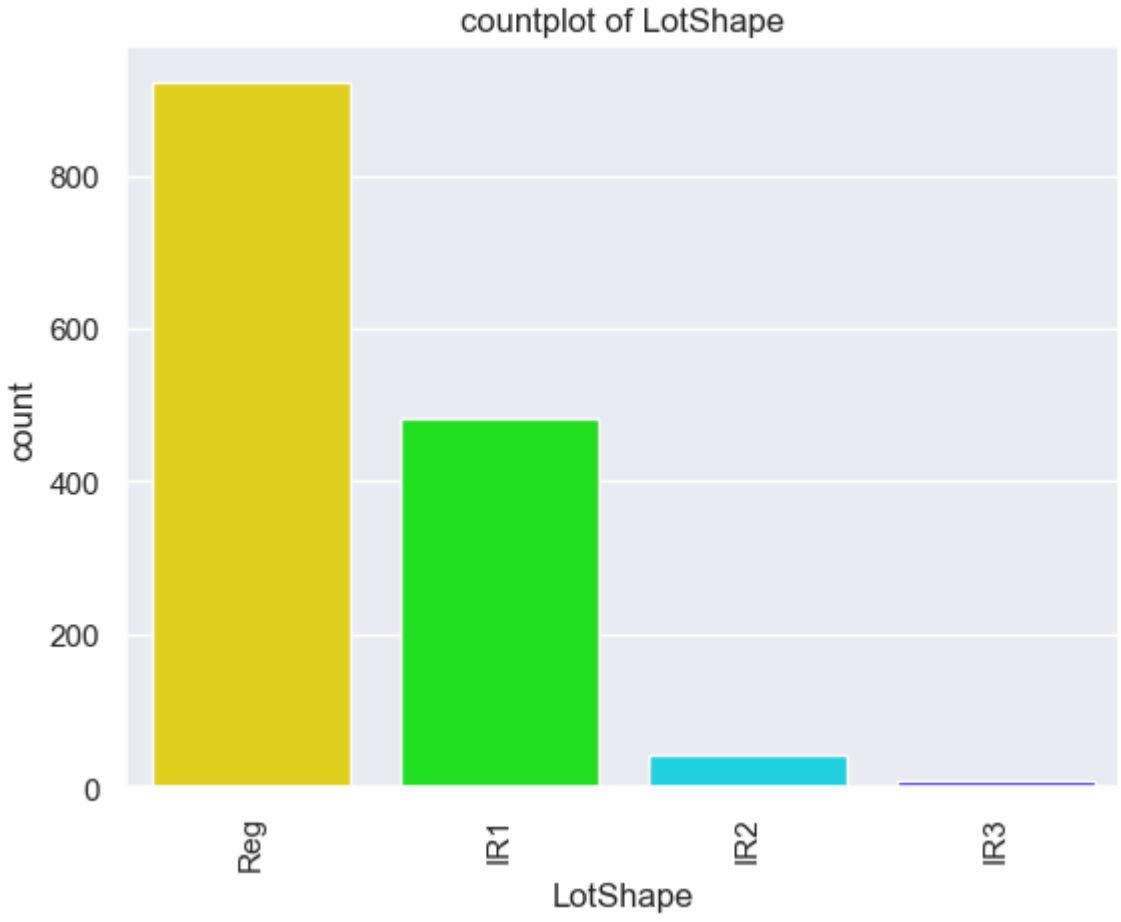
- Kitchen Quality is another important feature to predict house price. There is a very big difference in price between houses with different kitchen quality

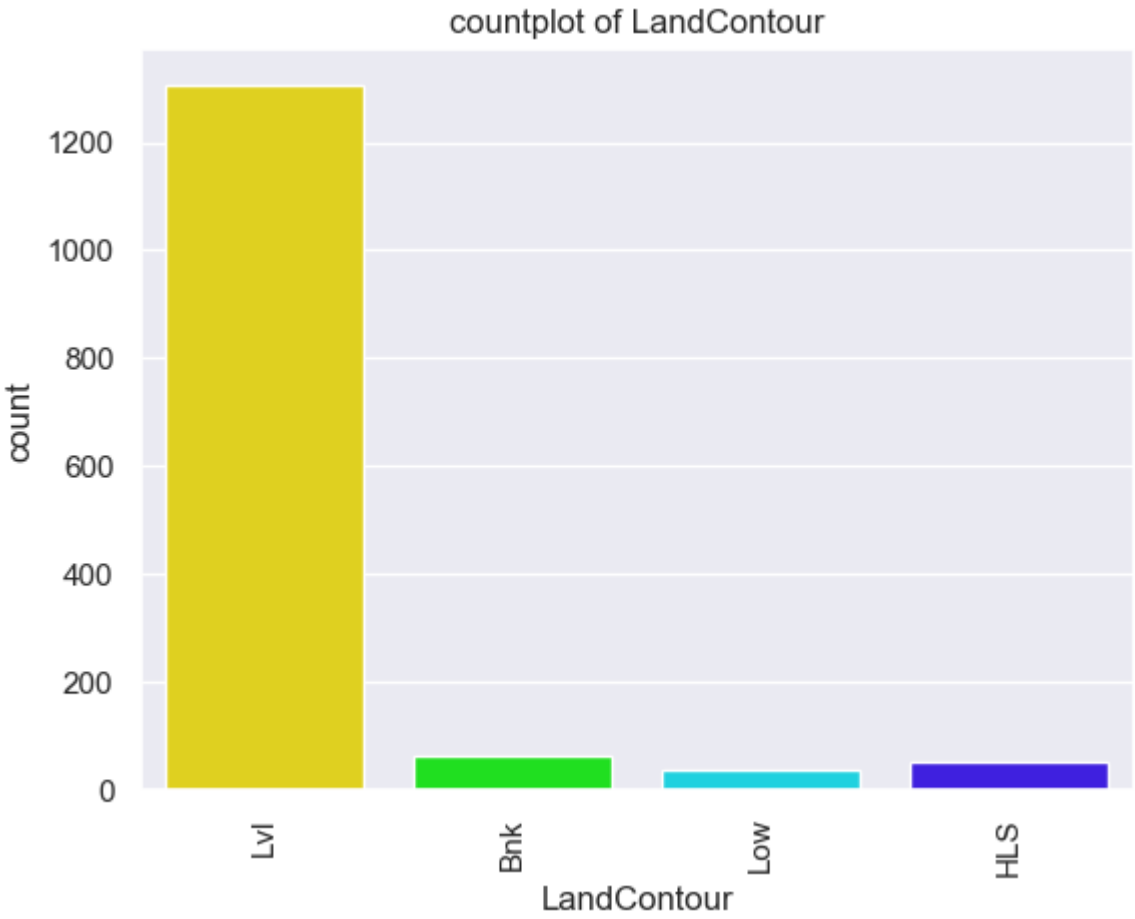
Plot bar /countplot - categorical columns -

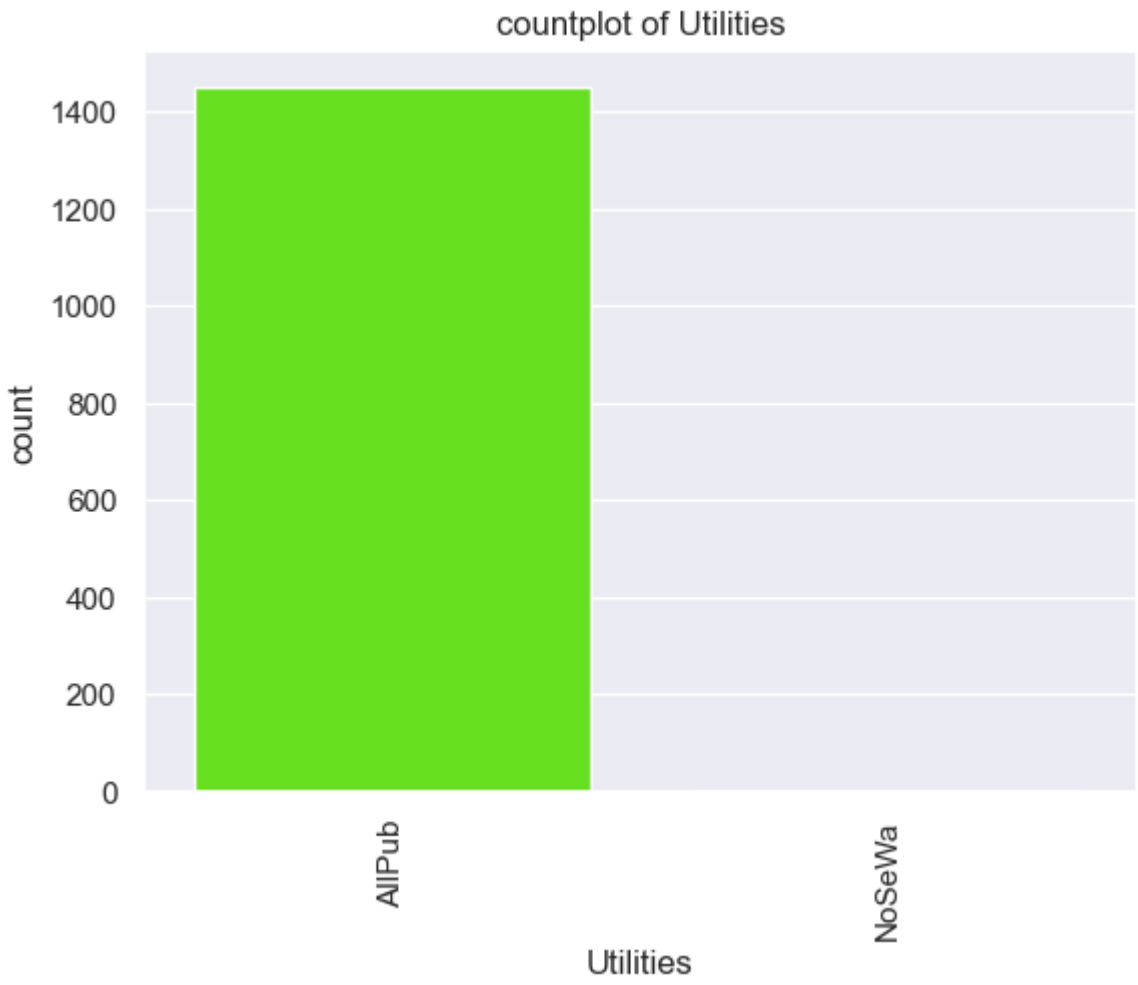
In [81]:
```
# using for loop you can plot countplot : of all categorical columns :
fig = plt.figure(figsize=(5,6))

for col in categorical:

    sns.countplot(x=col, data=df, palette = "gist_rainbow")
    plt.xticks(rotation=90)
    plt.title(f"countplot of {col}")

    plt.show()
fig.tight_layout(pad=0.2)
```

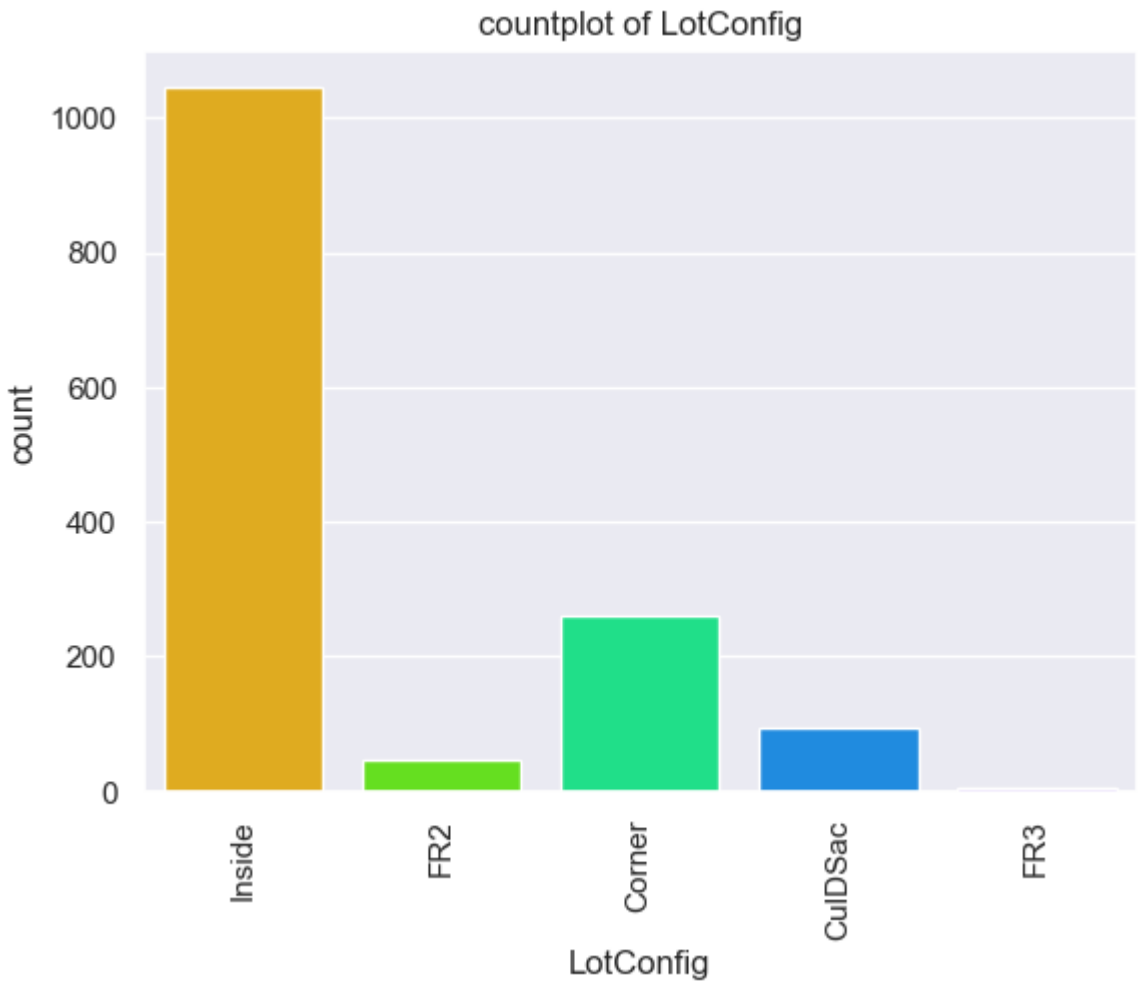


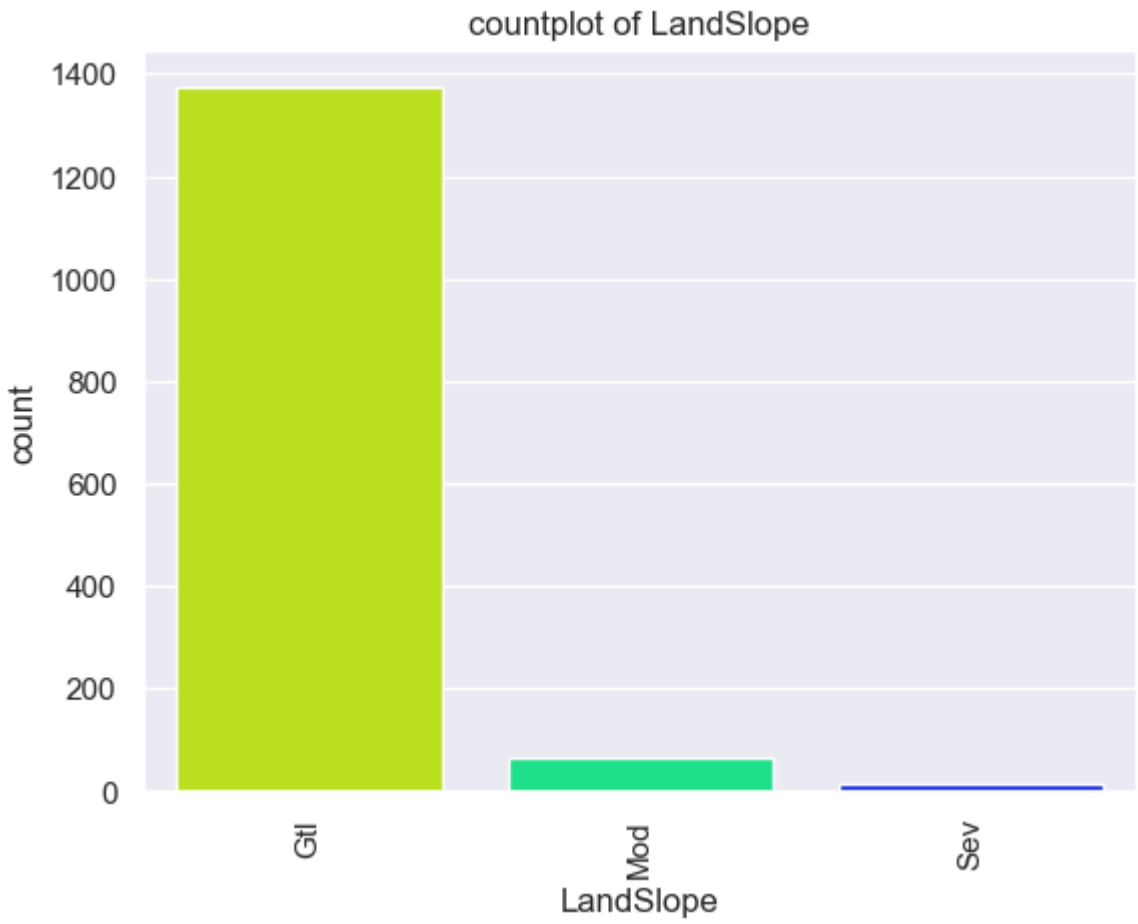

countplot of Street
1400
1200
1000
800
600
400
200
0
count
Pave
Grvl
Street

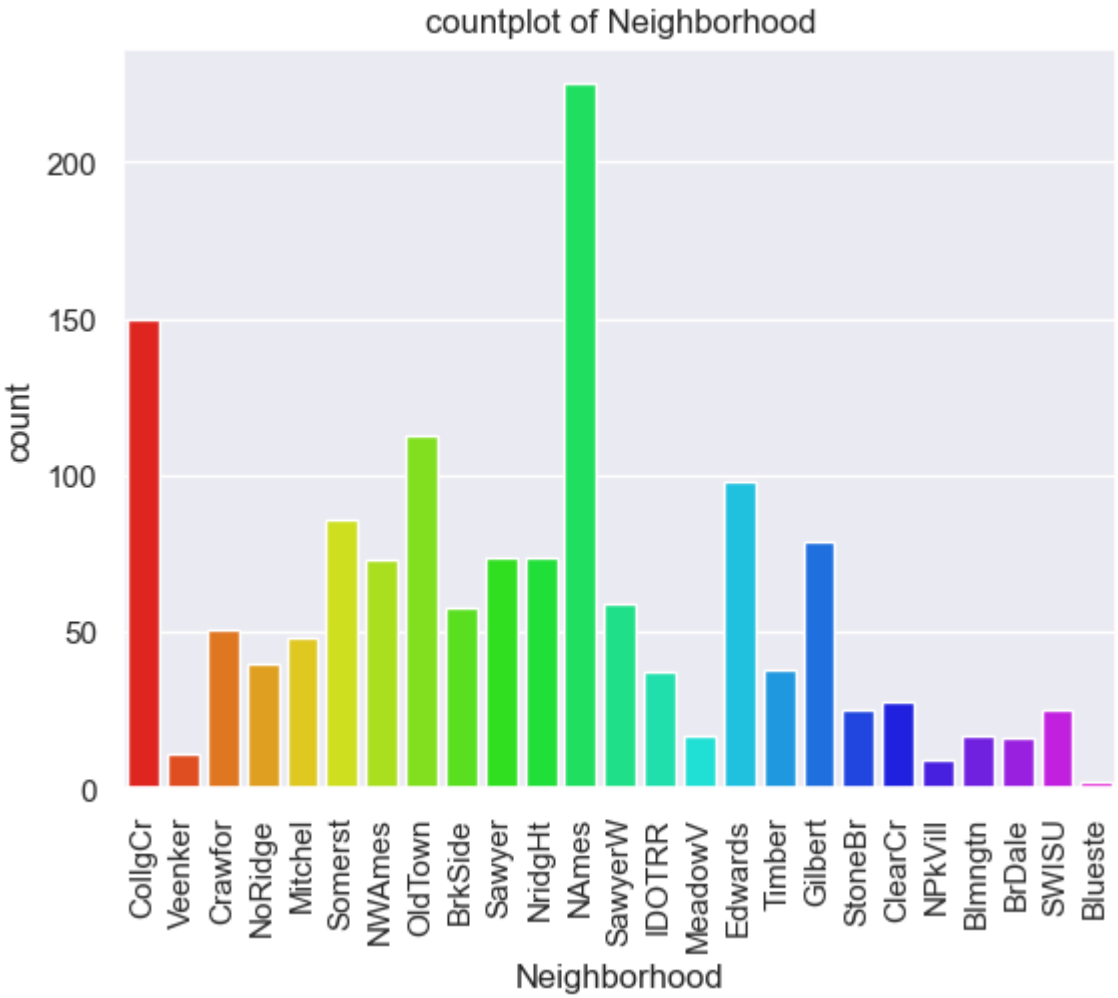

countplot of LotShape
800
600
400
200
0
count
Reg
IR1
IR2
IR3
LotShape

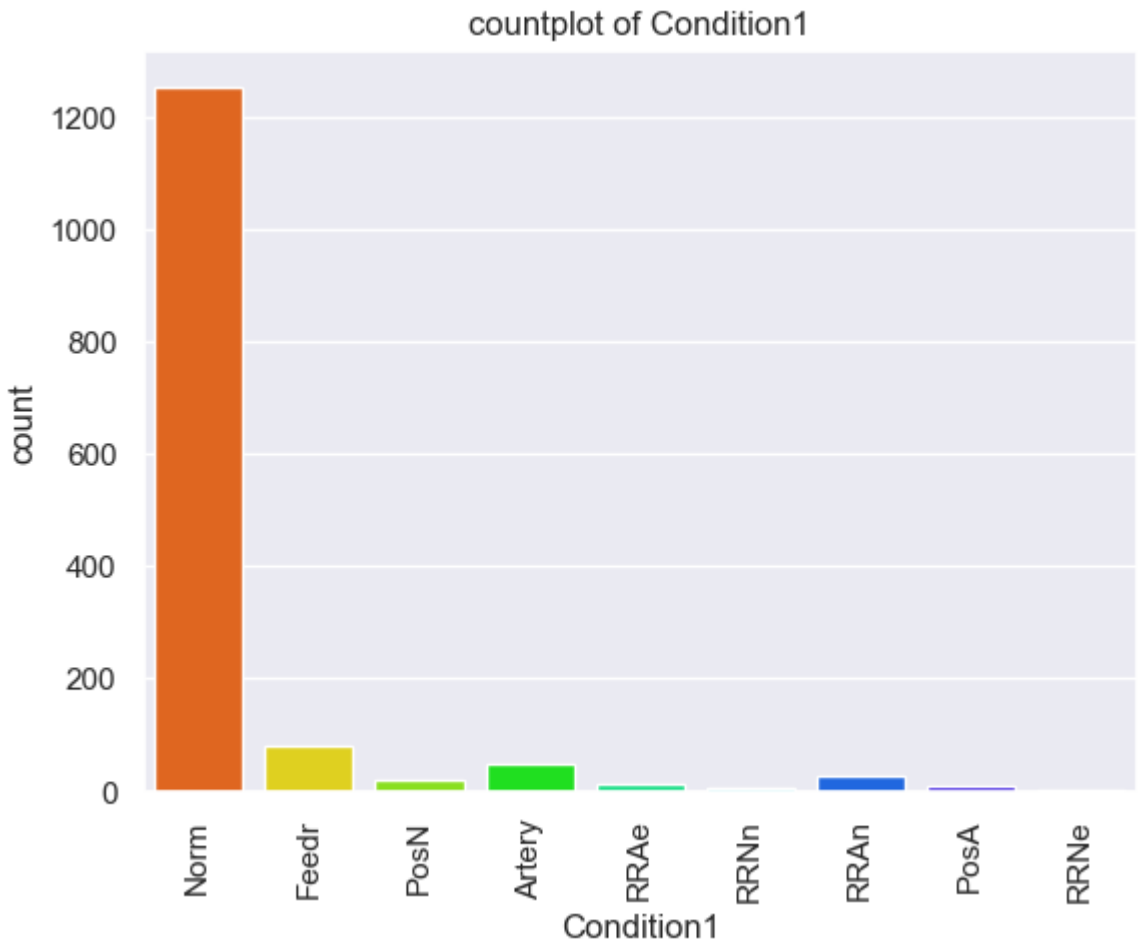

countplot of LandContour
count
1200
1000
800
600
400
200
0
Lvl
Bnk
Low
HLS
LandContour

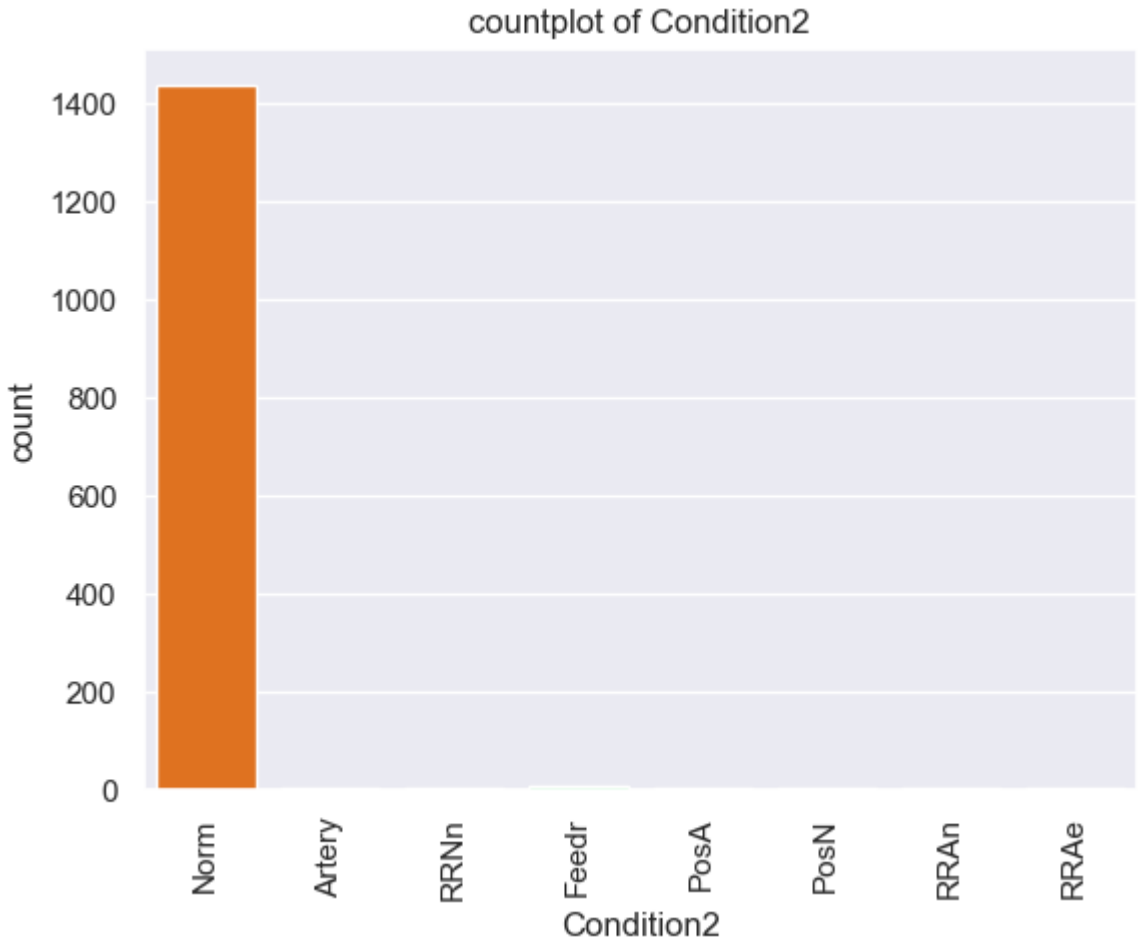

countplot of Utilities
count
1400
1200
1000
800
600
400
200
0
AllPub
NoSeWa
Utilities

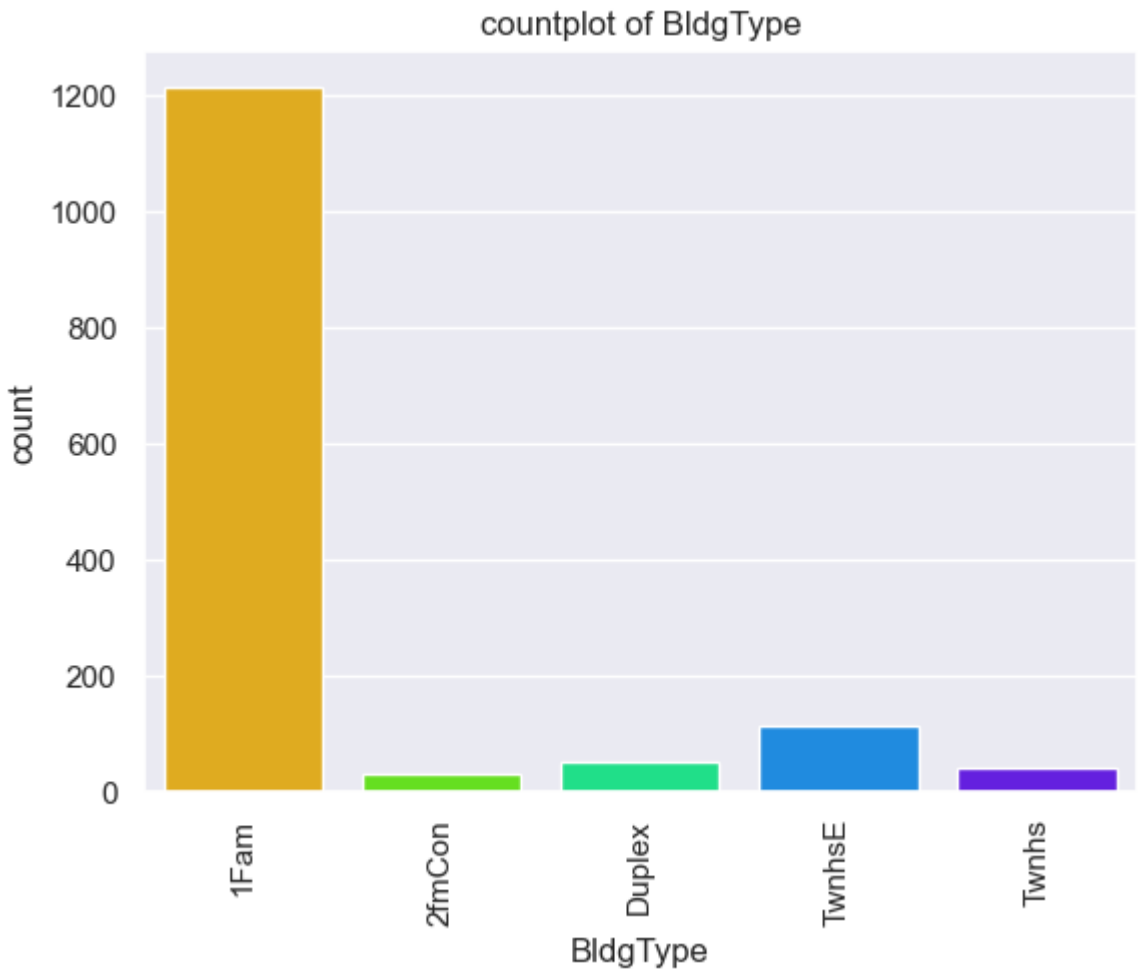

countplot of LotConfig
count
1000
800
600
400
200
0
Inside
FR2
Corner
CulDSac
FR3
LotConfig

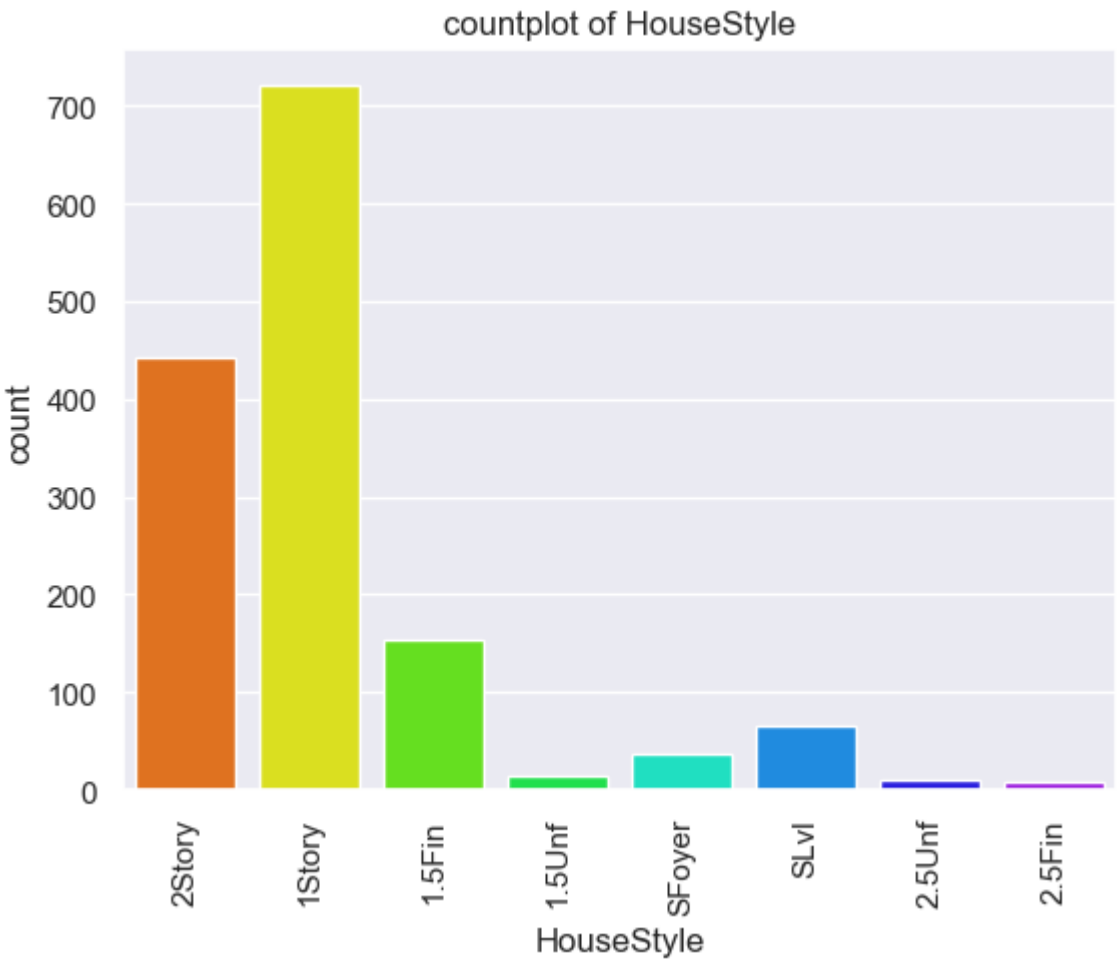

countplot of LandSlope
1400
1200
1000
800
600
400
200
0
count
Gtl
Mod
Sev
LandSlope

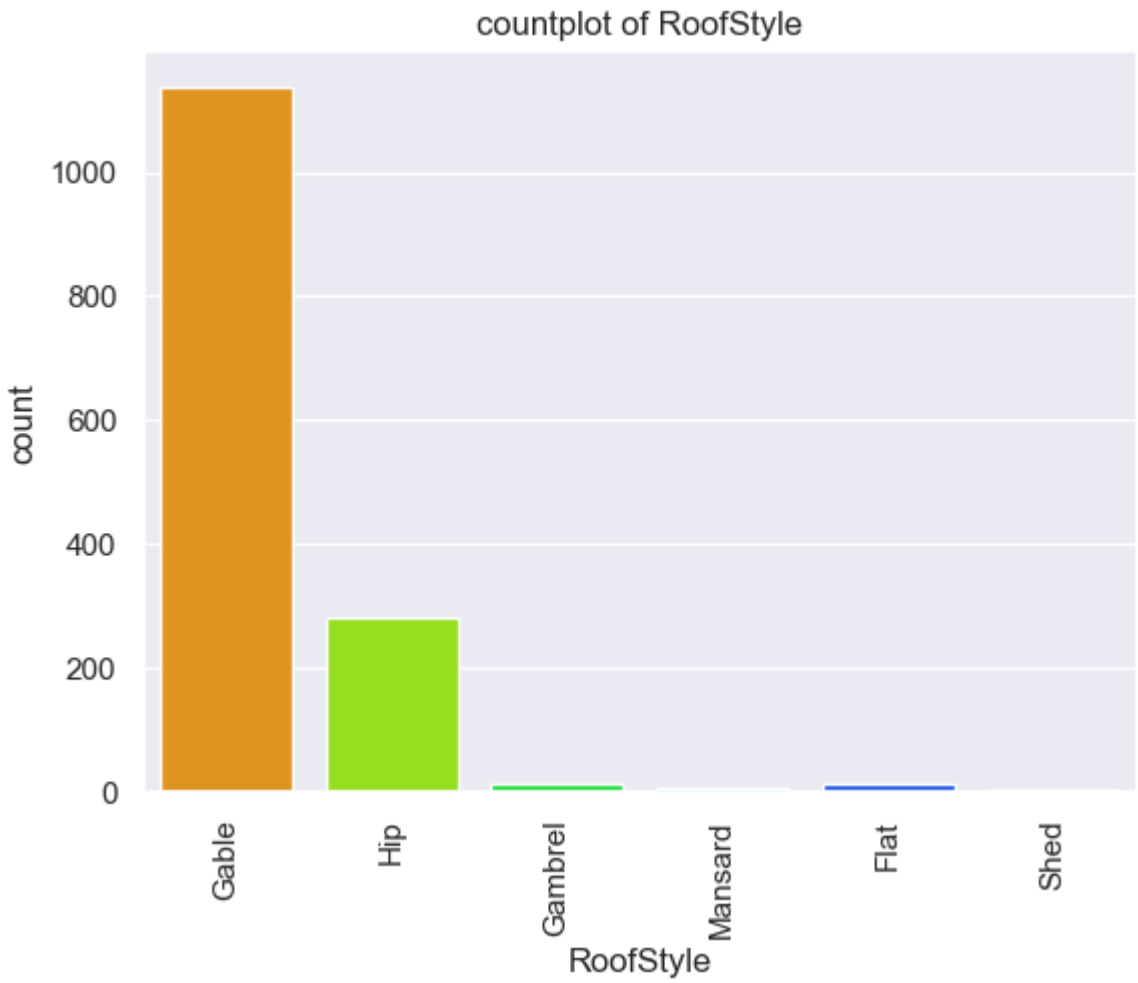

countplot of Neighborhood
count
0
50
100
150
200
CollgCr
Veenker
Crawfor
NoRidge
Mitchel
Somerst
NWAmes
OldTown
BrkSide
Sawyer
NridgHt
NAmes
SawyerW
IDOTRR
MeadowV
Edwards
Timber
Gilbert
StoneBr
ClearCr
NPkVill
Blmngtn
BrDale
SWISU
Blueste
Neighborhood

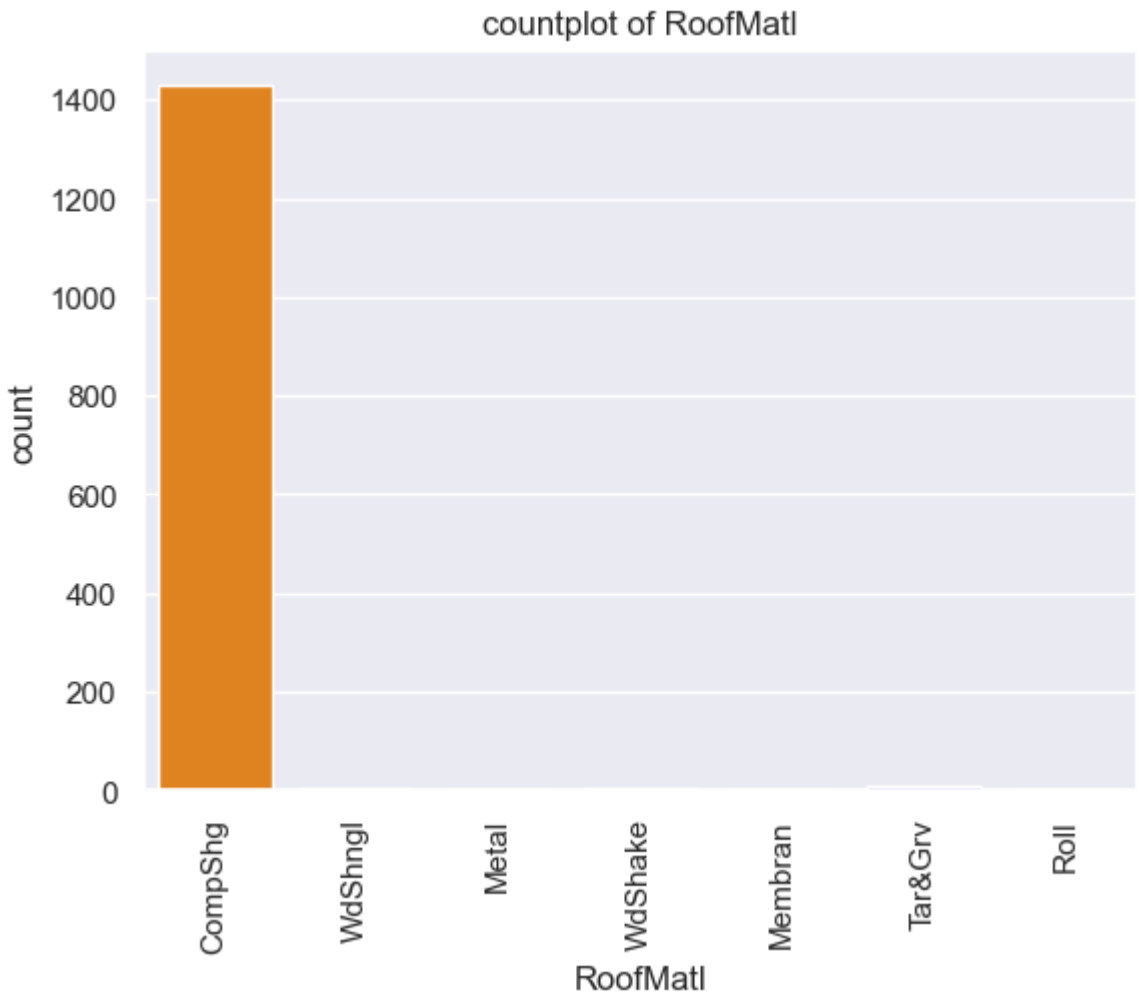

countplot of Condition1
count
0
200
400
600
800
1000
1200
Norm
Feedr
PosN
Artery
RRAe
RRNn
RRAn
PosA
RRNe
Condition1

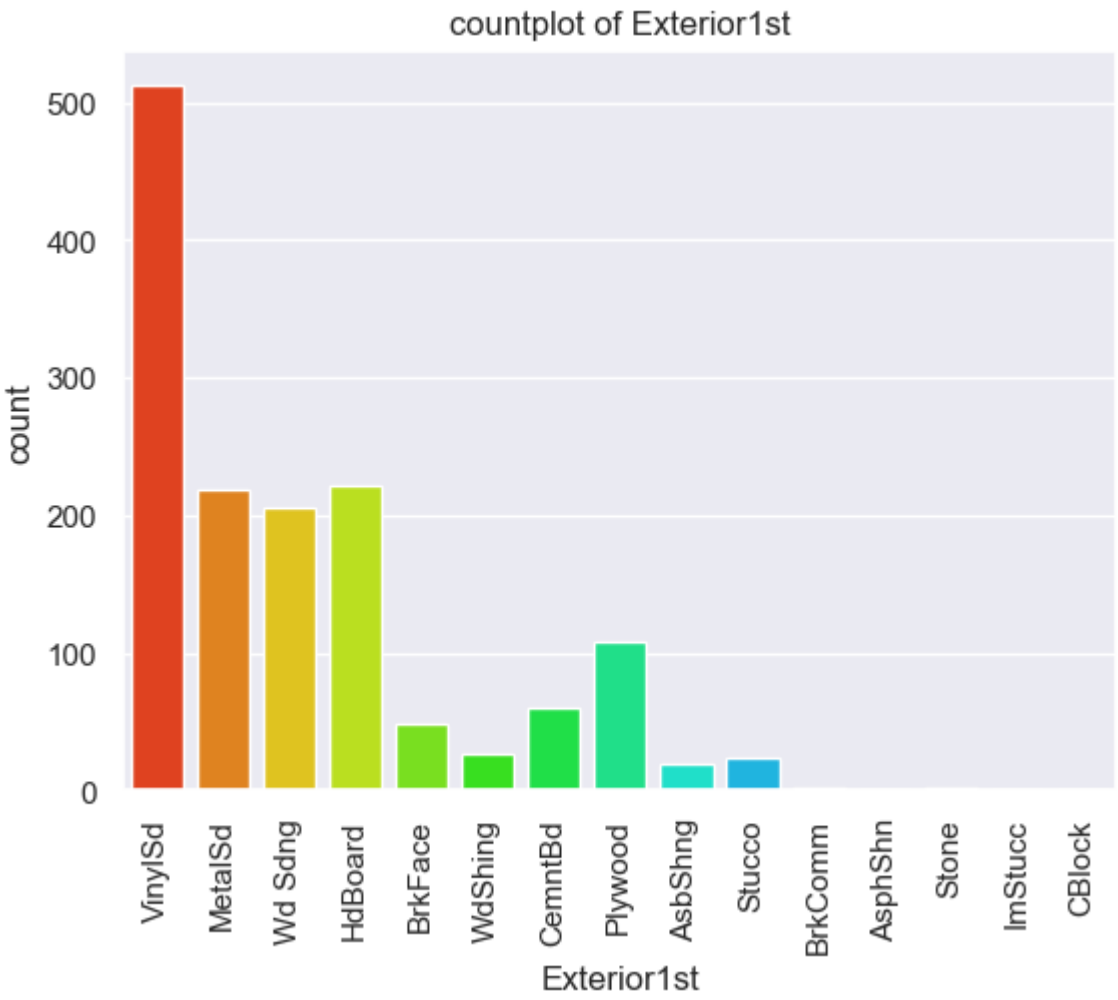

countplot of Condition2
count
0
200
400
600
800
1000
1200
1400
Norm
Artery
RRNn
Feedr
PosA
PosN
RRAn
RRAe
Condition2

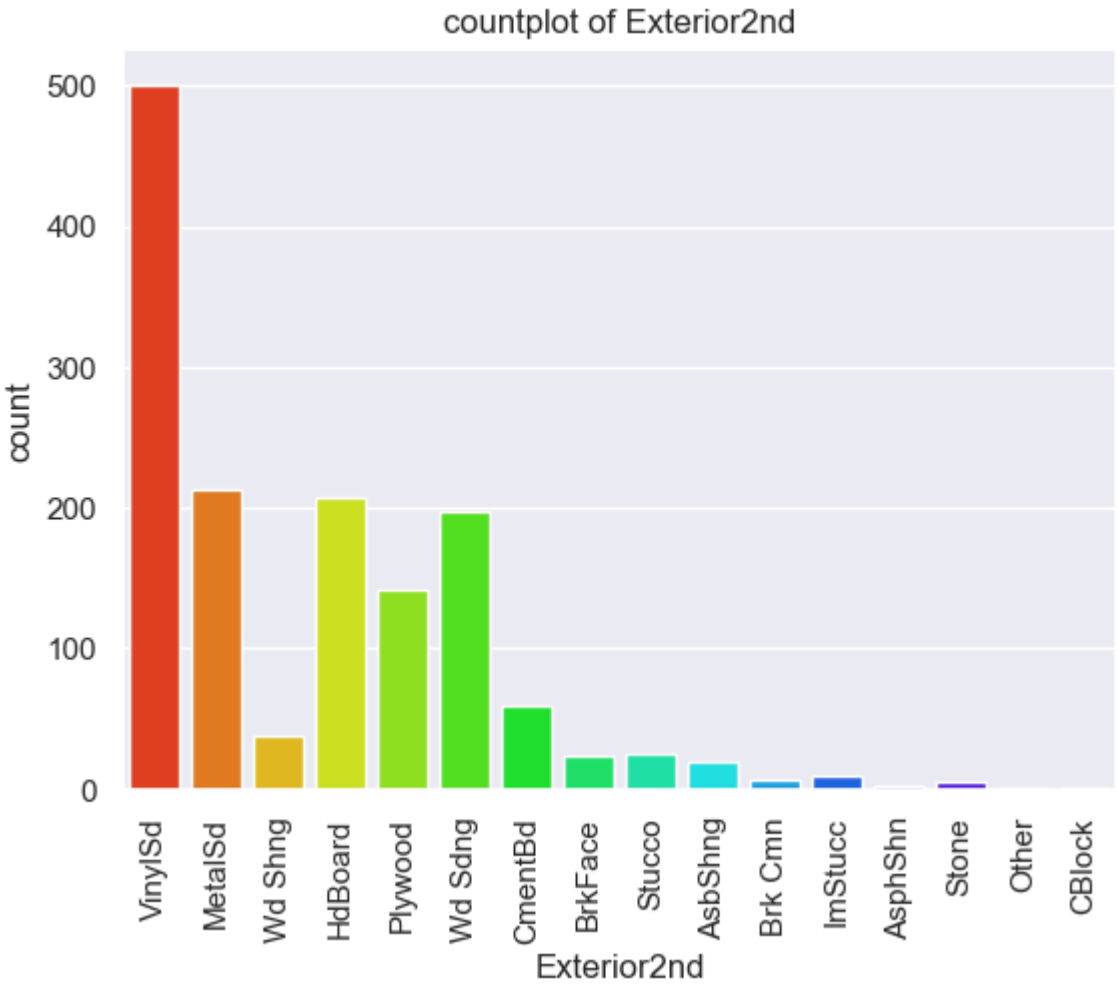

countplot of BldgType
count
1200
1000
800
600
400
200
0
1Fam
2fmCon
Duplex
TwnhsE
Twnhs
BldgType

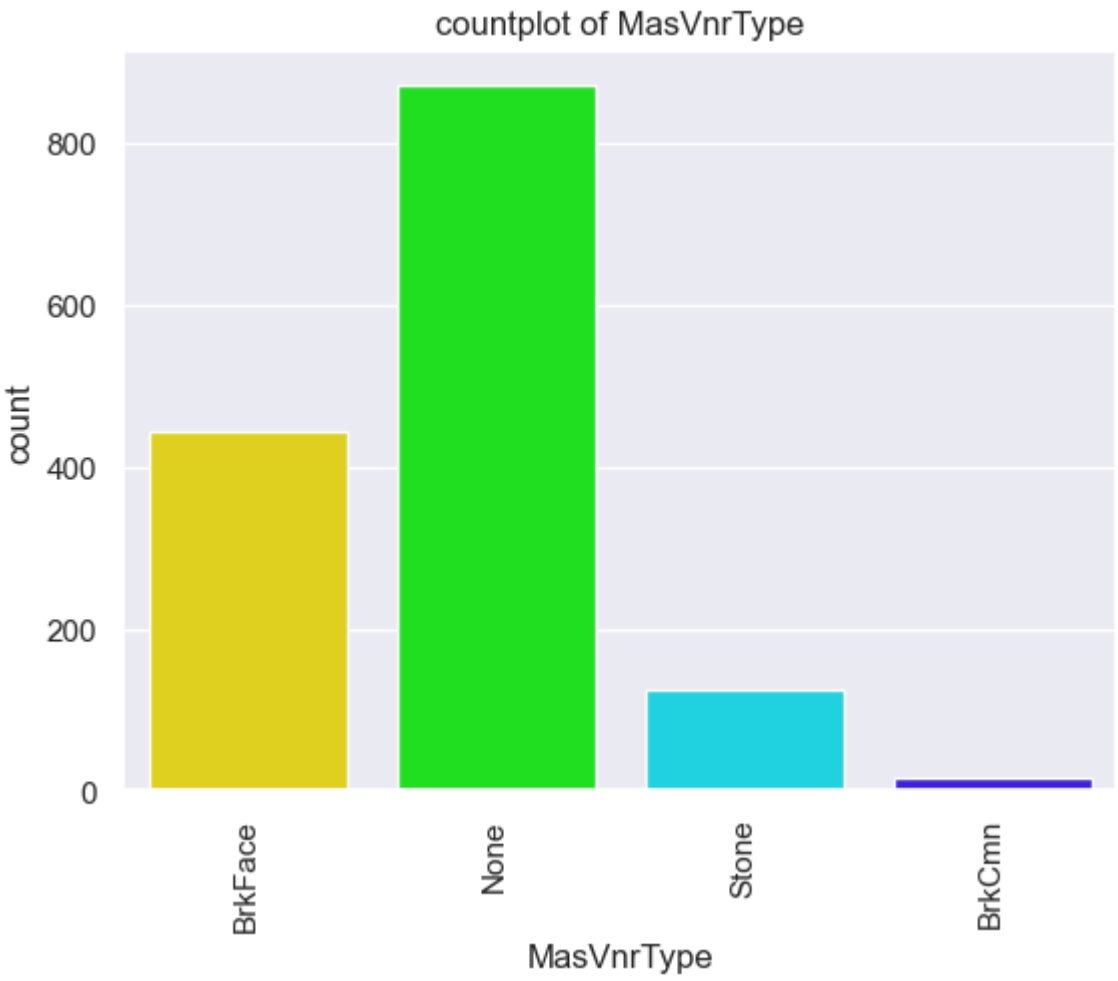

countplot of HouseStyle
count
0
100
200
300
400
500
600
700
2Story
1Story
1.5Fin
1.5Unf
SFoyer
SLvl
2.5Unf
2.5Fin
HouseStyle

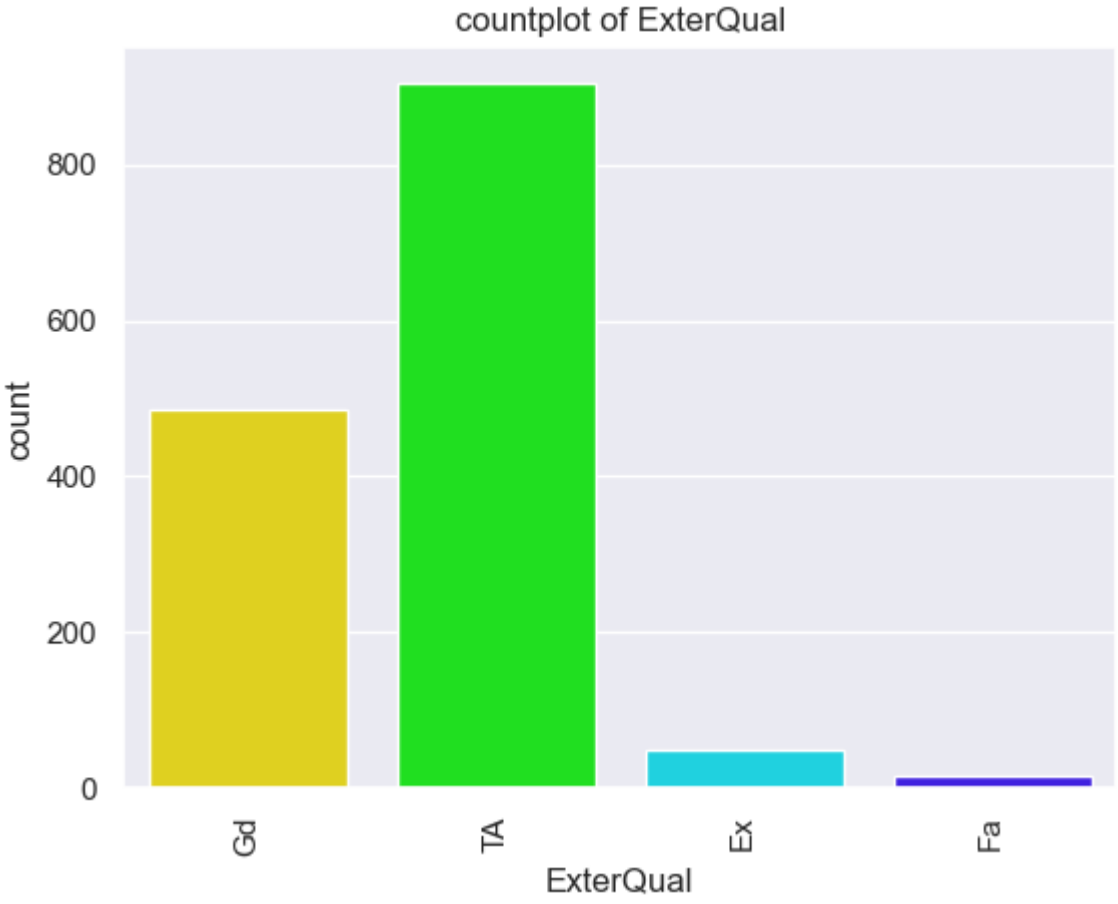

countplot of RoofStyle
count
1000
800
600
400
200
0
Gable
Hip
Gambrel
Mansard
Flat
Shed
RoofStyle

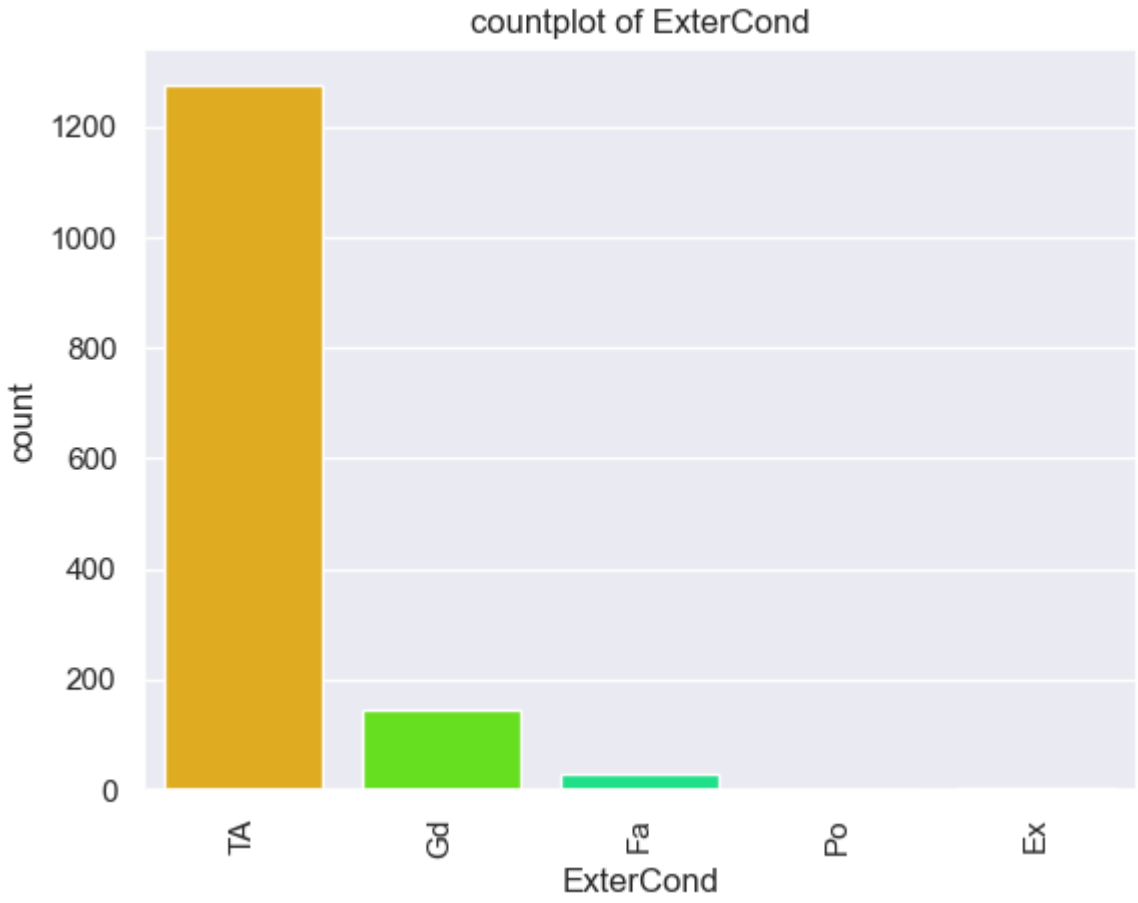

countplot of RoofMatl
count
1400
1200
1000
800
600
400
200
0
CompShg
WdShngl
Metal
WdShake
Membran
Tar&Grv
Roll
RoofMatl

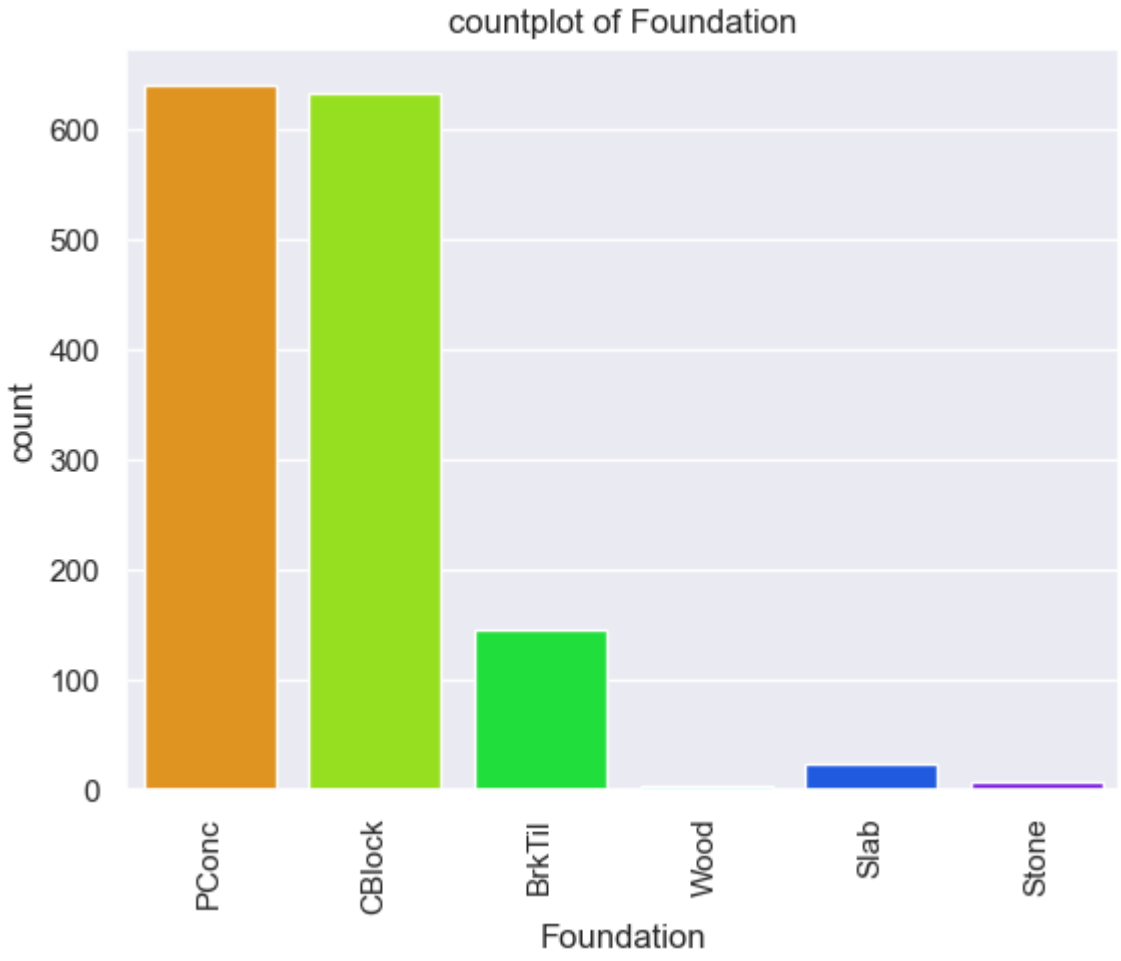

countplot of Exterior1st
count
500
400
300
200
100
0
VinylSd
MetalSd
Wd Sdng
HdBoard
BrkFace
WdShing
CemntBd
Plywood
AsbShng
Stucco
BrkComm
AsphShn
Stone
ImStucc
CBlock
Exterior1st

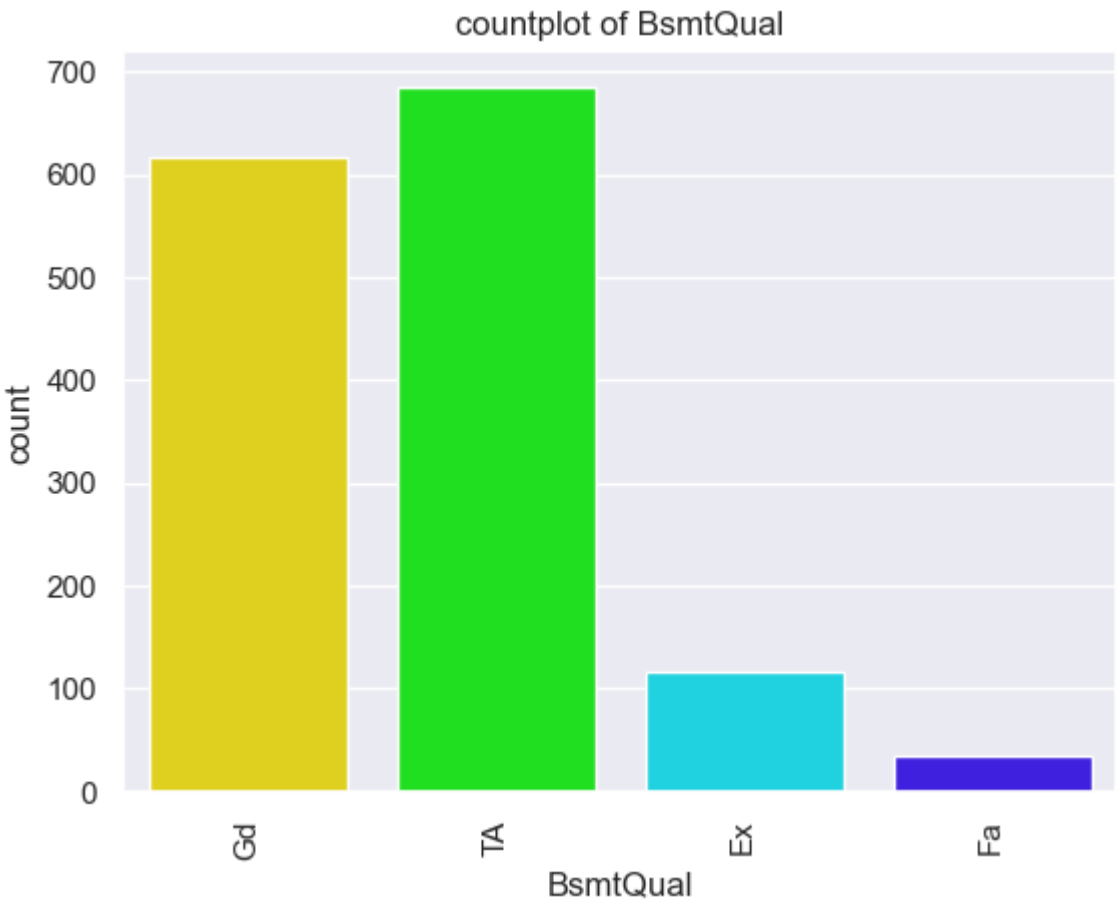

countplot of Exterior2nd
count
500
400
300
200
100
0
VinylSd
MetalSd
Wd Shng
HdBoard
Plywood
Wd Sdng
CmentBd
BrkFace
Stucco
AsbShng
Brk Cmn
ImStucc
AsphShn
Stone
Other
CBlock
Exterior2nd

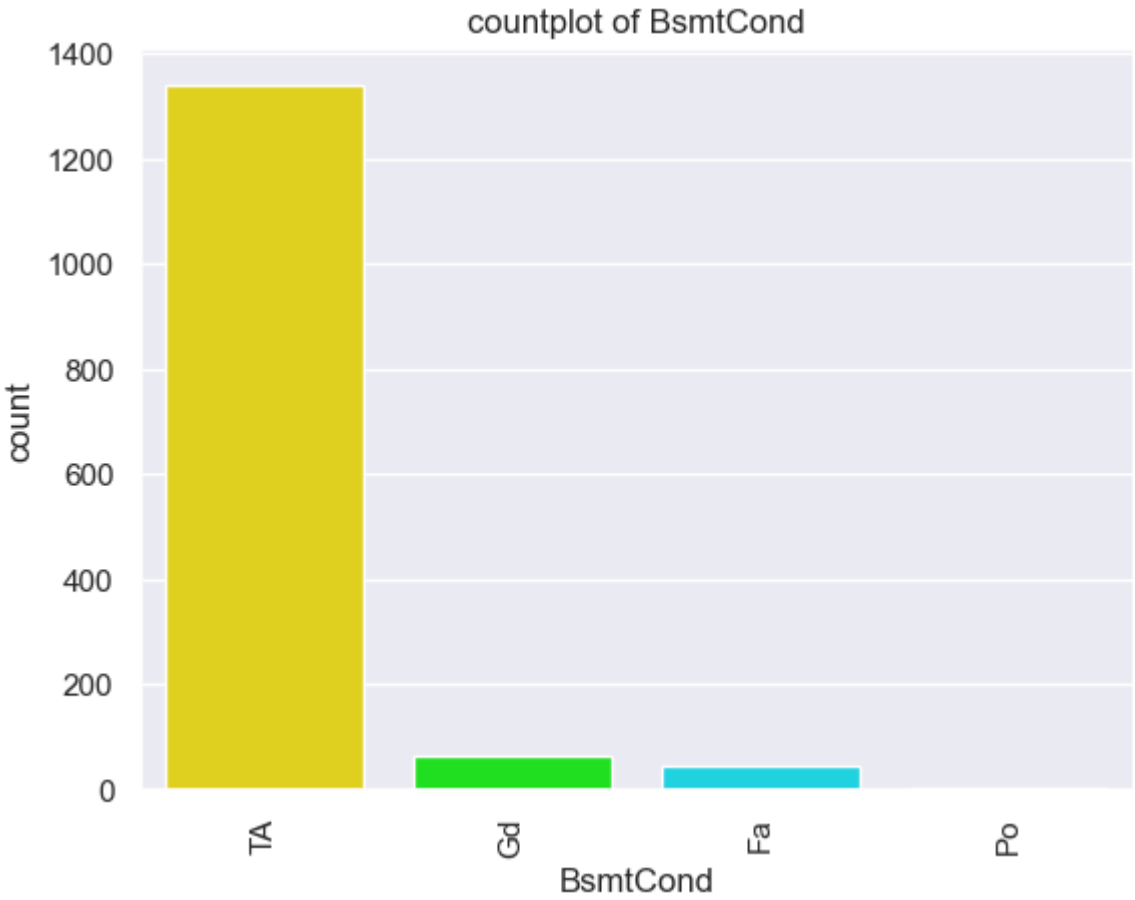

countplot of MasVnrType
count
800
600
400
200
0
BrkFace
None
Stone
BrkCmn
MasVnrType

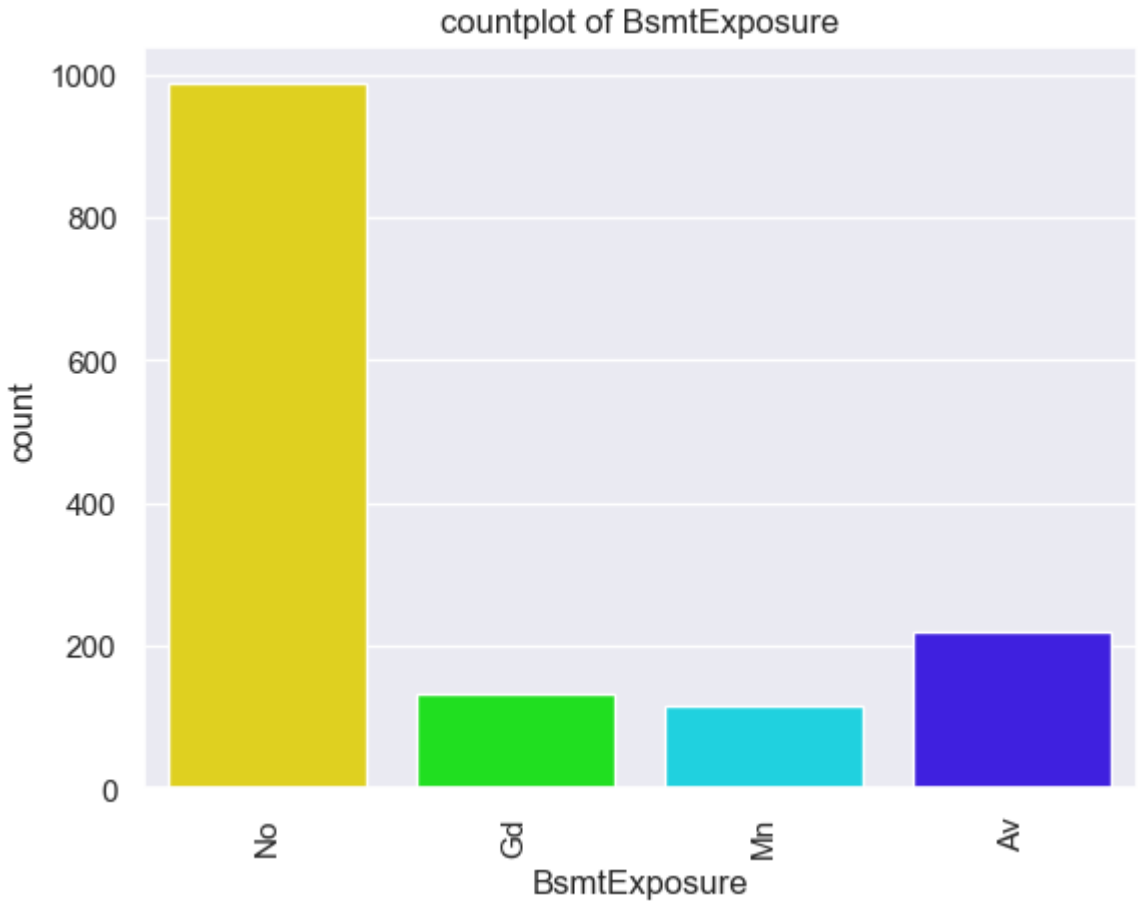

countplot of ExterQual
count
800
600
400
200
0
Gd
TA
Ex
Fa
ExterQual

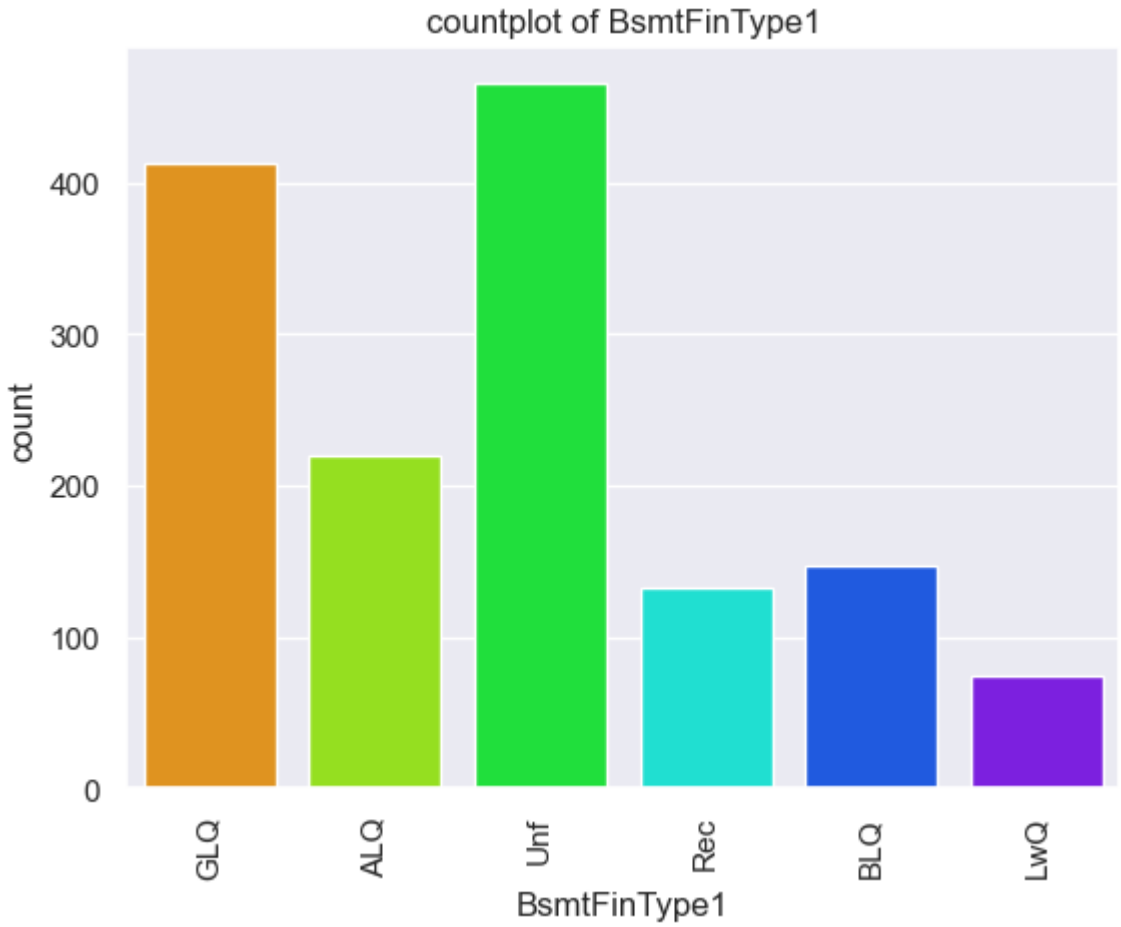

countplot of ExterCond
count
1200
1000
800
600
400
200
0
TA
Gd
Fa
Po
Ex
ExterCond

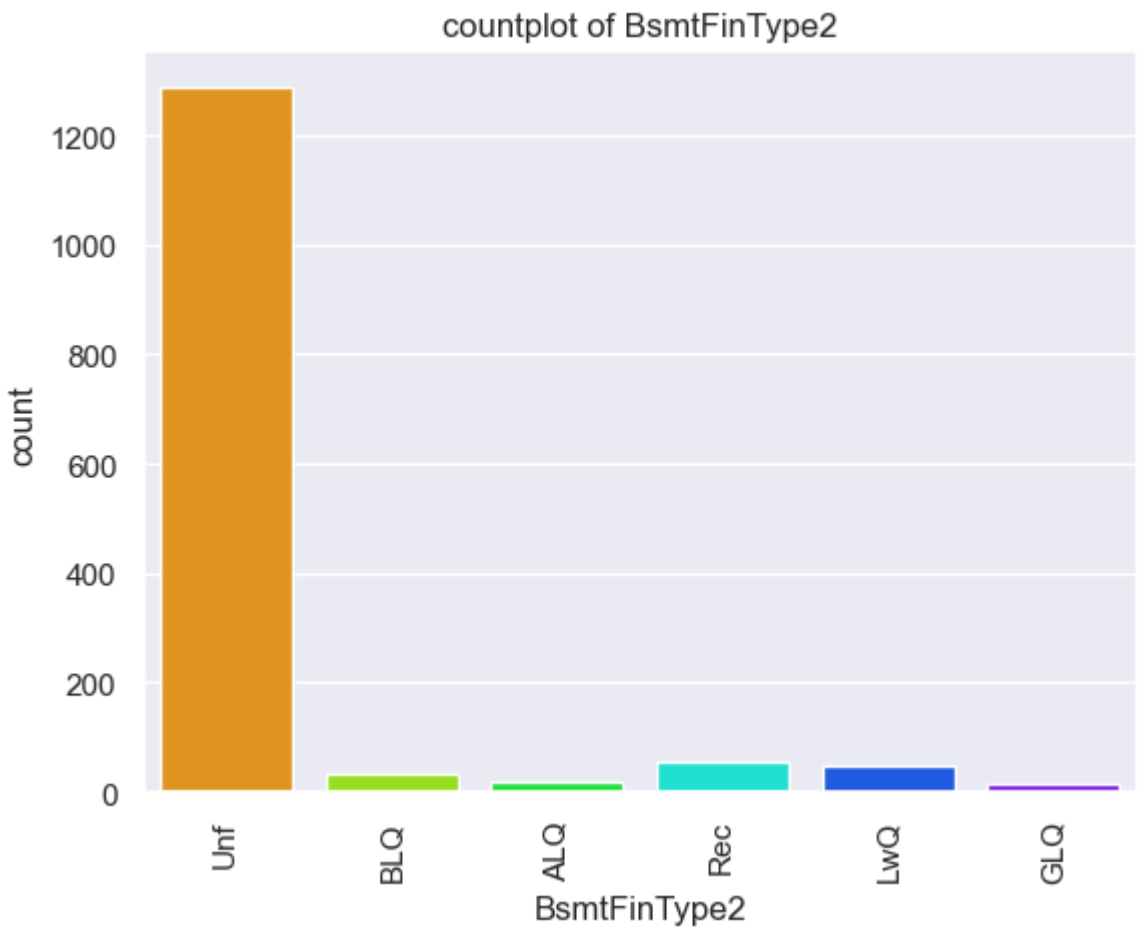

countplot of Foundation
count
600
500
400
300
200
100
0
PConc
CBlock
BrkTil
Wood
Slab
Stone
Foundation

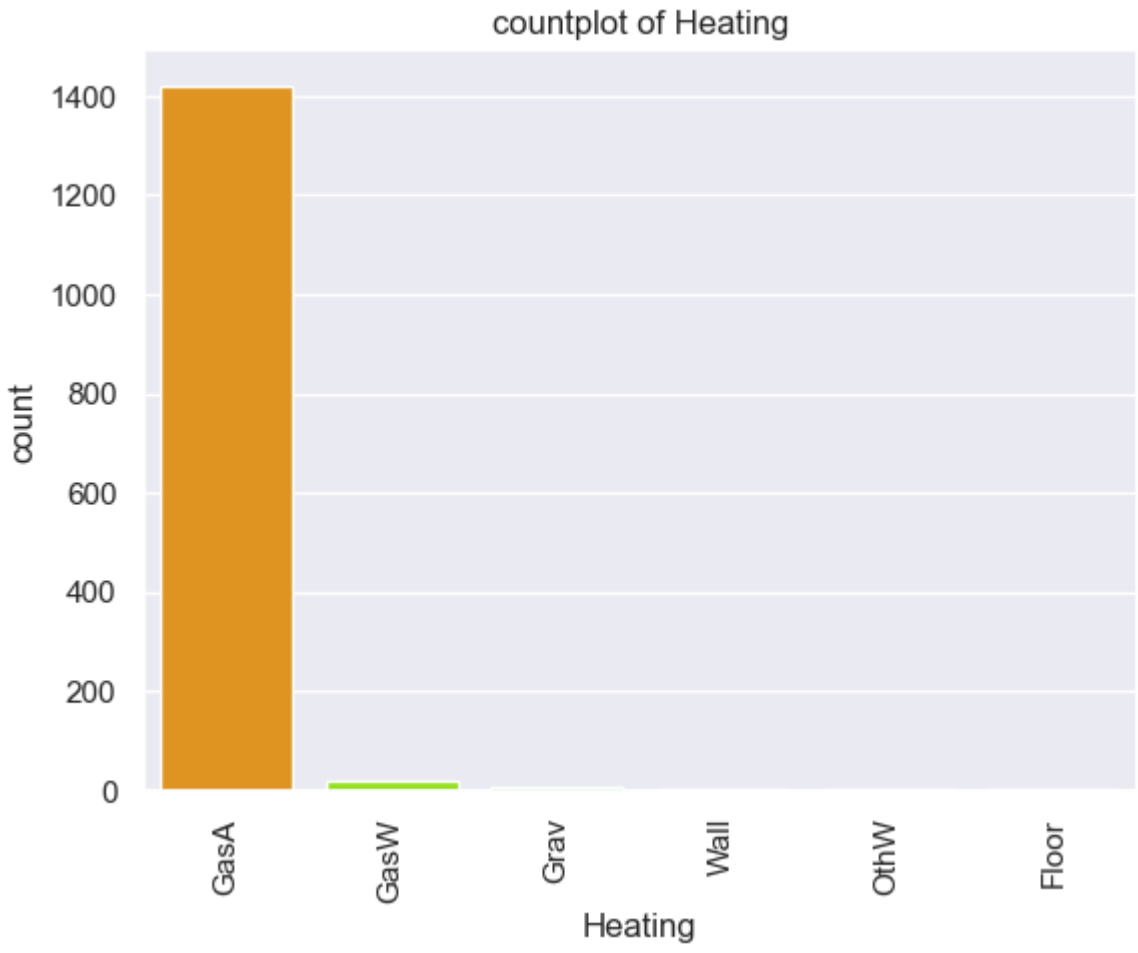

countplot of BsmtQual
700
600
500
400
300
200
100
0
count
Gd
TA
Ex
Fa
BsmtQual

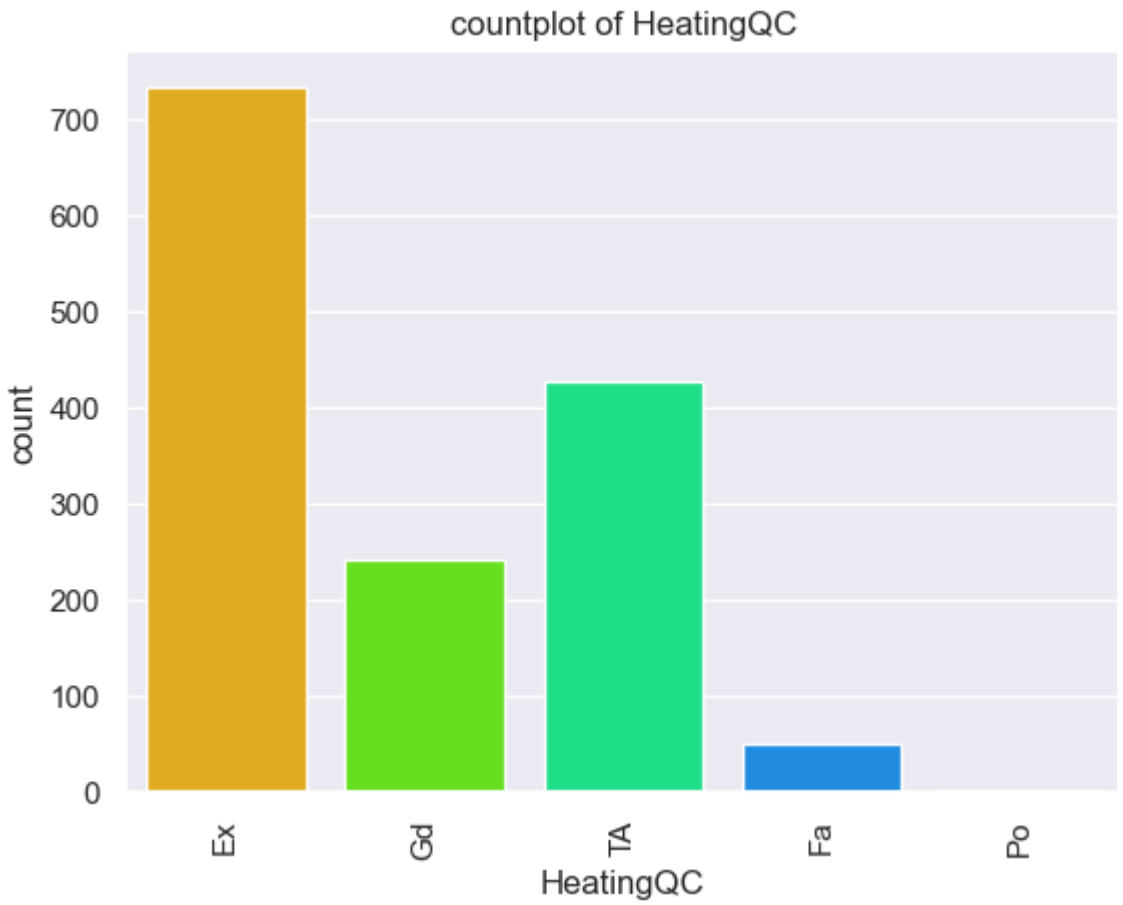

countplot of BsmtCond
1400
1200
1000
800
600
400
200
0
count
TA
Gd
Fa
Po
BsmtCond

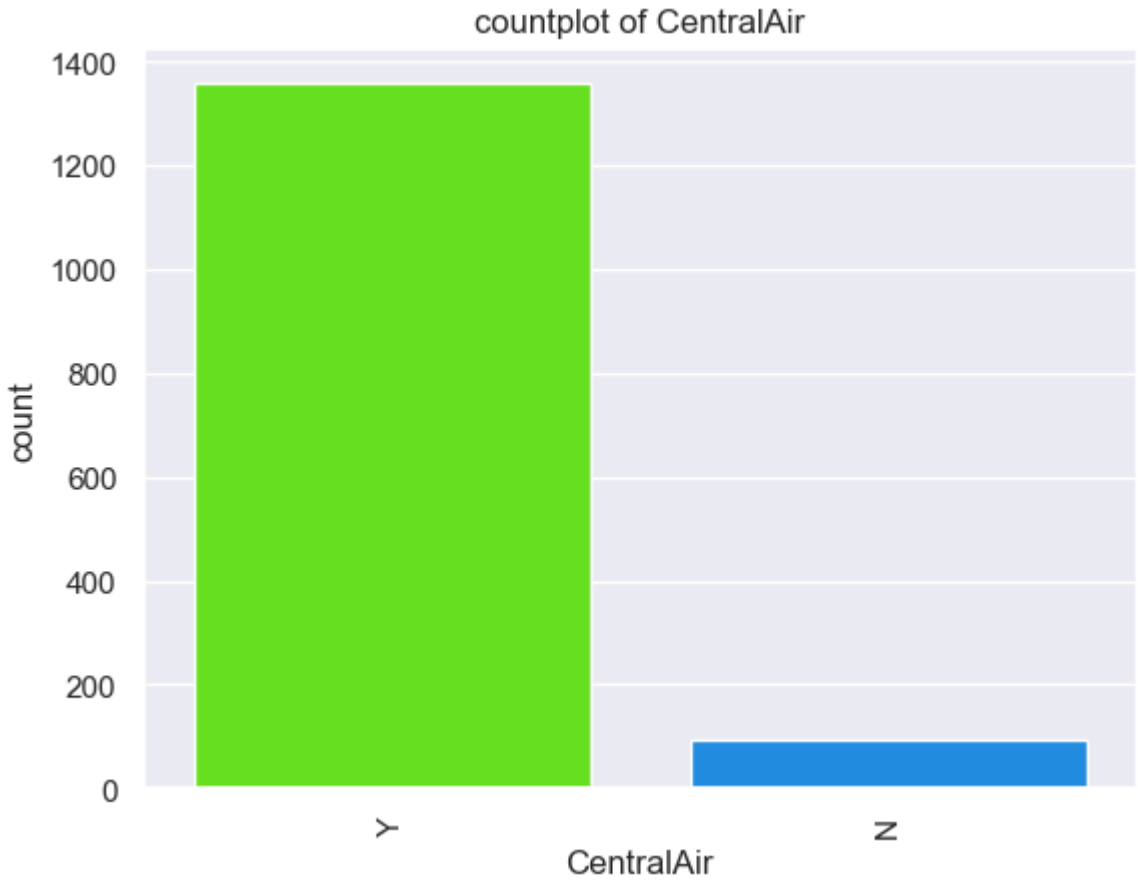

countplot of BsmtExposure
1000
800
600
400
200
0
count
No
Gd
Mn
Av
BsmtExposure

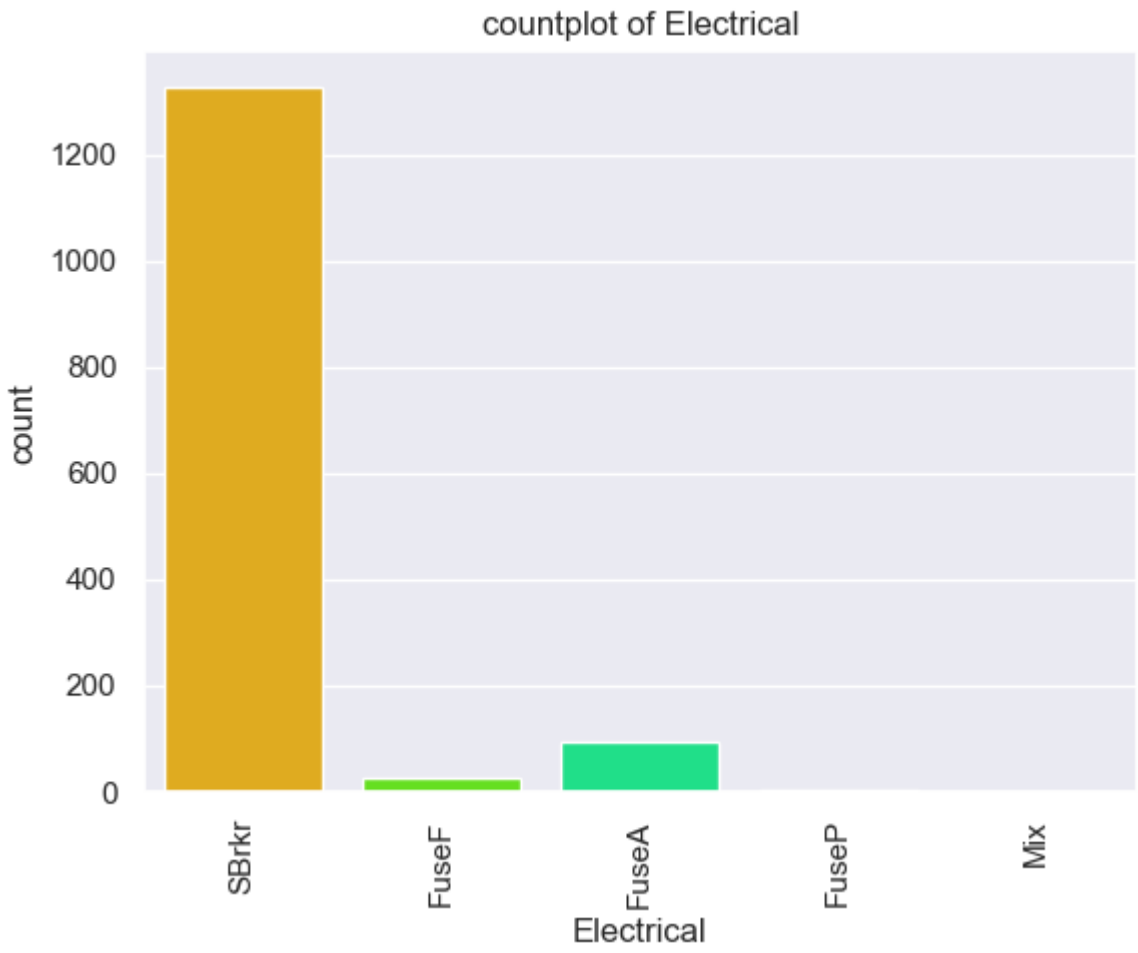

countplot of BsmtFinType1
400
300
200
100
0
count
GLQ
ALQ
Unf
Rec
BLQ
LwQ
BsmtFinType1

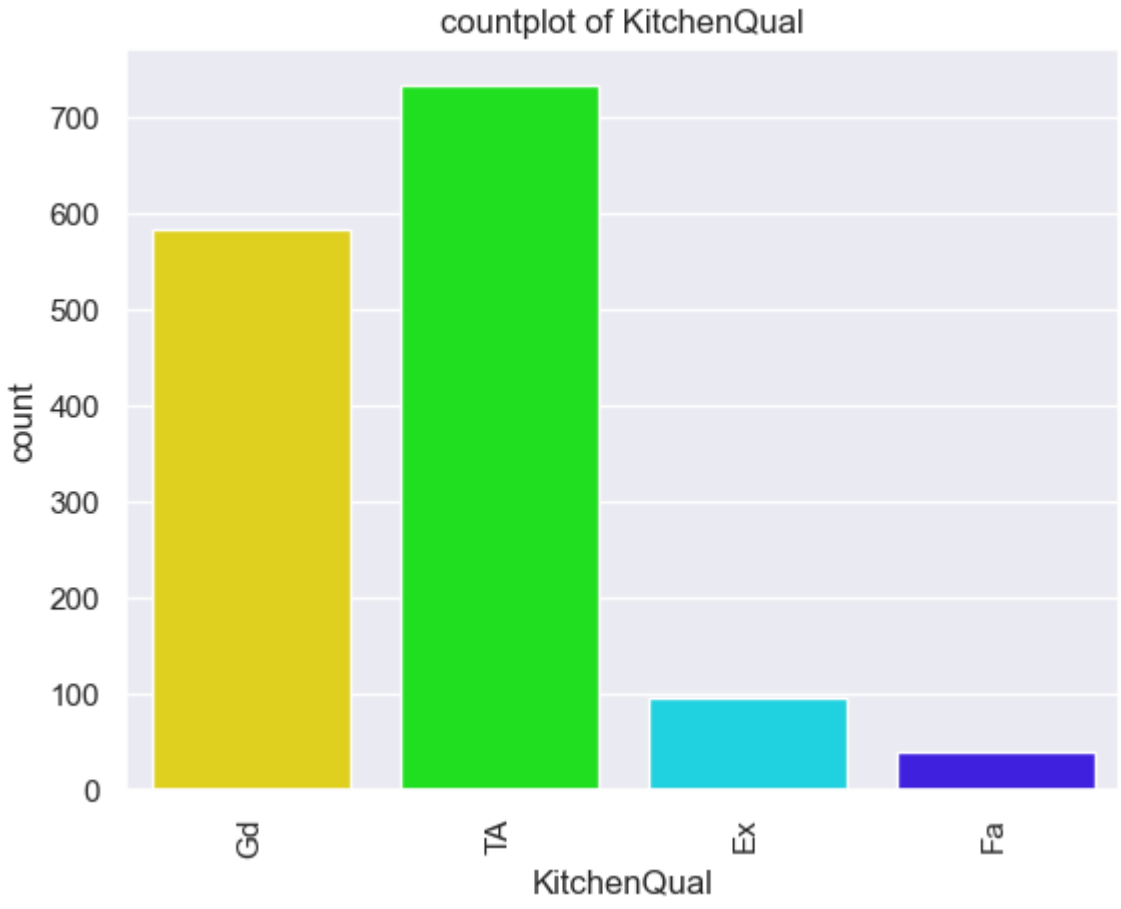

countplot of BsmtFinType2
count
1200
1000
800
600
400
200
0
Unf
BLQ
ALQ
Rec
LwQ
GLQ
BsmtFinType2

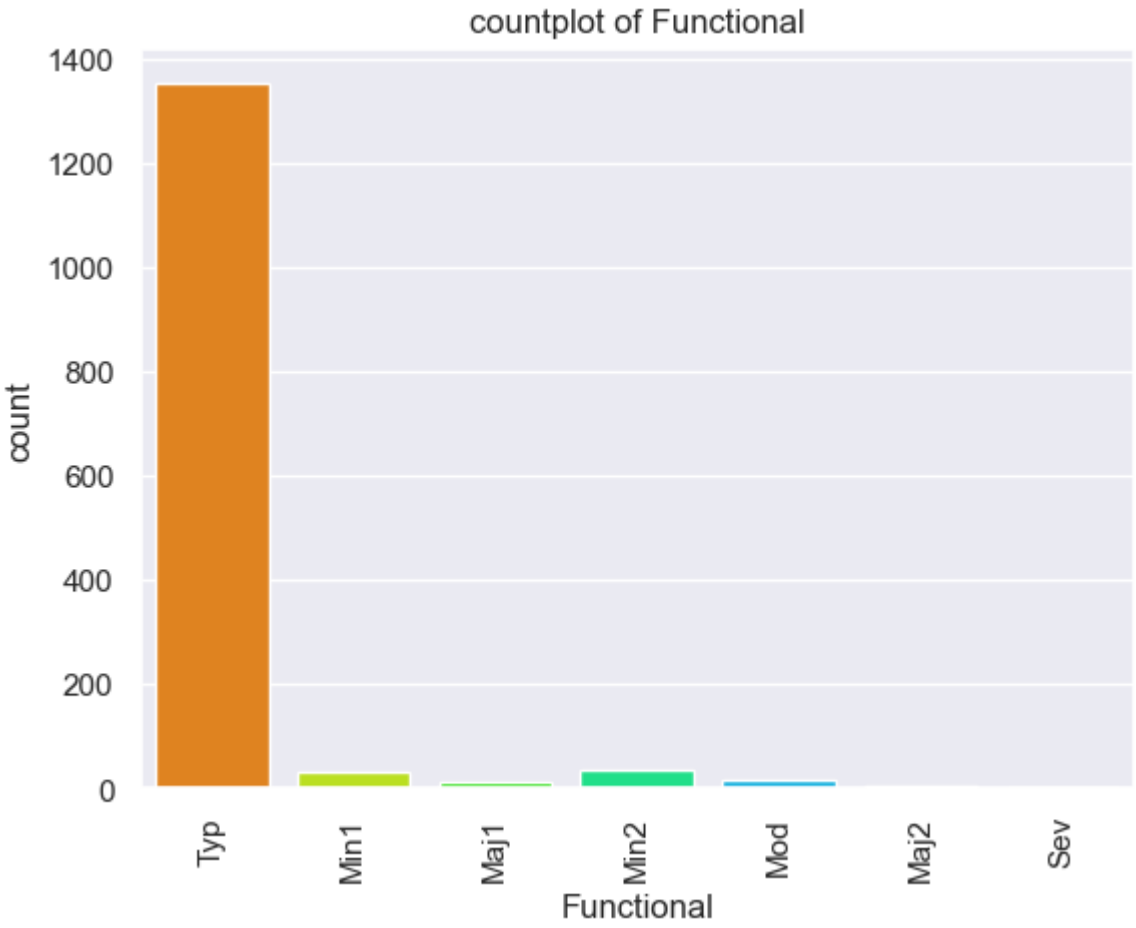

countplot of Heating
count
1400
1200
1000
800
600
400
200
0
GasA
GasW
Grav
Wall
OthW
Floor
Heating

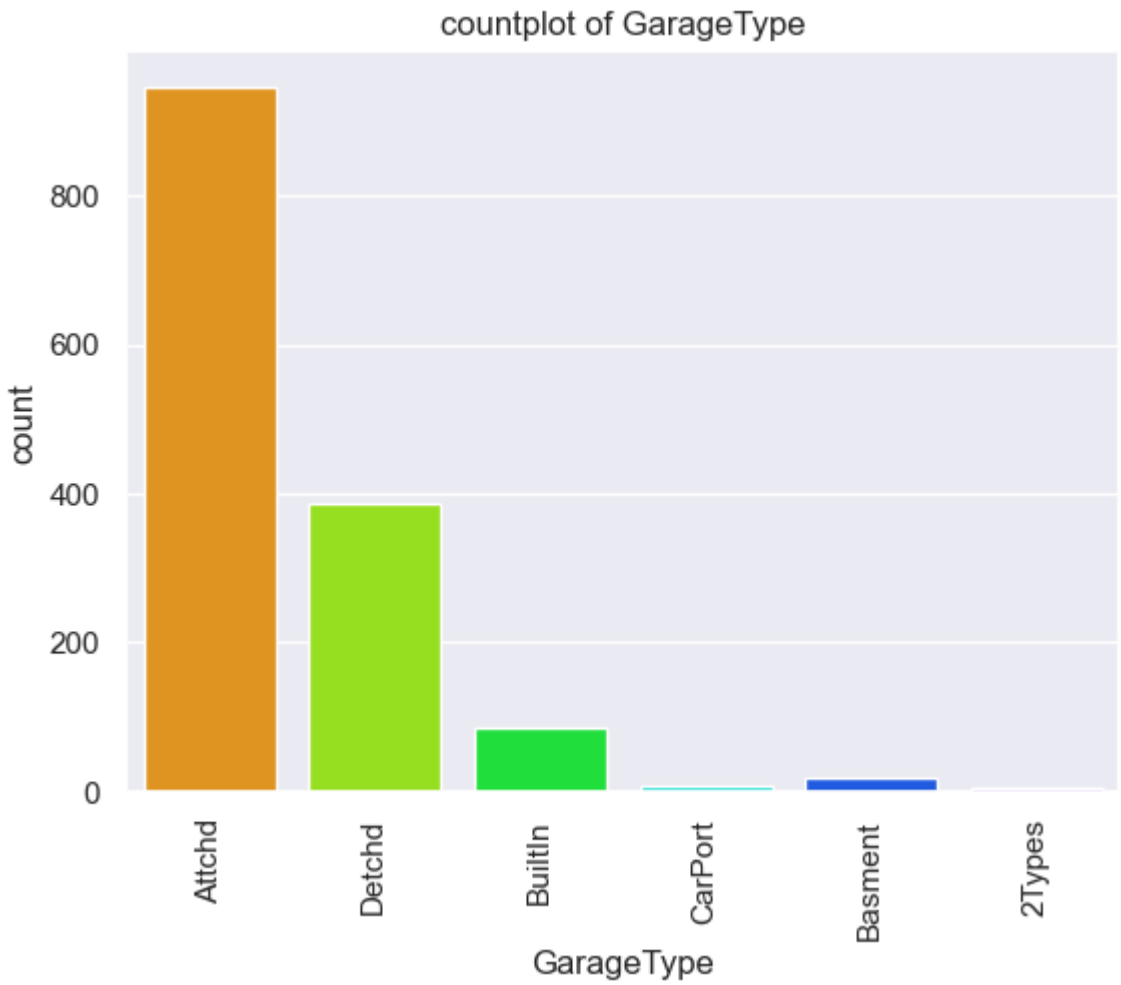

countplot of HeatingQC
700
600
500
400
300
200
100
0
count
Ex
Gd
TA
Fa
Po
HeatingQC

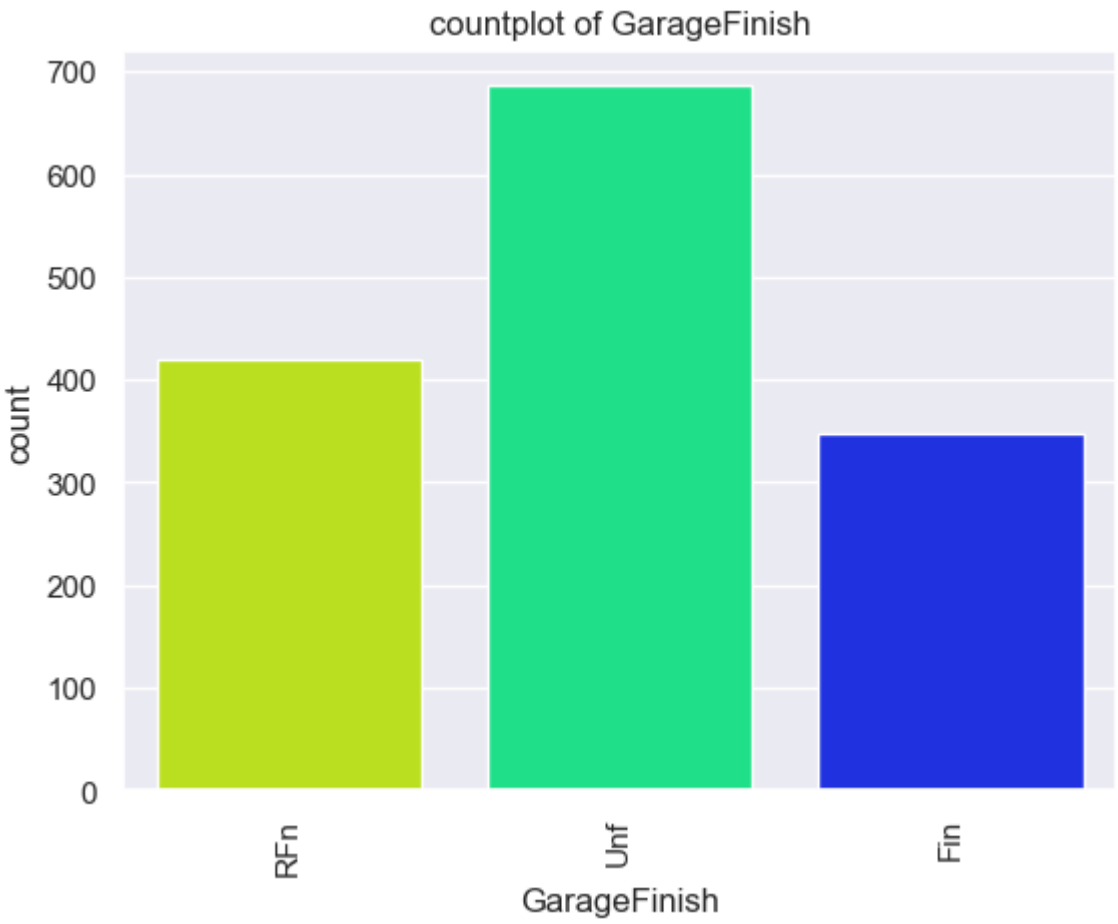

countplot of CentralAir
1400
1200
1000
800
600
400
200
0
count
Y
N
CentralAir

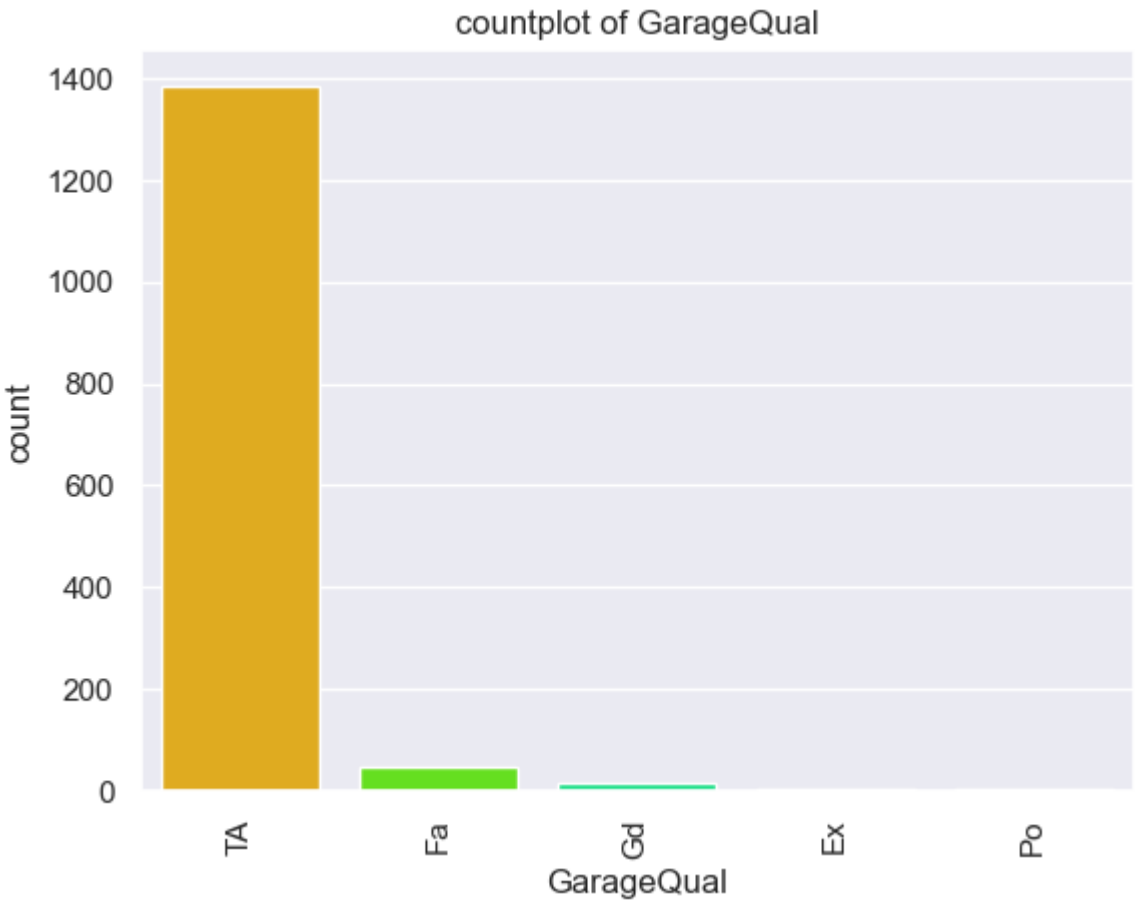

countplot of Electrical
count
0
200
400
600
800
1000
1200
SBrkr
FuseF
FuseA
FuseP
Mix
Electrical

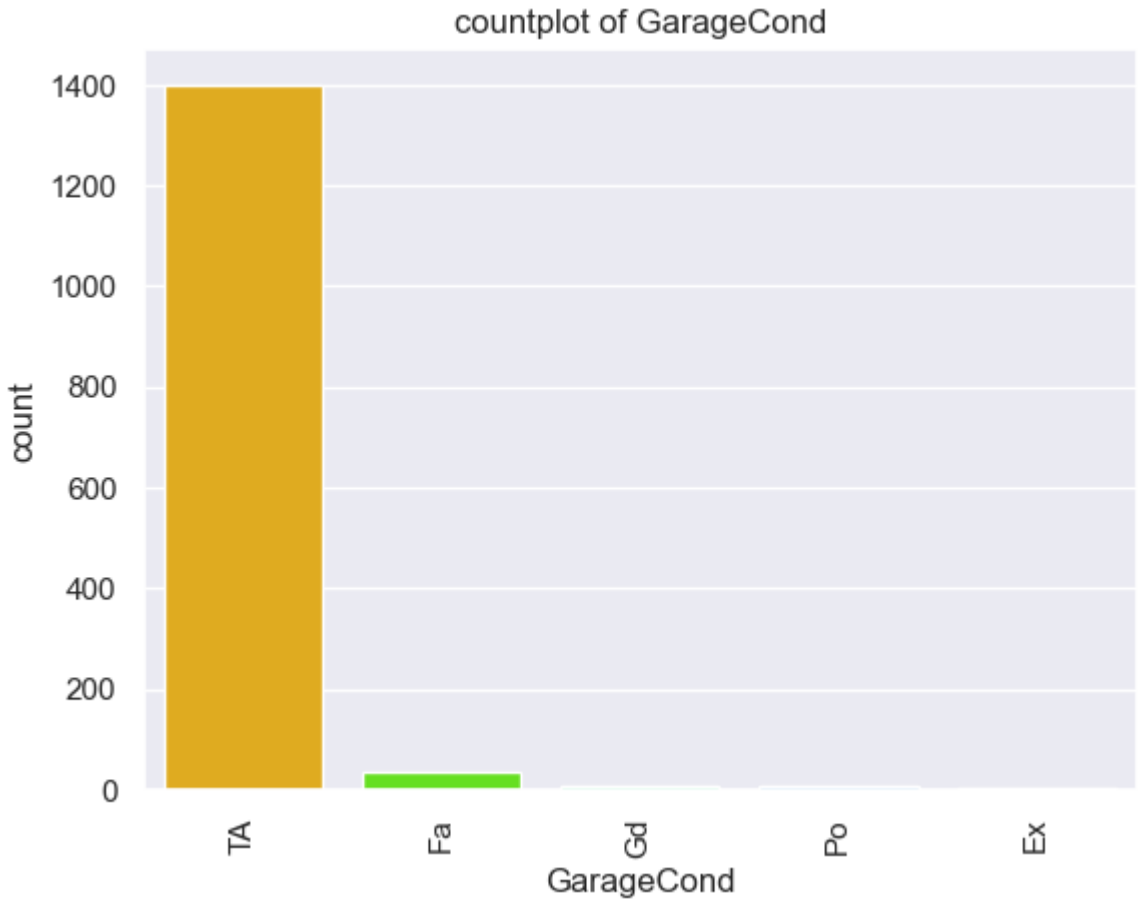

countplot of KitchenQual
count
0
100
200
300
400
500
600
700
Gd
TA
Ex
Fa
KitchenQual

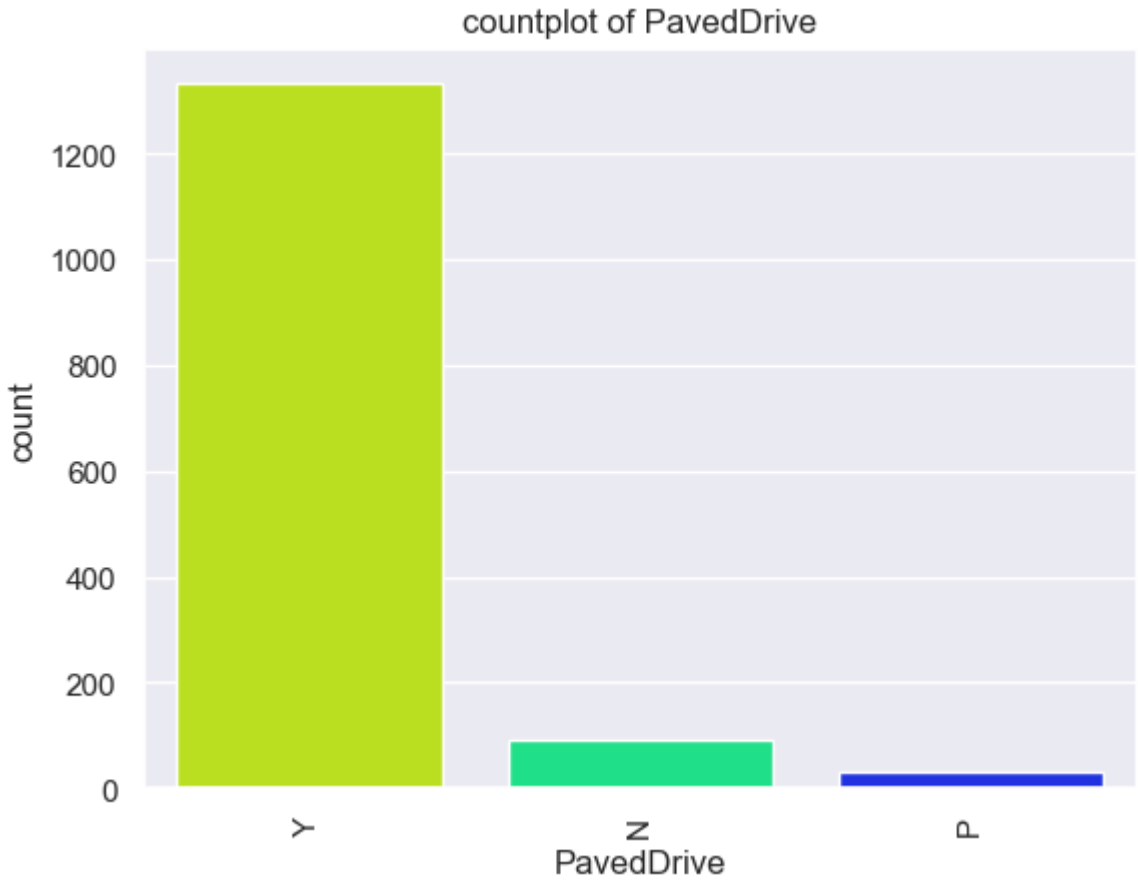

countplot of Functional
1400
1200
1000
800
600
400
200
0
count
Typ
Min1
Maj1
Min2
Mod
Maj2
Sev
Functional

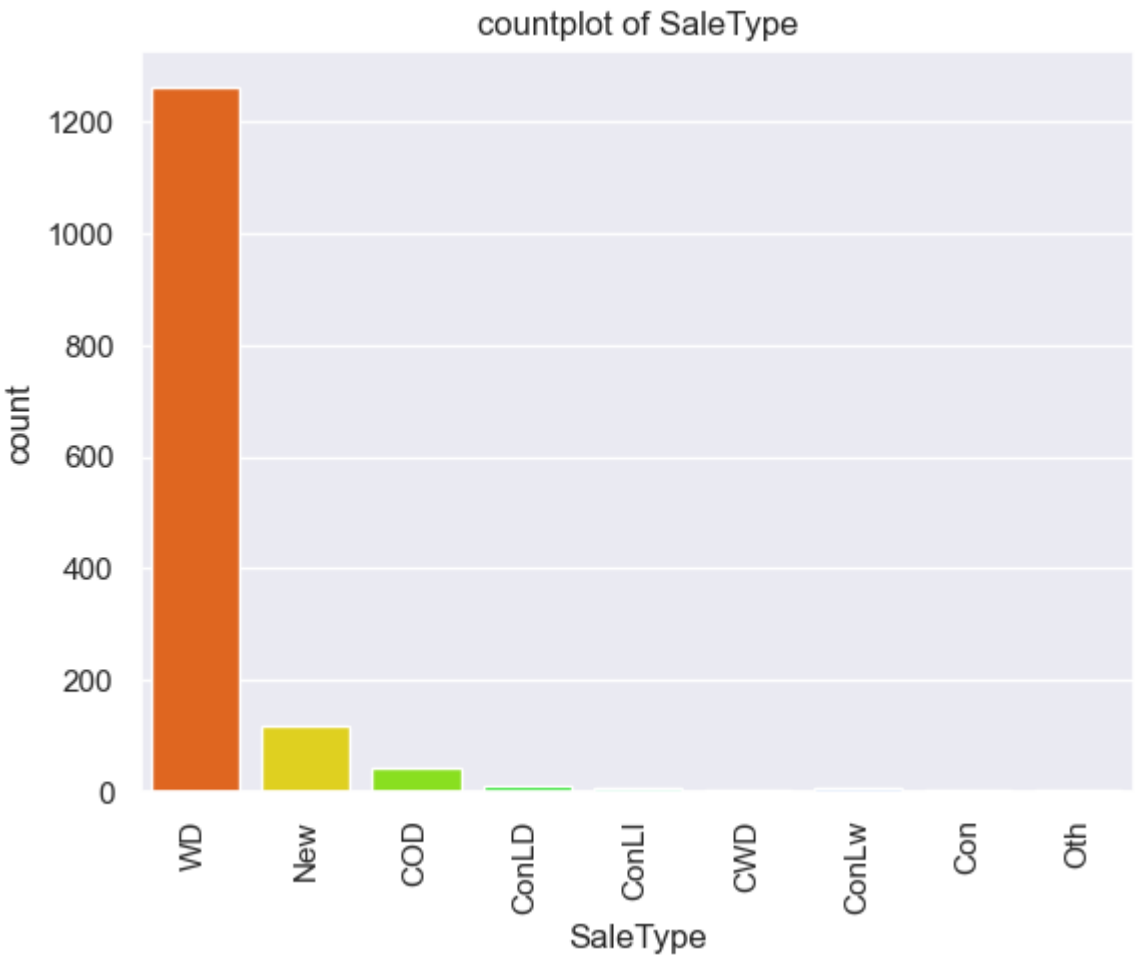

countplot of GarageType
count
800
600
400
200
0
Attchd
Detchd
BuiltIn
CarPort
Basment
2Types
GarageType

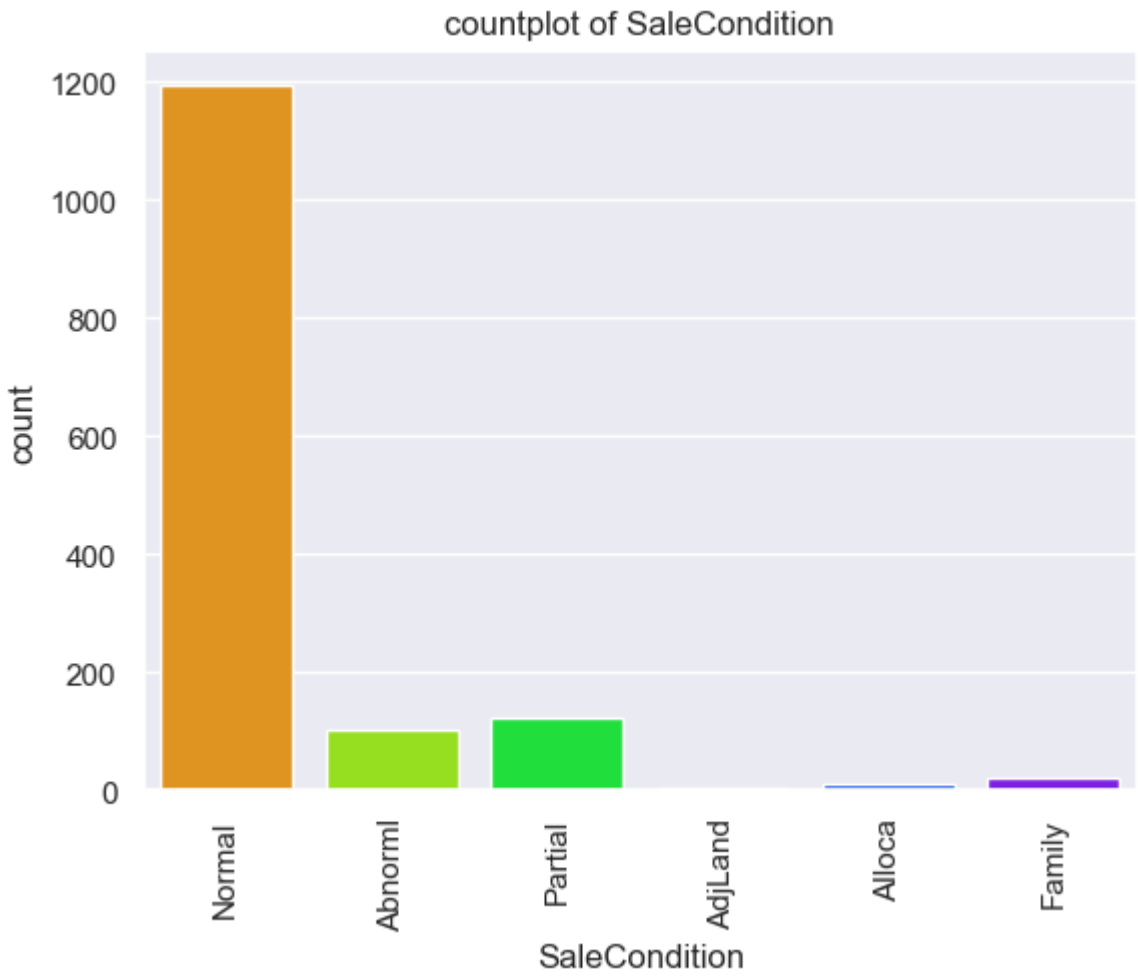

countplot of GarageFinish
700
600
500
400
300
200
100
0
count
RFn
Unf
Fin
GarageFinish


countplot of GarageQual
1400
1200
1000
800
600
400
200
0
count
TA
Fa
Gd
Ex
Po
GarageQual


countplot of GarageCond
count
1400
1200
1000
800
600
400
200
0
TA
Fa
Gd
Po
Ex
GarageCond


countplot of PavedDrive
count
1200
1000
800
600
400
200
0
Y
N
P
PavedDrive


countplot of SaleType
count
1200
1000
800
600
400
200
0
WD
New
COD
ConLD
ConLI
CWD
ConLw
Con
Oth
SaleType



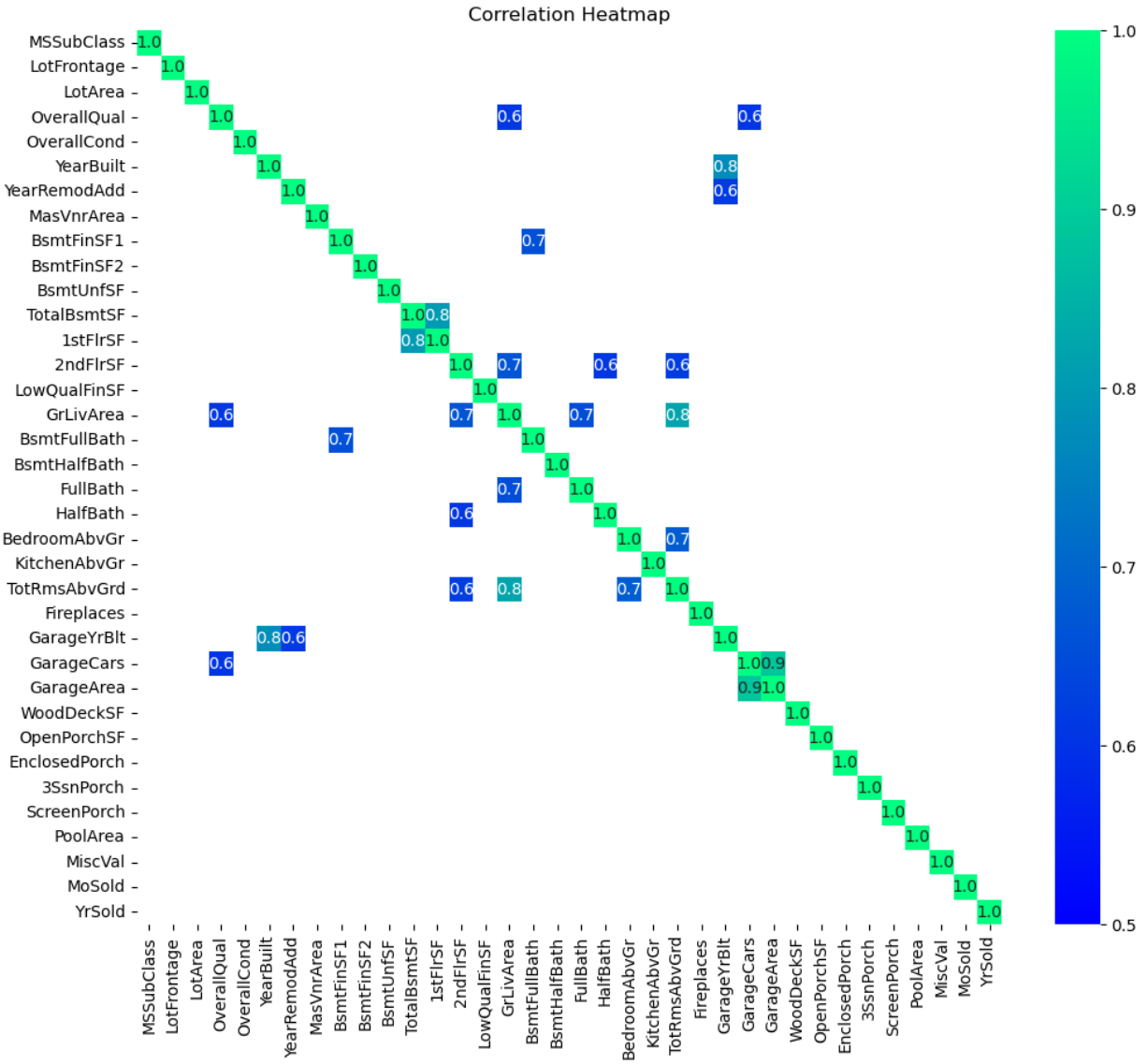
INSIGHTS FROM ABOV GRAPHS OF CATEGORICAL COLUMNS : 1. MSzoning - most RL category is most population- Residential Low Density 2. Street - most street pave have more peoples are lived. /more sales 3. LotShape- General shape of property is most count of REG,IR1- REGULAR 4.LandContour: Flatness of the property- MOST COUNTS Near Flat/Level. 5. RoofStyle: Type of roof- most type are Gable then hip less 6. LotConfig: Lot configuration - most inside lot counts 7.landslope- GTL (gentel)TYPE slope is most there 8.Neighborhood: Physical locations within this city limits - all neighborhood most cities are there # # INSIGHTS FROM ABOV GRAPHS OF CATEGORICAL COLUMNS :basement features. 9.BsmtQual: Evaluates the height of the basement- most houses of Gd Good (90-99 inches) TA Typical (80-89 inches) 10.BsmtExposure: Refers to walkout or garden level walls-: No-No Exposure most found in most houses 11.BsmtFinType1: Rating of basement finished area: GLQ- Good Living Quarters, Below Average Living Quarters : most found oridnal categories. 12. BsmtFinType2: Rating of basement finished area (if multiple types)-: mostly Unf-Unfinshed basement is there so we can drop this column because it is not important for model and sale price. - HeatingQC Ex 731 most property excellent shown - ordinal category - CentralAir Y 1353- most houses have Central air conditioning - GarageCars: Size of garage in car capacity not affecting saleprice: so drop - GarageArea: Size of garage in square feet - i keep # GarageQual, GarageCond same values so we can drop any one # i will drop same columns charactersitcs - BldgType: Type of dwelling, HouseStyle: Style of dwelling drop any 1 . # GARAGE TYPE- MOSTLY AVAREGE TYPE - TA:GarageQual-TA 1380 : DROP # BldgType - MOST TYPE 1Fam 120: BldgType: Type of dwelling and HouseStyle: Style of dwelling : same type of meaning so i drop : BldgType : shows same characteristics not affect price .it not showing too much building type # df.GarageCond.describe().T :top TAfreQ 1407,

BIVARIATE Analysis:

- Need to find out more about the relationship between all these features with one another.
- Alreay plot scatterplot of highly correlated 10 features with saleprice.
- Also plot heatmap .

In [ ]:
```
- Heatmap : checking correlation between columns
```

In [49]:
```
# Create correlation matrix from train data excluding `SalePrice`
corr_mat = df.iloc[:, :-1].corr() # all columns except Sale price column

# Select correlations greater than 0.6
high_corr_mat = corr_mat[abs(corr_mat) >= 0.6] # highly strongly correlated features.

# Plot correlation heatmap

plt.figure(figsize=(12, 10))
sns.heatmap(high_corr_mat,
            annot=True,
            fmt='.1f',
            cmap='winter',
            vmin=0.5,
            vmax=1)

title = plt.title('Correlation Heatmap')
```



There is multicollinearity in our df dataset. Below features are highly correlated: - # GarageCars and GarageArea - # GarageYrBlt and YearBuilt # 1stFlrSF and TotalBsmtSF # 'GarageYrBlt','YearBuilt HIGHLT correlated 0.80 so we can drop any1. # GrLivArea and TotRmsAbvGrd Multicolliniearity has a negative impact on our prediction models and

makes errors of our estimates increases. Therefore, for each pair of highly correlated features, I will drop a feature that has a lower correlation with SalePrice.

ALSO CHECK HOW MANY SAME ROW CONTAIN 0 VALUES not usful columns for further analysis :

In [44]:
```
# i see many columns  also have zeros :
col_zeros=df.eq(0).sum(axis=0)# for row
threshold=0.98 # set threshold upto 80% of zeros
zero_per=(df==0).mean()
print(zero_per)
coldrop=zero_per[zero_per>=threshold].index

print(coldrop)
```

```
MSSubClass       0.000000
MSZoning         0.000000
LotFrontage      0.000000
LotArea          0.000000
Street           0.000000
LotShape         0.000000
LandContour      0.000000
Utilities        0.000000
LotConfig        0.000000
LandSlope        0.000000
Neighborhood     0.000000
Condition1       0.000000
Condition2       0.000000
BldgType         0.000000
HouseStyle       0.000000
OverallQual      0.000000
OverallCond      0.000000
YearBuilt        0.000000
YearRemodAdd     0.000000
RoofStyle        0.000000
RoofMatl         0.000000
Exterior1st      0.000000
Exterior2nd      0.000000
MasVnrType       0.000000
MasVnrArea       0.597385
ExterQual        0.000000
ExterCond        0.000000
Foundation       0.000000
BsmtQual         0.000000
BsmtCond         0.000000
BsmtExposure     0.000000
BsmtFinType1     0.000000
BsmtFinSF1       0.320716
BsmtFinType2     0.000000
BsmtFinSF2       0.886442
BsmtUnfSF        0.081211
TotalBsmtSF      0.025465
Heating          0.000000
HeatingQC        0.000000
CentralAir       0.000000
Electrical       0.000000
1stFlrSF         0.000000
2ndFlrSF         0.567103
LowQualFinSF     0.982106
GrLivArea        0.000000
BsmtFullBath     0.588438
BsmtHalfBath     0.943565
FullBath         0.006194
HalfBath         0.625602
BedroomAbvGr     0.004129
KitchenAbvGr     0.000688
KitchenQual      0.000000
TotRmsAbvGrd     0.000000
Functional       0.000000
Fireplaces       0.474880
GarageType       0.000000
GarageYrBlt      0.000000
GarageFinish     0.000000
GarageCars       0.055747
GarageArea       0.055747
```

```
GarageQual       0.000000
GarageCond       0.000000
PavedDrive       0.000000
WoodDeckSF       0.521679
OpenPorchSF      0.450103
EnclosedPorch    0.857536
3SsnPorch        0.983482
ScreenPorch      0.920853
PoolArea         0.995871
MiscVal          0.964212
MoSold           0.000000
YrSold           0.000000
SaleType         0.000000
SaleCondition    0.000000
SalePrice        0.000000
dtype: float64
Index(['LowQualFinSF', '3SsnPorch', 'PoolArea'], dtype='object')
```

In [45]:
```
print(coldrop) # I WILL DROP ABOVE 3  COLUMNS BECAUSE IT HAVING LARGE NUMBER OF ZEROS
# ALSO MORE NUMBER OF ZEROS NOT PRODUCING GOOD RESULT
```

```
Index(['LowQualFinSF', '3SsnPorch', 'PoolArea'], dtype='object')
```

In [46]:
```
df.drop(columns=coldrop,axis=1, inplace=True) # delete above 3 columns
```

In [47]:
```
df.shape # LowQualFinSF', '3SsnPorch', 'PoolArea columns dropped more number of zros r
```

Out[47]:
```
(1453, 72)
```

4.feature Engineering :

- first do addition of columns together : reduce the complexity. then do encoding
- adding some columns to form new column .

In [54]:
```
df[['GarageYrBlt','YearBuilt','YearRemodAdd','MoSold','YrSold']].describe()
```

Out[54]:

	GarageYrBlt	YearBuilt	YearRemodAdd	MoSold	YrSold
count	1453.000000	1453.000000	1453.000000	1453.000000	1453.000000
mean	1978.484515	1971.138334	1984.789401	6.324157	2007.814866
std	23.993642	30.197139	20.649701	2.700413	1.329678
min	1900.000000	1872.000000	1950.000000	1.000000	2006.000000
25%	1962.000000	1954.000000	1966.000000	5.000000	2007.000000
50%	1980.000000	1972.000000	1994.000000	6.000000	2008.000000
75%	2001.000000	2000.000000	2004.000000	8.000000	2009.000000
max	2010.000000	2010.000000	2010.000000	12.000000	2010.000000

In [48]:
```
# Convert year related columns to number of years, to find how old the house is, or ho
# house was sold.
# there 4 date time features -YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
```

```
df['HouseAge'] = df['YrSold'] - df['YearBuilt']

df['YearRemodel'] = 2023 - df['YearRemodAdd']
df['GarageAge'] = 2023 - df['GarageYrBlt']
df['YrSold'] = 2023 - df['YrSold']

#df.drop(['GarageYrBlt'], axis=1, inplace=True)
```

In [51]:
```
df.drop(columns={'YearBuilt','YearRemodAdd','GarageYrBlt'},axis=1,inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_924\946123788.py in <module>
----> 1 df.drop(columns={'YearBuilt','YearRemodAdd','GarageYrBlt'},axis=1,inplace=Tru
e)

~\anaconda3\lib\site-packages\pandas\util\_decorators.py in wrapper(*args, **kwargs)
    309                     stacklevel=stacklevel,
    310                 )
--> 311             return func(*args, **kwargs)
    312
    313         return wrapper

~\anaconda3\lib\site-packages\pandas\core\frame.py in drop(self, labels, axis, index,
columns, level, inplace, errors)
   4955                 weight  1.0     0.8
   4956         """
-> 4957         return super().drop(
   4958             labels=labels,
   4959             axis=axis,

~\anaconda3\lib\site-packages\pandas\core\generic.py in drop(self, labels, axis, inde
x, columns, level, inplace, errors)
   4265         for axis, labels in axes.items():
   4266             if labels is not None:
-> 4267                 obj = obj._drop_axis(labels, axis, level=level, errors=error
s)
   4268
   4269         if inplace:

~\anaconda3\lib\site-packages\pandas\core\generic.py in _drop_axis(self, labels, axi
s, level, errors, consolidate, only_slice)
   4309                 new_axis = axis.drop(labels, level=level, errors=errors)
   4310             else:
-> 4311                 new_axis = axis.drop(labels, errors=errors)
   4312             indexer = axis.get_indexer(new_axis)
   4313

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in drop(self, labels, error
s)
   6659         if mask.any():
   6660             if errors != "ignore":
-> 6661                 raise KeyError(f"{list(labels[mask])} not found in axis")
   6662             indexer = indexer[~mask]
   6663         return self.delete(indexer)

KeyError: "['YearBuilt', 'GarageYrBlt', 'YearRemodAdd'] not found in axis"
```

In [52]:
```
df.shape
```

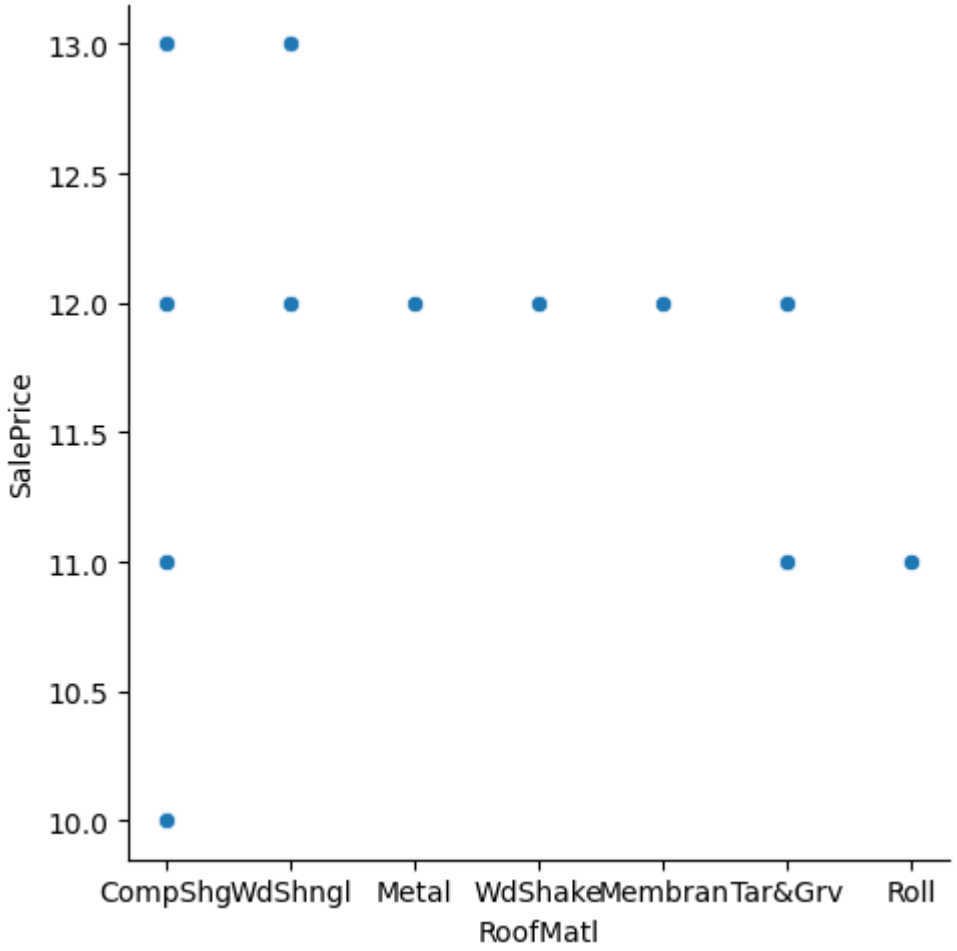
Out[52]:
```
(1453, 72)
```

In [53]:
```
# reducing columns numbers and creating one column from many columns.
df['TotalSF'] = df['TotalBsmtSF']+df['1stFlrSF']+df['2ndFlrSF']
df=df.drop(columns={'TotalBsmtSF','1stFlrSF', '2ndFlrSF'})

df['Bsmt'] = df['BsmtFinSF1']+ df['BsmtFinSF2']
df = df.drop(columns={'BsmtFinSF1','BsmtFinSF2'})

df['TotalBathroom'] = df['FullBath'] +(0.5)*df['HalfBath']
df = df.drop(columns={'FullBath','HalfBath'})

df['BsmtBath'] = df['BsmtFullBath']+(0.5)*df['BsmtHalfBath'] # basement bathrooms adde
df=df.drop(columns={'BsmtFullBath', 'BsmtHalfBath'})


df['TotalPorch'] =df['EnclosedPorch']+df['OpenPorchSF']+df['ScreenPorch']+df['WoodDeck
df=df.drop(columns={'EnclosedPorch','OpenPorchSF','ScreenPorch','WoodDeckSF'})


# add bedroom and kitchen making total rooms
# no need to add  TotRmsAbvGrd this feature is there so no need to separate bvedrooms
#df['TotRooms'] = df['BedroomAbvGr']+df['KitchenAbvGr'] # basement bathrooms added.
# df=df.drop(columns={'BedroomAbvGr','KitchenAbvGr'})# BedroomAbvG , KitchenAbvGr
```

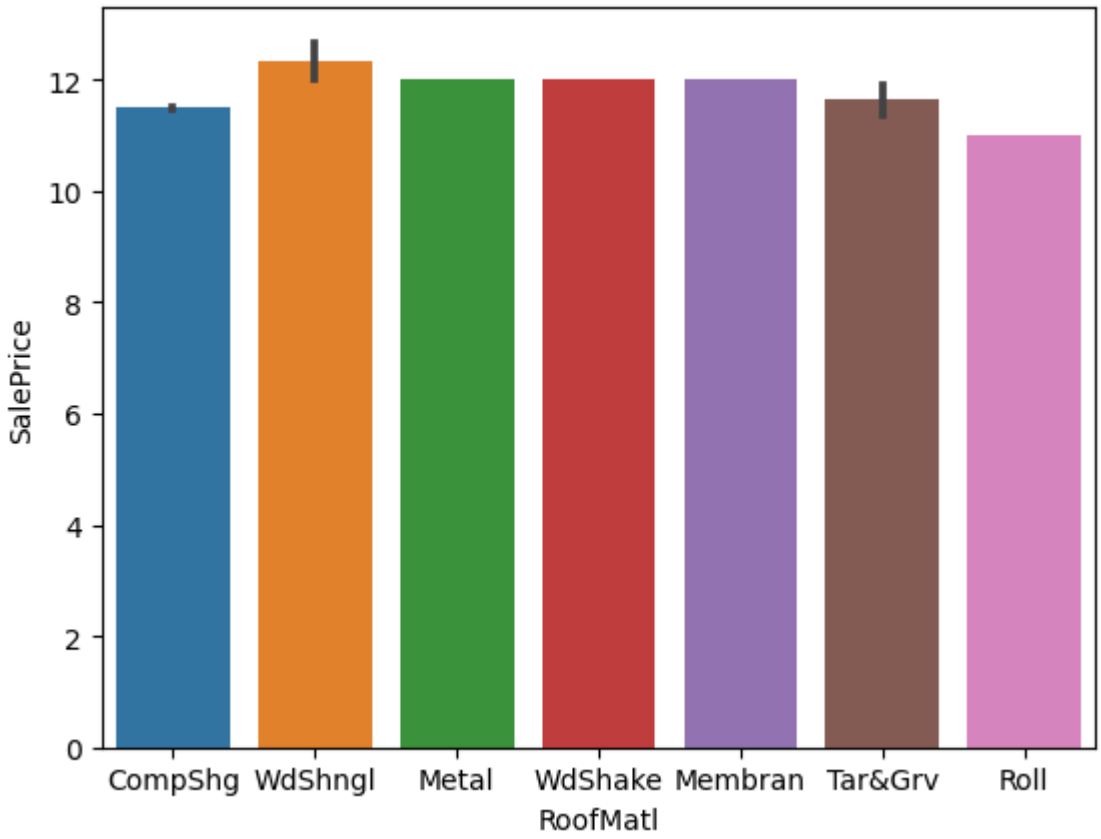
In [59]:
```
df.shape
```

Out[59]:
```
(1453, 64)
```

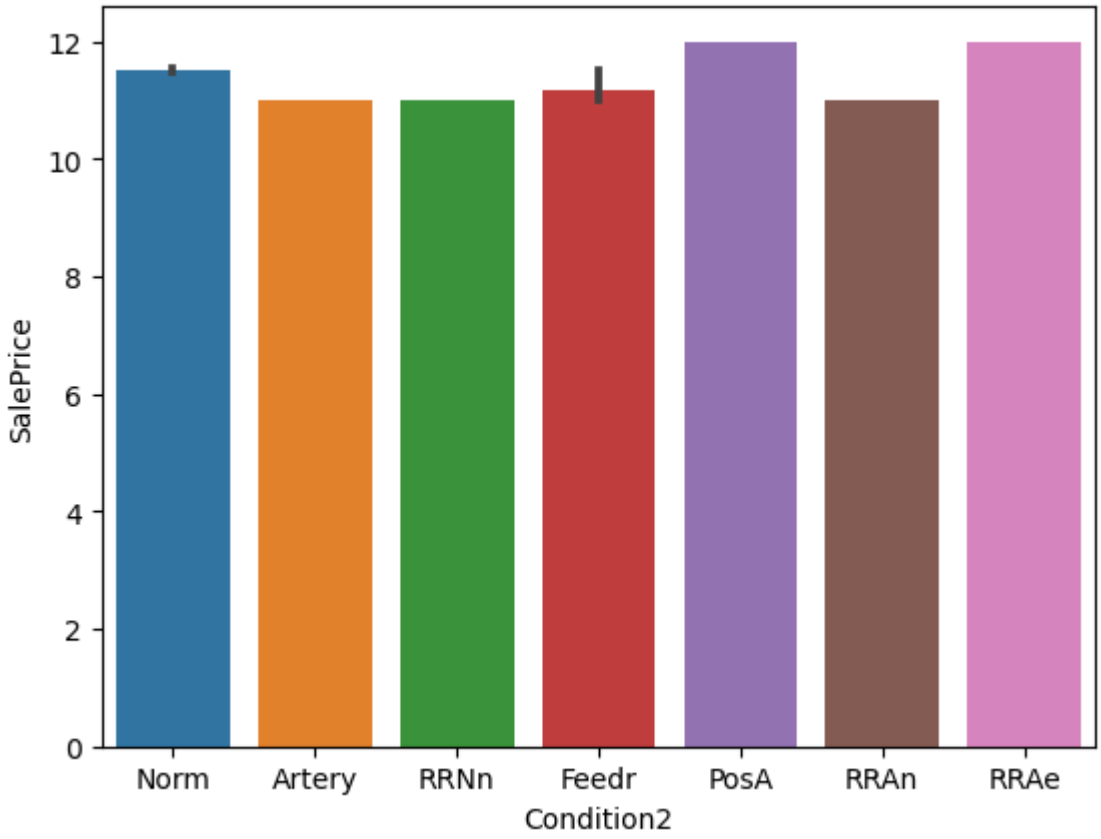
In [72]:
```
sns.relplot(data=df, x='RoofMatl', y="SalePrice")
plt.show() # no relationship between RoofMatalt with saleprice
```



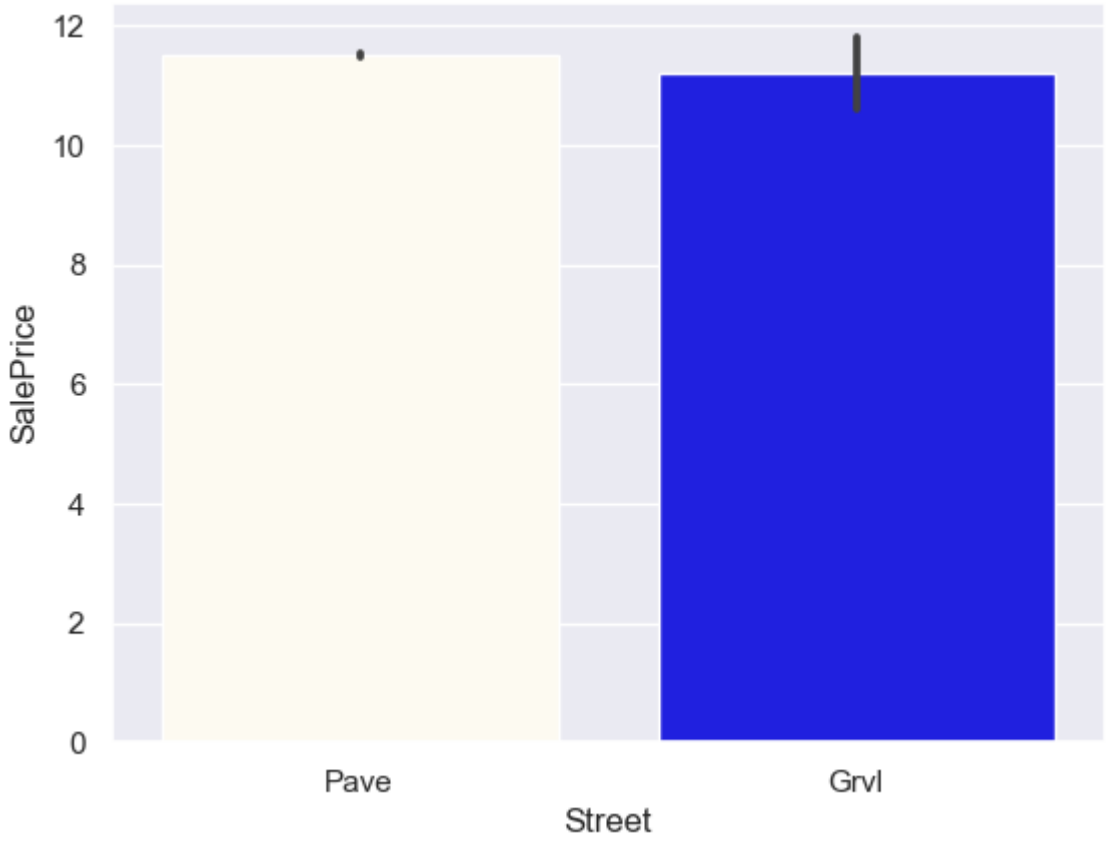
In [73]:
```
sns.barplot(data=df, x='RoofMatl', y="SalePrice")
plt.show()
```



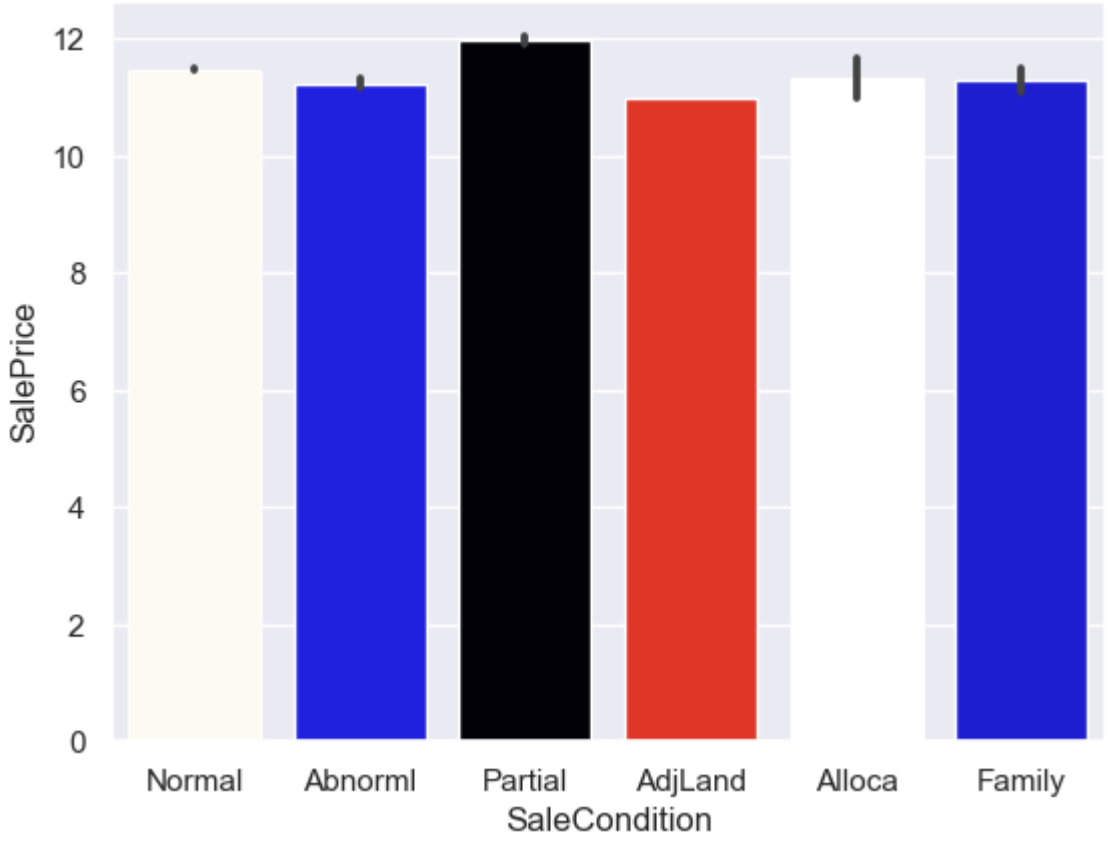
In [74]:
```
sns.barplot(data=df, x='Condition2', y="SalePrice")
plt.show()
```



In [106…
```
sns.barplot(data=df, x='Street', y="SalePrice")
plt.show()
# only pave strret more  and no relationship with sale price
```



In [103…
```
sns.barplot(x='SaleCondition',y='SalePrice',data=df)
plt.show()
# no relationship
```



In [54]:
```
df['Utilities'].value_counts()
```

Out[54]:
```
AllPub    1452
NoSeWa       1
Name: Utilities, dtype: int64
```

In [55]:
```
df.drop(columns='Utilities',inplace=True,axis=1)
print('Drop Utilities \n')
'''# this categorical Features with >95% of the same value
# these features totaly nosewa having only one value so it is no use for category '''
# why drop this Utilities column -because this column contain only one category Allpub
# in most cases  we drop it ,because it not not giving useful information for output
```

```
Drop Utilities
```

Out[55]:
```
'# this categorical Features with >95% of the same value\n# these features totaly nos
ewa having only one value so it is no use for category '
```

In [56]:
```
df.drop(columns=['MiscVal','OverallCond','GarageCars','BedroomAbvGr','KitchenAbvGr','B
                 'RoofMatl' ],inplace=True,axis=1)

print('Drop MiscVal ,OverallCond,GarageCars,BedroomAbvGr,KitchenAbvGr, BsmtQual,Garage
# 10 columns : dropped from numerical as well as categorical columns using describe ()
# RoofMatl - constant value


# we're going to drop condition_2 because it will skew our data too much,ALSO TOP VALU
# LIKE UTILITIES ,STREET FEATURE WE CAN DROP HERE. Condition2
print(df.shape)
df.drop(['Condition2'], axis=1, inplace=True)
print(df.shape)
```

```
Drop MiscVal ,OverallCond,GarageCars,BedroomAbvGr,KitchenAbvGr, BsmtQual,GarageQual,
Street, Heating,RoofMatl

(1453, 53)
(1453, 52)
```

from above poolarea , MiscVal,screenporch , enclosedporch ,openporch,ScreenPorch - contains large number of zeros so i decided to drop these columns.or adding all zeros columns make new columns # garage cars or garage area same ,means garage cars not affecting the prize. if garage area is more according to that we can accomodate number of cars . # # 'BedroomAbvGr','KitchenAbvGr' should drop because totalroomsgr showing same property . # OverallQual: Overall material and finish quality OverallCond: Overall condition rating drop 1 add - , garage cars drop # categorical columns BsmtQual & BsmtCond both shows same categories , both highly correlated with each other so drop any 1 . # same for GarageQual& GarageCond same property so drop any 1. # DROP -'Street' column because this column contain Pave category more it constant value 6 value of grvl street. so i drop it

5. FEATURE ENCODING

- We need to convert these categorical variables to numbers such that the model is able to understand and extract valuable information.
  - Two types of categorical data
  - Ordinal Data: The categories have an inherent order
- Nominal Data: The categories do not have an inherent order

###ENCODING : Categorical columns into numerical columns. - PERFORME LABEL CODING ON LESS CATEGORY COLUMNS OR MAP WITH O AND 1 VALUE - Perform ordinal encoding only on - ordinal columns - frequency encoding - when large no. of categories is there no ordinal only nominal values are there : # we can perform ColumnTranformer class - for scaling encoding numerical as well as categorical colum and to fit tranform # r

In [57]:
```
df_ = df.select_dtypes(exclude=['int', 'float']) # excluding numerical columns
for x in range(0,34):   # 0 to 34 categorical columns we have to display  here x is ir
    print ("---- {} ---".format(df_.columns[x]))
    print("Null count:",df_.iloc[:,x].isnull().sum()) # also print null values are not
    print(df_.iloc[:,x].value_counts(dropna=False).sort_index(ascending=True))
    print()
```

```
---- MSZoning ---
Null count: 0
C (all)      10
FV           65
RH           16
RL         1144
RM          218
Name: MSZoning, dtype: int64

---- LotShape ---
Null count: 0
IR1    481
IR2     41
IR3      9
Reg    922
Name: LotShape, dtype: int64

---- LandContour ---
Null count: 0
Bnk      61
HLS      50
Low      35
Lvl    1307
Name: LandContour, dtype: int64

---- LotConfig ---
Null count: 0
Corner      262
CulDSac      94
FR2          47
FR3           4
Inside     1046
Name: LotConfig, dtype: int64

---- LandSlope ---
Null count: 0
Gtl    1376
Mod      64
Sev      13
Name: LandSlope, dtype: int64

---- Neighborhood ---
Null count: 0
Blmngtn     17
Blueste      2
BrDale      16
BrkSide     58
ClearCr     28
CollgCr    150
Crawfor     51
Edwards     98
Gilbert     79
IDOTRR      37
MeadowV     17
Mitchel     48
NAmes      225
NPkVill      9
NWAmes      73
NoRidge     40
NridgHt     74
```

```
OldTown    113
SWISU       25
Sawyer      74
SawyerW     59
Somerst     86
StoneBr     25
Timber      38
Veenker     11
Name: Neighborhood, dtype: int64

---- Condition1 ---
Null count: 0
Artery      48
Feedr       80
Norm      1255
PosA         8
PosN        18
RRAe        11
RRAn        26
RRNe         2
RRNn         5
Name: Condition1, dtype: int64

---- BldgType ---
Null count: 0
1Fam      1214
2fmCon      30
Duplex      52
Twnhs       43
TwnhsE     114
Name: BldgType, dtype: int64

---- HouseStyle ---
Null count: 0
1.5Fin    154
1.5Unf     14
1Story    721
2.5Fin      8
2.5Unf     11
2Story    443
SFoyer     37
SLvl       65
Name: HouseStyle, dtype: int64

---- RoofStyle ---
Null count: 0
Flat         13
Gable      1139
Gambrel      11
Hip         281
Mansard       7
Shed          2
Name: RoofStyle, dtype: int64

---- Exterior1st ---
Null count: 0
AsbShng     20
AsphShn      1
BrkComm      2
BrkFace     49
```

```
CBlock       1
CemntBd     60
HdBoard    222
ImStucc      1
MetalSd    219
Plywood    108
Stone        2
Stucco      24
VinylSd    512
Wd Sdng    206
WdShing     26
Name: Exterior1st, dtype: int64

---- Exterior2nd ---
Null count: 0
AsbShng     20
AsphShn      3
Brk Cmn      7
BrkFace     24
CBlock       1
CmentBd     59
HdBoard    207
ImStucc     10
MetalSd    213
Other        1
Plywood    142
Stone        5
Stucco      25
VinylSd    501
Wd Sdng    197
Wd Shng     38
Name: Exterior2nd, dtype: int64

---- MasVnrType ---
Null count: 0
BrkCmn      15
BrkFace    443
None       871
Stone      124
Name: MasVnrType, dtype: int64

---- ExterQual ---
Null count: 0
Ex     48
Fa     14
Gd    486
TA    905
Name: ExterQual, dtype: int64

---- ExterCond ---
Null count: 0
Ex       3
Fa      28
Gd     145
Po       1
TA    1276
Name: ExterCond, dtype: int64

---- Foundation ---
Null count: 0
```

```
BrkTil    146
CBlock    633
PConc     641
Slab       24
Stone       6
Wood        3
Name: Foundation, dtype: int64

---- BsmtCond ---
Null count: 0
Fa      45
Gd      65
Po       2
TA    1341
Name: BsmtCond, dtype: int64

---- BsmtExposure ---
Null count: 0
Av    220
Gd    131
Mn    114
No    988
Name: BsmtExposure, dtype: int64

---- BsmtFinType1 ---
Null count: 0
ALQ    220
BLQ    147
GLQ    413
LwQ     74
Rec    133
Unf    466
Name: BsmtFinType1, dtype: int64

---- BsmtFinType2 ---
Null count: 0
ALQ      18
BLQ      33
GLQ      14
LwQ      46
Rec      54
Unf    1288
Name: BsmtFinType2, dtype: int64

---- HeatingQC ---
Null count: 0
Ex    734
Fa     49
Gd    241
Po      1
TA    428
Name: HeatingQC, dtype: int64

---- CentralAir ---
Null count: 0
N      95
Y    1358
Name: CentralAir, dtype: int64

---- Electrical ---
```

```
Null count: 0
FuseA      94
FuseF      27
FuseP       3
Mix         1
SBrkr    1328
Name: Electrical, dtype: int64

---- KitchenQual ---
Null count: 0
Ex     96
Fa     39
Gd    584
TA    734
Name: KitchenQual, dtype: int64

---- Functional ---
Null count: 0
Maj1      14
Maj2       5
Min1      31
Min2      34
Mod       15
Sev        1
Typ     1353
Name: Functional, dtype: int64

---- GarageType ---
Null count: 0
2Types       5
Attchd     946
Basment     19
BuiltIn     87
CarPort      9
Detchd     387
Name: GarageType, dtype: int64

---- GarageFinish ---
Null count: 0
Fin    348
RFn    419
Unf    686
Name: GarageFinish, dtype: int64

---- GarageCond ---
Null count: 0
Ex       2
Fa      35
Gd       9
Po       7
TA    1400
Name: GarageCond, dtype: int64

---- PavedDrive ---
Null count: 0
N      90
P      30
Y    1333
Name: PavedDrive, dtype: int64
```

```
---- SaleType ---
Null count: 0
COD        43
CWD         4
Con         2
ConLD       9
ConLI       5
ConLw       5
New       119
Oth         3
WD       1263
Name: SaleType, dtype: int64

---- SaleCondition ---
Null count: 0
Abnorml     101
AdjLand       4
Alloca       12
Family       20
Normal     1194
Partial     122
Name: SaleCondition, dtype: int64
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_924\2808675860.py in <module>
      1 df_ = df.select_dtypes(exclude=['int', 'float']) # excluding numerical column
s
      2 for x in range(0,34):   # 0 to 34 categorical columns we have to display  her
e x is index
----> 3     print ("---- {} ---".format(df_.columns[x]))
      4     print("Null count:",df_.iloc[:,x].isnull().sum()) # also print null value
s are not.
      5     print(df_.iloc[:,x].value_counts(dropna=False).sort_index(ascending=Tru
e))

~\anaconda3\lib\site-packages\pandas\core\indexes\base.py in __getitem__(self, key)
   5051             # GH#44051 exclude bool, which would return a 2d ndarray
   5052             key = com.cast_scalar_indexer(key, warn_float=True)
-> 5053             return getitem(key)
   5054
   5055         if isinstance(key, slice):

IndexError: index 31 is out of bounds for axis 0 with size 31
```

In [59]:
```
from sklearn.preprocessing import LabelEncoder
```

Transform Numerical Variables to Categorical Variables:

In [60]:
```
num_cat = ['MSSubClass']

for col in num_cat:
    df[col] = df[col].apply(str)
```

Because I have calculated age of houses, YearBuilt is no longer needed. However, YrSold could have a large impact on house price Therefore, I will transform it into categorical variables. Like YrSold, some numerical variables don't have any ordinal meaning (e.g. MSSubClass). I will transform them into categorical variables.

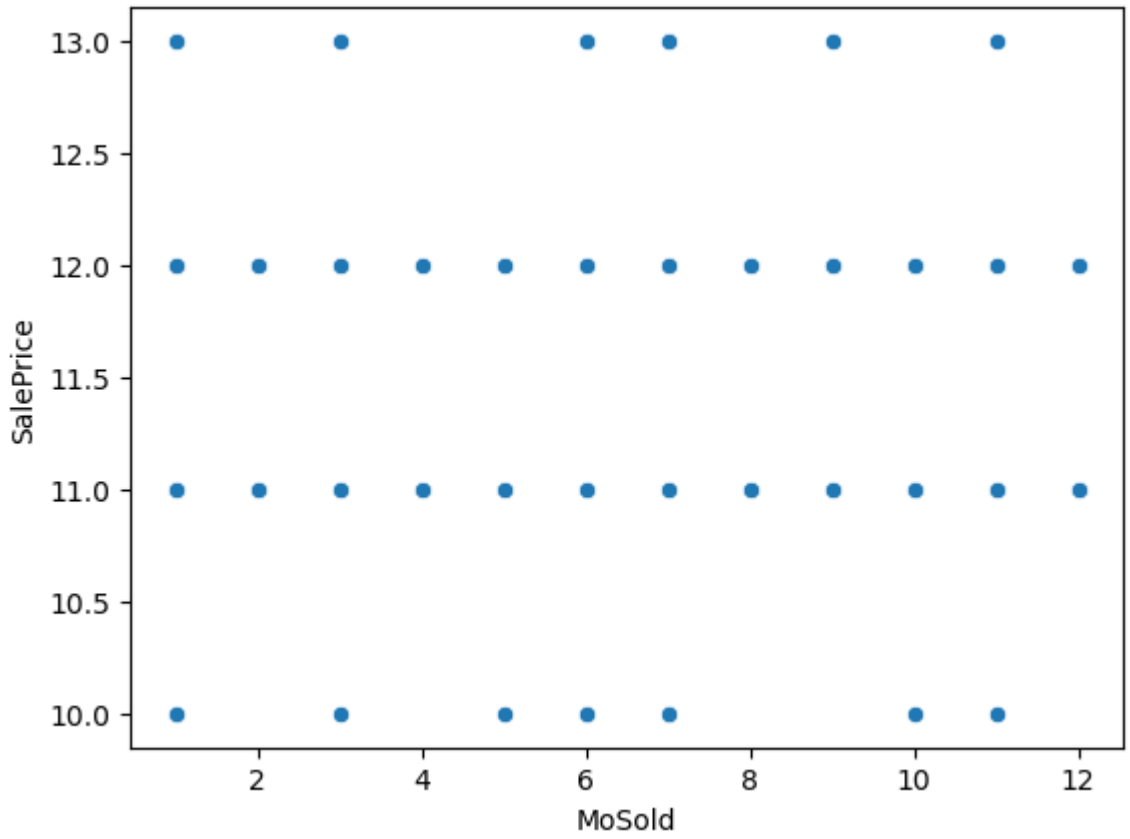
In [88]:
```
df.describe().T
```

Out[88]:

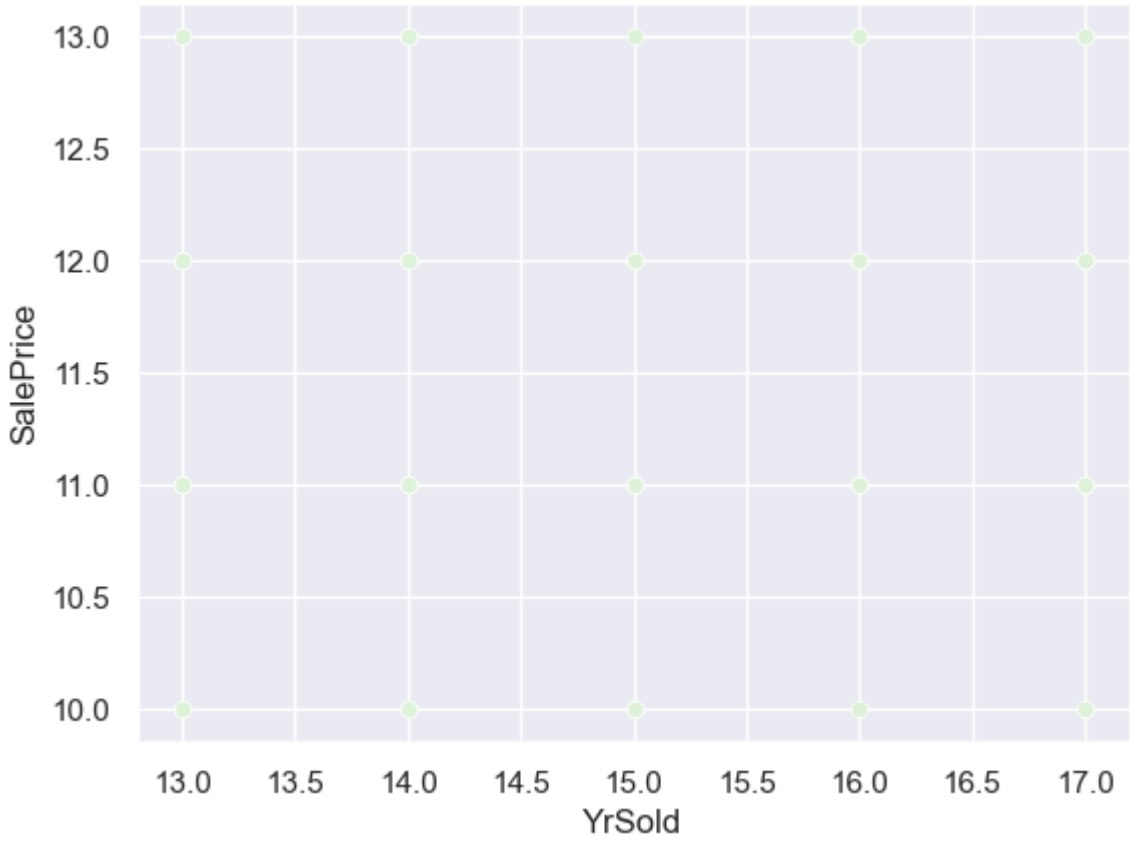
	count	mean	std	min	25%	50%	75%
LotFrontage	1453.0	69.055747	12.917424	49.000000	60.000000	69.000000	79.000000
LotArea	1453.0	9555.064831	2811.326155	5000.000000	7535.000000	9453.000000	11553.000000
OverallQual	1453.0	6.088782	1.371077	1.000000	5.000000	6.000000	7.000000
MasVnrArea	1453.0	101.887130	179.223197	0.000000	0.000000	0.000000	162.000000
BsmtUnfSF	1453.0	564.580179	438.550601	0.000000	223.000000	474.000000	806.000000
GrLivArea	1453.0	7.264687	0.329820	5.811141	7.028201	7.284821	7.481556
TotRmsAbvGrd	1453.0	6.504474	1.612883	2.000000	5.000000	6.000000	7.000000
Fireplaces	1453.0	0.609085	0.641731	0.000000	0.000000	1.000000	1.000000
GarageArea	1453.0	470.340674	209.602757	0.000000	328.000000	478.000000	576.000000
MoSold	1453.0	6.324157	2.700413	1.000000	5.000000	6.000000	8.000000
YrSold	1453.0	15.185134	1.329678	13.000000	14.000000	15.000000	16.000000
SalePrice	1453.0	11.503097	0.533477	10.000000	11.000000	12.000000	12.000000
HouseAge	1453.0	51.861666	30.197139	13.000000	23.000000	51.000000	69.000000
YearRemodel	1453.0	38.210599	20.649701	13.000000	19.000000	29.000000	57.000000
GarageAge	1453.0	44.515485	23.993642	13.000000	22.000000	43.000000	61.000000
TotalSF	1453.0	2549.629043	761.405707	334.000000	2008.000000	2470.000000	2999.000000
Bsmt	1453.0	482.775637	450.873139	0.000000	0.000000	464.000000	788.000000
TotalBathroom	1453.0	1.753613	0.634083	0.000000	1.000000	2.000000	2.500000
BsmtBath	1453.0	0.451480	0.513475	0.000000	0.000000	0.000000	1.000000
TotalPorch	1453.0	177.041982	154.514347	0.000000	44.000000	160.000000	262.000000

df.describe(include='O').T

In [61]:
```
# no use of month sold
sns.scatterplot(x="MoSold",y="SalePrice",data=df)
plt.show()# no relationship between MoSold and SalePrice every mothe sale price is sam
```



In [91]:
```
sns.scatterplot(x="YrSold",y="SalePrice",data=df)
plt.show()
# every year house saled with prize no realtionship . so i drop it.
```



In [62]:
```
df.drop(['MoSold'], axis=1, inplace=True)
df.drop(['YrSold'], axis=1, inplace=True)
print(df.shape)
''' no relationship with sale price so we dropped it.'''
```

Out[62]:
```
(1453, 50)
' no relationship with sale price so we dropped it.'
```

In [63]:
```
##identifying categorical variables
categorical2 = df.select_dtypes(include=['object'])
categorical2 = categorical2.columns
```

In [70]:
```
len(categorical2)
```

Out[70]:
```
32
```

In [64]:
```
 from sklearn.preprocessing import  LabelEncoder # import LabelEncoder
```

In [65]:
```
# Ordinal categorical columns
label_encoding_cols = [
     "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2",
     "ExterCond", "ExterQual", "Functional","CentralAir",
     "HeatingQC", "KitchenQual", "LandSlope","Electrical",
    "LotShape", "PavedDrive","GarageFinish","GarageCond"
]

# Apply Label Encoder
label_encoder = LabelEncoder()

for col in label_encoding_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

In [74]:
```
# we can drop central air , heatingqc  because functional columns shows same property
# but applied later after scaling check the correaltion highest correalted featurec.

'''MSZoning','MsSubClass'  'LandContour', 'LotConfig',
       'Neighborhood', 'Condition1', 'BldgType', 'HouseStyle', 'RoofStyle',
       'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'Foundation', 'GarageType',  ,
       'SaleType', 'SaleCondition' '''
```

Out[74]:
```
"MSZoning','MsSubClass'  'LandContour', 'LotConfig',\n       'Neighborhood', 'Conditi
on1', 'BldgType', 'HouseStyle', 'RoofStyle',\n       'Exterior1st', 'Exterior2nd', 'M
asVnrType', \n       'Foundation', 'GarageType',  ,\n       'SaleType', 'SaleConditio
n' "
```

All above nominal features contains large categories more than 10 so using One Hot Encoding ,redudant columns are increases Next we move to perform Target Encoding. # In Target Encoding a categorical value is substituted with the mean value of the target variable for this specific categorical variable in the training set # We follow this approach as one-hot encoding would create a lot of extra columns and would increase considerably the computational cost of our model. We fit only on the training set in order not to have data leakage. For this method we would have also not other option as for the test set we do not know the target value. However, we could have used other options like Label Encoding and test the best one in a cross validation set. # Prefer ordinal coding or label encoding on some columns and target encoding on some remaning columns.

In [75]:
```
df.shape
```

Out[75]:
```
(1453, 50)
```

In [66]:
```
cat_cols_n = df.select_dtypes(include=['object']).columns
cat_cols_n
```

Out[66]:
```
Index(['MSSubClass', 'MSZoning', 'LandContour', 'LotConfig', 'Neighborhood',
       'Condition1', 'BldgType', 'HouseStyle', 'RoofStyle', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'Foundation', 'GarageType', 'SaleType',
       'SaleCondition'],
      dtype='object')
```

- Use Target Encoding :

In [67]:
```
#Target encoder :
import category_encoders as ce # target encoder
enc=ce.TargetEncoder()
```

In [69]:
```
df[cat_cols_n]=enc.fit_transform(df[cat_cols_n],df['SalePrice'])
''' Applied Target Encoding'''
```

Out[69]:
```
Warning: No categorical columns found. Calling 'transform' will only return input dat
a.
' Applied Target Encoding'
```

In [70]:
```
pd.concat([df,df[cat_cols_n]],axis=1)
```

Out[70]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LandContour	LotConfig	LandSlope
1	11.969697	11.576049	65.0	8450.0	3	11.503443	11.479924	0
2	11.479323	11.576049	80.0	9600.0	3	11.503443	11.530100	0
3	11.969697	11.576049	68.0	11250.0	0	11.503443	11.479924	0
4	11.401854	11.576049	60.0	9550.0	0	11.503443	11.503817	0
5	11.969697	11.576049	84.0	14226.4	0	11.503443	11.530100	0
...	...	...	...	...	...	...	...	...
1456	11.969697	11.576049	62.0	7917.0	3	11.503443	11.479924	0
1457	11.479323	11.576049	85.0	13175.0	3	11.503443	11.479924	0
1458	11.401854	11.576049	66.0	9042.0	3	11.503443	11.479924	0
1459	11.479323	11.576049	68.0	9717.0	3	11.503443	11.479924	0
1460	11.479323	11.576049	75.0	9937.0	3	11.503443	11.479924	0

1453 rows × 66 columns

In [71]:
```
# from above concat columns are duplicates columns so remove it.
df = df.drop_duplicates()
print( df.shape )
```

```
(1452, 50)
```

In [106…

```
df # final done  encoding
```

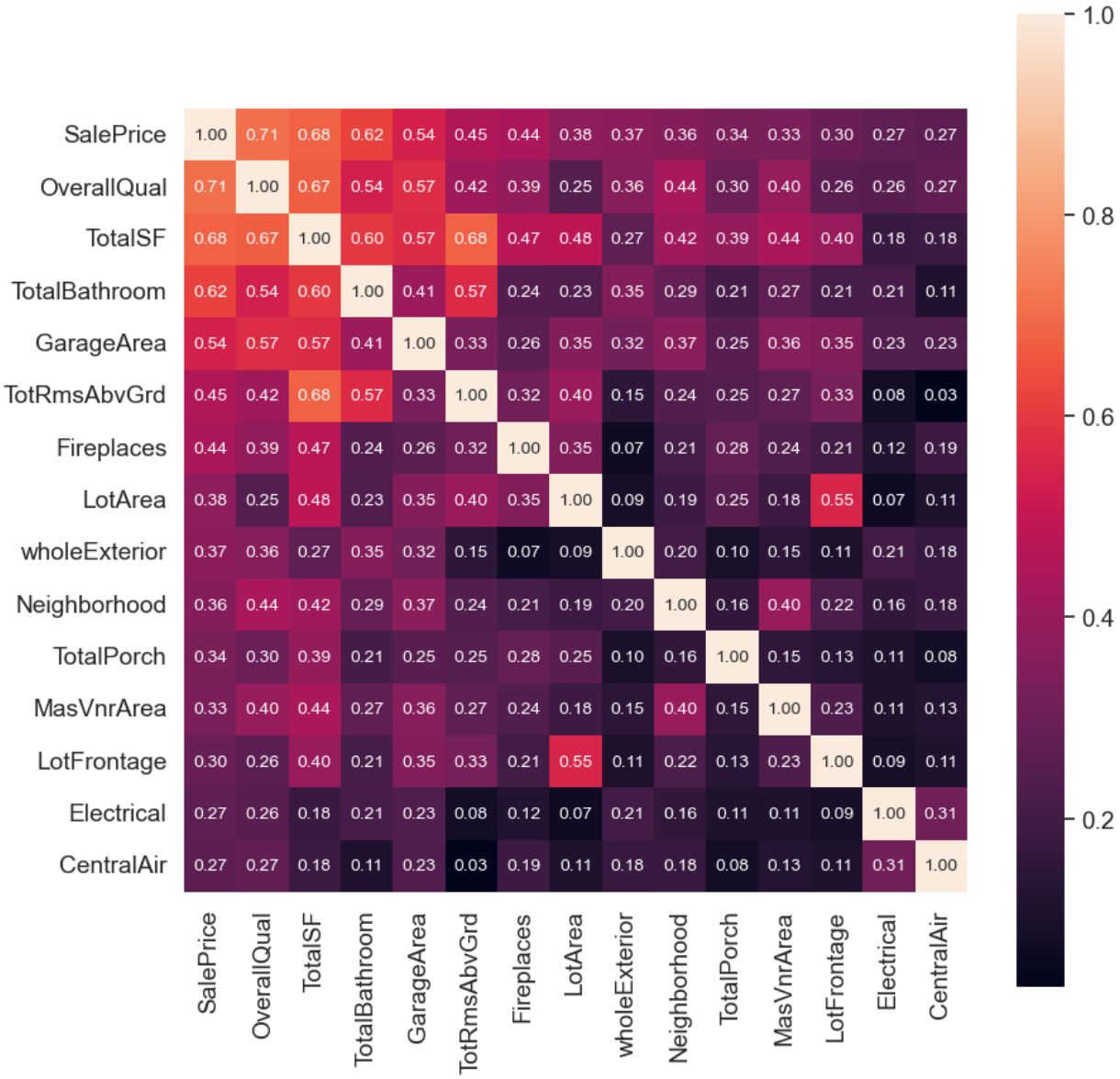
Out[106]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	LotShape	LandContour	LotConfig	LandSlope
1	11.969697	11.576049	65.0	8450.0	3	11.503443	11.479924	0
2	11.479323	11.576049	80.0	9600.0	3	11.503443	11.530100	0
3	11.969697	11.576049	68.0	11250.0	0	11.503443	11.479924	0
4	11.401854	11.576049	60.0	9550.0	0	11.503443	11.503817	0
5	11.969697	11.576049	84.0	14226.4	0	11.503443	11.530100	0
…	…	…	…	…	…	…	…	…
1456	11.969697	11.576049	62.0	7917.0	3	11.503443	11.479924	0
1457	11.479323	11.576049	85.0	13175.0	3	11.503443	11.479924	0
1458	11.401854	11.576049	66.0	9042.0	3	11.503443	11.479924	0
1459	11.479323	11.576049	68.0	9717.0	3	11.503443	11.479924	0
1460	11.479323	11.576049	75.0	9937.0	3	11.503443	11.479924	0

1452 rows × 50 columns

In [135…

```
# SOME COLUMNS ALL STASTICAL VALUES ARE ZEROS ,SO WE CAN DROP
#CHECK CORREALTATION :
cor = df.corr()
k = 15 #number of variables for heatmap
cols = cor.nlargest(k, 'SalePrice')['SalePrice'].index
cmt = np.corrcoef(df[cols].values.T) # correatiion matrix
sns.set(font_scale=1.25)
plt.figure(figsize=(10,10))
hm = sns.heatmap(cmt, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size'
plt.show()
print("affecting target varaible most : ", cols, len(cols))
```




```
affecting target varaible most :  Index(['SalePrice', 'OverallQual', 'TotalSF', 'Tota
lBathroom', 'GarageArea',
       'TotRmsAbvGrd', 'Fireplaces', 'LotArea', 'wholeExterior',
       'Neighborhood', 'TotalPorch', 'MasVnrArea', 'LotFrontage', 'Electrical',
       'CentralAir'],
      dtype='object') 15
```

In [72]:
```
# some columns are highly correalated with each other
'''TotalSF & GrLivArea ,first columns mostly dependent on saleprice so we drop GrLivAr
''' TotRmsAbvGrd  with GrLivArea- but i drop only GrLivArea'''
df.drop(['GrLivArea'], axis=1, inplace=True)
df.shape
```

Out[72]:
```
(1452, 49)
```

In [73]:
```
'


df.drop(['LandSlope'], axis=1, inplace=True)
df.shape
''' LandSlope, contains large numer of zeros as well stastical values almost zero .so
it is  not affecting output. '''
```

Out[73]:
```
(1452, 48)
```

In [74]:
```
# we can add important columns after encoding for reducing columns :

df['wholeExterior'] = df['Exterior1st']+df['Exterior2nd']
df=df.drop(columns={'Exterior1st','Exterior2nd'})

# BsmtFinType1 BsmtFinType2
df['Bsmtfintotal'] = df['BsmtFinType1']+df['BsmtFinType2']
df=df.drop(columns={'BsmtFinType1','BsmtFinType2'})
```

- convert insame dattypes:

In [75]:
```
df= df.astype('int64') # convert all datatypes into same type i.e int64
```

In [116…
```
 #now to check if everything is numerical
print(df.dtypes)
```

```
MSSubClass       int64
MSZoning         int64
LotFrontage      int64
LotArea          int64
LotShape         int64
LandContour      int64
LotConfig        int64
Neighborhood     int64
Condition1       int64
BldgType         int64
HouseStyle       int64
OverallQual      int64
RoofStyle        int64
MasVnrType       int64
MasVnrArea       int64
ExterQual        int64
ExterCond        int64
Foundation       int64
BsmtCond         int64
BsmtExposure     int64
BsmtUnfSF        int64
HeatingQC        int64
CentralAir       int64
Electrical       int64
KitchenQual      int64
TotRmsAbvGrd     int64
Functional       int64
Fireplaces       int64
GarageType       int64
GarageFinish     int64
GarageArea       int64
GarageCond       int64
PavedDrive       int64
SaleType         int64
SaleCondition    int64
SalePrice        int64
HouseAge         int64
YearRemodel      int64
GarageAge        int64
TotalSF          int64
Bsmt             int64
TotalBathroom    int64
BsmtBath         int64
TotalPorch       int64
wholeExterior    int64
Bsmtfintotal     int64
dtype: object
```

After encoding check skewness then apply log TRANSFORMATION.

In [76]:
```
print(df['LotArea'].skew(), df['MasVnrArea'].skew() ,df['BsmtUnfSF'].skew(),df['Garage
      df['TotalSF'].skew(), df['TotalPorch'].skew())
#not use of applying log beacuse generate many  nan values .so skip it. directly do sc
''' no need do log transformation directly do scaling '''
```

```
0.0687263952195317 2.702414295552311 0.9086598369978268 0.04554058557321854 0.7236219
960520385 1.1358860589214363
```

Out[76]:
```
' no need do log transformation directly do scaling '
```

Split the data ,then do scaling data only on input vcariables. then apply regression model

In [77]:
```
# split train and test set
from sklearn.model_selection import train_test_split
# import the sklearn libraries

from sklearn.preprocessing import RobustScaler # Robust scaler ()
from sklearn.linear_model import LinearRegression, Lasso, Ridge # Linear regression mo
from sklearn.ensemble import RandomForestRegressor

import xgboost as xgb
from sklearn import ensemble
from sklearn.ensemble import RandomForestRegressor

# calculatae error :
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import  r2_score # check score for evaluation
```

In [78]:
```
#Sepearate depdent and indepedent features
X=df.drop('SalePrice',axis=1) # indepdent features
y=df['SalePrice']  # Dependent features.
X.shape # 1452 rows and 45 columns
```

Out[78]:
```
(1452, 45)
```

In [79]:
```
y.shape
```

Out[79]:
```
(1452,)
```

Scaling : Robust scaler() or Satandard scaler ()

RobustScaler is a transformation technique that removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). It is also robust to outliers, which makes it ideal for data where there are too many outliers that will drastically reduce the number of training data.

In [91]:
```
# Initialize the Robust Scaler
scaler = RobustScaler()

#Fit and transform the training features using the Robust Scaler
X_scaled = scaler.fit_transform(X)
'''Scaled all indepdent features then split the data'''
```

Out[91]:
```
'Scaled all indepdent features then split the data'
```

In [85]:
```
X_scaled.shape
```

Out[85]:
```
(1452, 45)
```

4.Feature Selection : RFE

In [86]:
```
from sklearn.feature_selection import RFE
```

In [87]:
```
# Feature Selection using Linear Regression
model = LinearRegression()
rfe = RFE(model, n_features_to_select=15)  # Adjust the number of features as needed
X_selected = rfe.fit_transform(X_scaled, y)

selected_feature_names = np.array(X.columns)[rfe.support_]
```

In [88]:
```
selected_feature_names
```

Out[88]:
```
array(['MSSubClass', 'LotArea', 'LotShape', 'Neighborhood', 'OverallQual',
       'BsmtUnfSF', 'HeatingQC', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageFinish', 'HouseAge', 'YearRemodel', 'TotalSF', 'Bsmt',
       'TotalBathroom'], dtype=object)
```

In [89]:
```
print("length of SELECTED RFE FEATURES:",len(selected_feature_names ))
```

```
length of SELECTED RFE FEATURES: 15
```

Split the dataset into test and train samples

In [92]:
```
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, rand
''' Split the dataset into training and testing sets'''
```

Out[92]:
```
' Split the dataset into training and testing sets'
```

5. MODEL Creation : Apply regression models

- Linear Regressoin ,lasso, ridge regressor
- Random forest regressor, xgboostregressor
- Ensemble all model together for simplification
- Evaluation model - calculate errors, and r2 score etc
- Select best model for hyperparameter tunig

- Create model with hypertuning -GridSearchCV
- except Linear Regression

In [94]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
```

In [95]:
```
# Hyperparameter Tuning for Lasso Regression
lasso = Lasso()
param_grid_lasso = {
    'alpha': [0.01, 0.1, 1.0, 10.0]
}
grid_search_lasso = GridSearchCV(estimator=lasso, param_grid=param_grid_lasso, cv=5)

grid_search_lasso.fit(X_train, y_train) # already x train scaled before spliting the d

best_lasso = grid_search_lasso.best_estimator_
```

In [97]:
```
#Hyperparameter Tuning for Ridge Regression
ridge = Ridge()
param_grid_ridge = {
    'alpha': [0.01, 0.1, 1.0, 10.0]
}
grid_search_ridge = GridSearchCV(estimator=ridge, param_grid=param_grid_ridge, cv=5)

grid_search_ridge.fit(X_train, y_train) # already x train scaled before spliting the d
best_ridge = grid_search_ridge.best_estimator_
```

In [98]:
```
# Hyperparameter Tuning for Random Forest
rf = RandomForestRegressor()
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20]
}
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5)
grid_search_rf.fit(X_train, y_train)
best_rf = grid_search_rf.best_estimator_
```

In [100…
```
from xgboost import XGBRegressor
```

In [101…
```
#Hyperparameter Tuning for XGBoost Regression
xgb = XGBRegressor()
param_grid_xgb = {
    'learning_rate': [0.01, 0.1, 0.3],
    'max_depth': [3, 5, 7],
    'n_estimators': [100, 200, 300]
}
grid_search_xgb = GridSearchCV(estimator=xgb, param_grid=param_grid_xgb, cv=5)
grid_search_xgb.fit(X_train, y_train)
best_xgb = grid_search_xgb.best_estimator_
```

6. Evaluate the models on the test set

In [102…
```
y_pred_lasso = best_lasso.predict(X_test) # predict lasso
y_pred_ridge = best_ridge.predict(X_test) # predict ridge
y_pred_xgb = best_xgb.predict(X_test) # predict xgb
y_pred_rf = best_rf.predict(X_test)# predict rf
```

- check r2 score

In [103…
```
#R2 score of each model :
r2_lasso = r2_score(y_test, y_pred_lasso)
r2_ridge = r2_score(y_test, y_pred_ridge)
r2_xgb = r2_score(y_test, y_pred_xgb)
r2_rf = r2_score(y_test, y_pred_rf)
```

In [104…
```
print("R2 Score - Random Forest:", r2_rf)
print("R2 Score - Lasso Regression:", r2_lasso)
print("R2 Score - Ridge Regression:", r2_ridge)
print("R2 Score - XGBoost Regression:", r2_xgb)
```

```
R2 Score - Random Forest: 0.7239717474783685
R2 Score - Lasso Regression: 0.688290117027986
R2 Score - Ridge Regression: 0.69647104119459
R2 Score - XGBoost Regression: 0.7307757701390384
```

- Loss function caluculation - MSE,RMSE,MAE

In [105…
```
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
rmse_lasso = np.sqrt(mse_lasso)
mae_lasso = mean_absolute_error(y_test, y_pred_lasso)
print("Lasso Regression LOSS:")
print(f" mse_lasso(MSE): {mse_lasso:.2f}")
print(f" rmse_lasso (RMSE): {rmse_lasso:.2f}")
print(f" mae_lasso (MAE): {mae_lasso:.2f}")
```

```
Lasso Regression LOSS:
 mse_lasso(MSE): 0.09
 rmse_lasso (RMSE): 0.29
 mae_lasso (MAE): 0.23
```

In [109…
```
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
rmse_ridge = np.sqrt(mse_ridge)
mae_ridge = mean_absolute_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print("Ridge Regression loss:")
print(f" mse_ridge(MSE): {mse_ridge:.2f}")
print(f" rmse_ridge (RMSE): {rmse_ridge:.2f}")
print(f" mae_ridge (MAE): {mae_ridge:.2f}")

print(f"r2_ridge (R2): {r2_ridge:.2f}")
```

```
Ridge Regression loss:
 mse_ridge(MSE): 0.08
 rmse_ridge (RMSE): 0.29
 mae_ridge (MAE): 0.22
r2_ridge (R2): 0.70
```

In [110…
```
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest loss :")
print(f"mse_rf (MSE): {mse_rf:.2f}")
print(f" rmse_rf_ridge (RMSE): {rmse_rf:.2f}")
print(f" mae_rf (MAE): {mae_rf:.2f}")

print(f"r2_rf (R2): {r2_rf:.2f}")
```

```
Random Forest loss :
mse_rf (MSE): 0.08
 rmse_rf_ridge (RMSE): 0.28
 mae_rf (MAE): 0.14
r2_rf (R2): 0.72
```

In [113…
```
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
rmse_xgb = np.sqrt(mse_xgb)
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
```

```
r2_xgb = r2_score(y_test, y_pred_xgb)
print("XGBoost model loss:")
print(f"mse_xgb (MSE): {mse_xgb:.2f}")
print(f" rmse_xgb (RMSE): {rmse_xgb:.2f}")
print(f" mae_xgb (MAE): {mae_xgb:.2f}")

print(f"r2_xgb (R2): {r2_xgb:.2f}")
```

```
XGBoost model loss:
mse_xgb (MSE): 0.07
 rmse_xgb (RMSE): 0.27
 mae_xgb (MAE): 0.18
r2_xgb (R2): 0.73
```

In [114…

```
# Compare the models based on their evaluation metrics
models = ["Lasso Regression", "Ridge Regression", "Random Forest", "XGBoost"]
mse_scores = [mse_lasso, mse_ridge, mse_rf, mse_xgb]
rmse_scores = [rmse_lasso, rmse_ridge, rmse_rf, rmse_xgb]
mae_scores = [mae_lasso, mae_ridge, mae_rf, mae_xgb]
r2_scores = [r2_lasso, r2_ridge, r2_rf, r2_xgb]
```

In [115…

```
print("Model Comparison based on Evaluation Metrics:")
print("{:<20} {:<12} {:<12} {:<12} {:<12}".format("Model", "MSE", "RMSE", "MAE", "R2 S
for model, mse_score, rmse_score, mae_score, r2_score in zip(models, mse_scores, rmse_
    print("{:<20} {:<12.2f} {:<12.2f} {:<12.2f} {:<12.2f}".format(model, mse_score, rm
```

```
Model Comparison based on Evaluation Metrics:
Model                MSE          RMSE         MAE          R2 Score
Lasso Regression     0.09         0.29         0.23         0.69
Ridge Regression     0.08         0.29         0.22         0.70
Random Forest        0.08         0.28         0.14         0.72
XGBoost              0.07         0.27         0.18         0.73
```

from above evaluation metrics :

- best moldel is Random forest regression model less erros and r2 score is best
- MSE=0.07 your model is making accurate predictions
- RMSE=0.26 QUITE GOOD
- R2 SCORE-0.76 MEANS RF model explains 76% of the variance in the given dataset .

Select the best model and its features:

In [116…

```
# Select the best model and its features
best_model = None
best_r2_score = max(r2_rf, r2_lasso, r2_ridge, r2_xgb)

if best_r2_score == r2_rf:
    best_model = best_rf

elif best_r2_score == r2_lasso:
    best_model = best_lasso

elif best_r2_score == r2_ridge:
    best_model = best_ridge
else:
    best_model = best_xgb
```

In [117…
```
#Get the selected features from RFE
selected_features_best_model = np.array(X.columns)[rfe.support_]
```

In [120…
```
print("Best Model:", best_model)
```

```
Best Model: XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

In [121…
```
print("Selected Features for Best Model:", selected_features_best_model)
```

```
Selected Features for Best Model: ['MSSubClass' 'LotArea' 'LotShape' 'Neighborhood'
'OverallQual'
 'BsmtUnfSF' 'HeatingQC' 'TotRmsAbvGrd' 'Fireplaces' 'GarageFinish'
 'HouseAge' 'YearRemodel' 'TotalSF' 'Bsmt' 'TotalBathroom']
```

Create a DataFrame with the best features from all models

In [122…
```
#Create a DataFrame with the best features from all models
best_features_df = pd.DataFrame({'Selected_Features': selected_features_best_model})
```

In [123…
```
best_features_df  # select total 15 features
```

Out[123]:

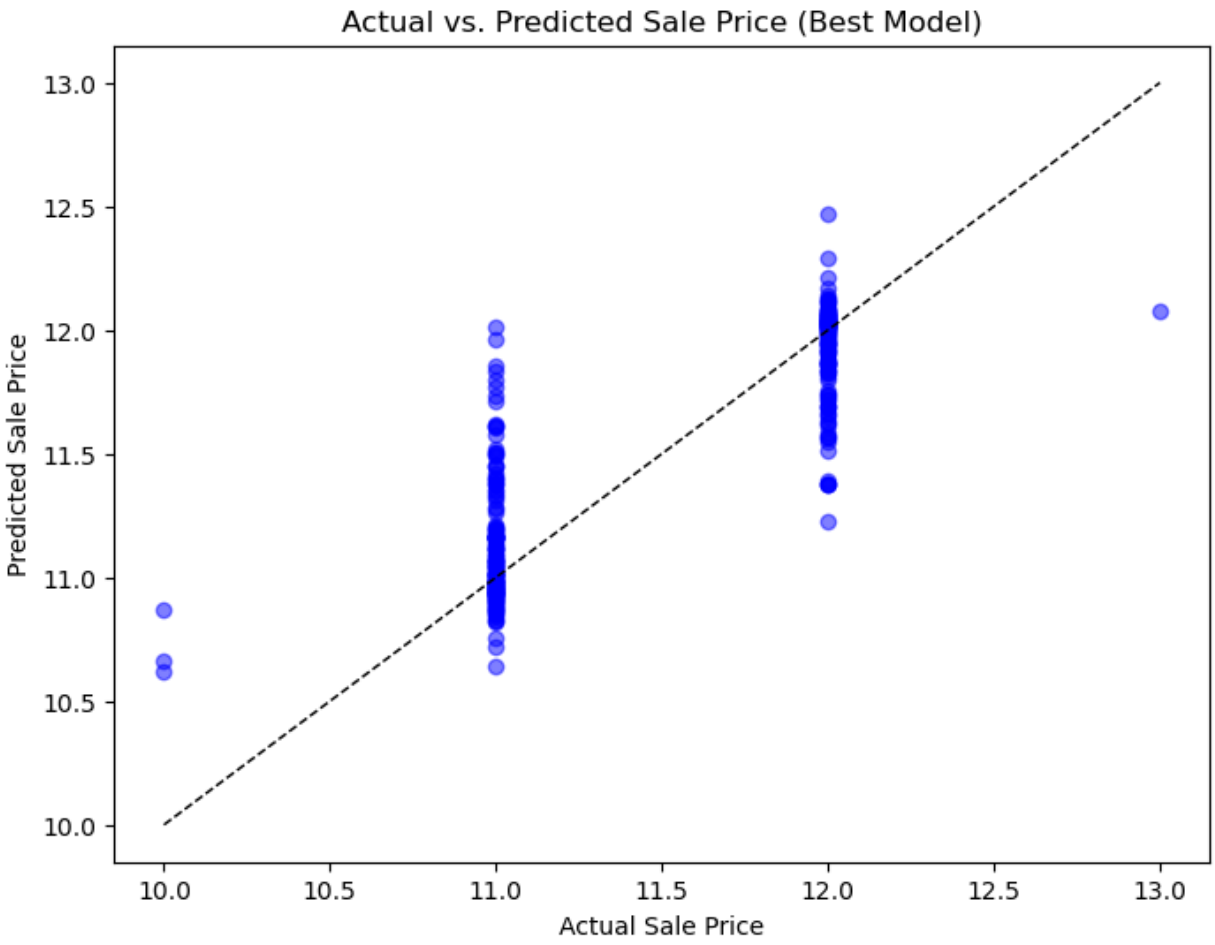
	Selected_Features
0	MSSubClass
1	LotArea
2	LotShape
3	Neighborhood
4	OverallQual
5	BsmtUnfSF
6	HeatingQC
7	TotRmsAbvGrd
8	Fireplaces
9	GarageFinish
10	HouseAge
11	YearRemodel
12	TotalSF
13	Bsmt
14	TotalBathroom

- Residuals of XGB model :

- Plot actual vs. predicted line for the best model

In [128…

```
# Plot actual vs. predicted line for the best model
y_pred_best_model = best_model.predict(X_test)

plt.figure(figsize=(8, 6))

plt.scatter(y_test, y_pred_best_model, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'k--', lw=1)
plt.xlabel('Actual Sale Price')
plt.ylabel('Predicted Sale Price')
plt.title('Actual vs. Predicted Sale Price (Best Model)')
plt.show()
```



In [129…
```
# Create a DataFrame with the best features from all models
selected_features_df = pd.DataFrame(columns=['Model', 'Selected_Features'])

selected_features_df = selected_features_df.append({'Model': 'Random Forest', 'Selecte
selected_features_df = selected_features_df.append({'Model': 'Lasso Regression', 'Sele
selected_features_df = selected_features_df.append({'Model': 'Ridge Regression', 'Sele
selected_features_df = selected_features_df.append({'Model': 'XGBoost Regression', 'Se

print("DataFrame with Selected Features from all models:")
```

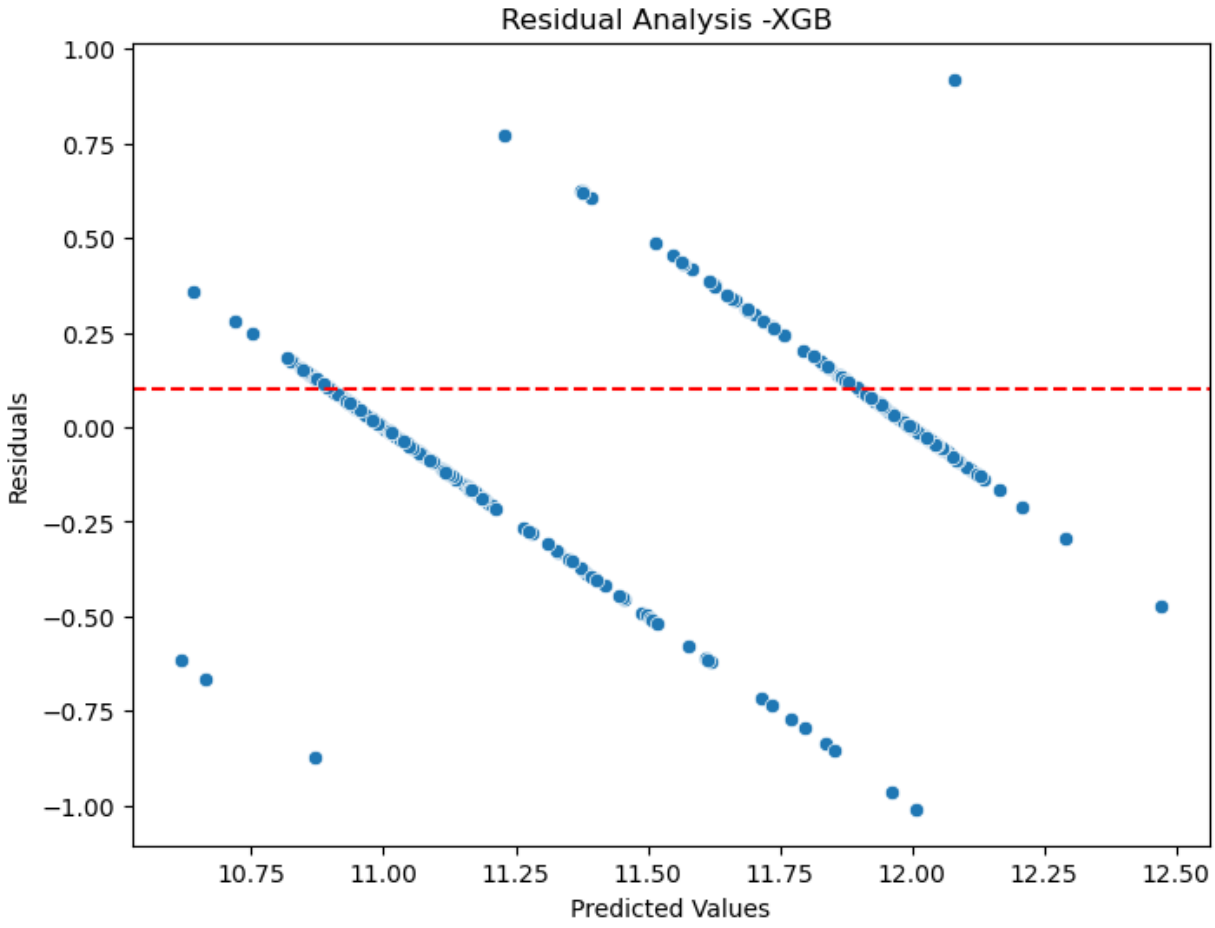
```
DataFrame with Selected Features from all models:
               Model                                  Selected_Features
0      Random Forest  [MSSubClass, LotArea, LotShape, Neighborhood, ...
1    Lasso Regression  [MSSubClass, LotArea, LotShape, Neighborhood, ...
2    Ridge Regression  [MSSubClass, LotArea, LotShape, Neighborhood, ...
3  XGBoost Regression  [MSSubClass, LotArea, LotShape, Neighborhood, ...
```

In [132…
```
selected_features_df
```

Out[132]:

	Model	Selected_Features
0	Random Forest	[MSSubClass, LotArea, LotShape, Neighborhood, ...
1	Lasso Regression	[MSSubClass, LotArea, LotShape, Neighborhood, ...
2	Ridge Regression	[MSSubClass, LotArea, LotShape, Neighborhood, ...
3	XGBoost Regression	[MSSubClass, LotArea, LotShape, Neighborhood, ...

In [125…
```
residuals = y_test - y_pred_xgb

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred_xgb, y=residuals)
plt.axhline(y=0.1, color='red', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Analysis -XGB ')
plt.show()
```

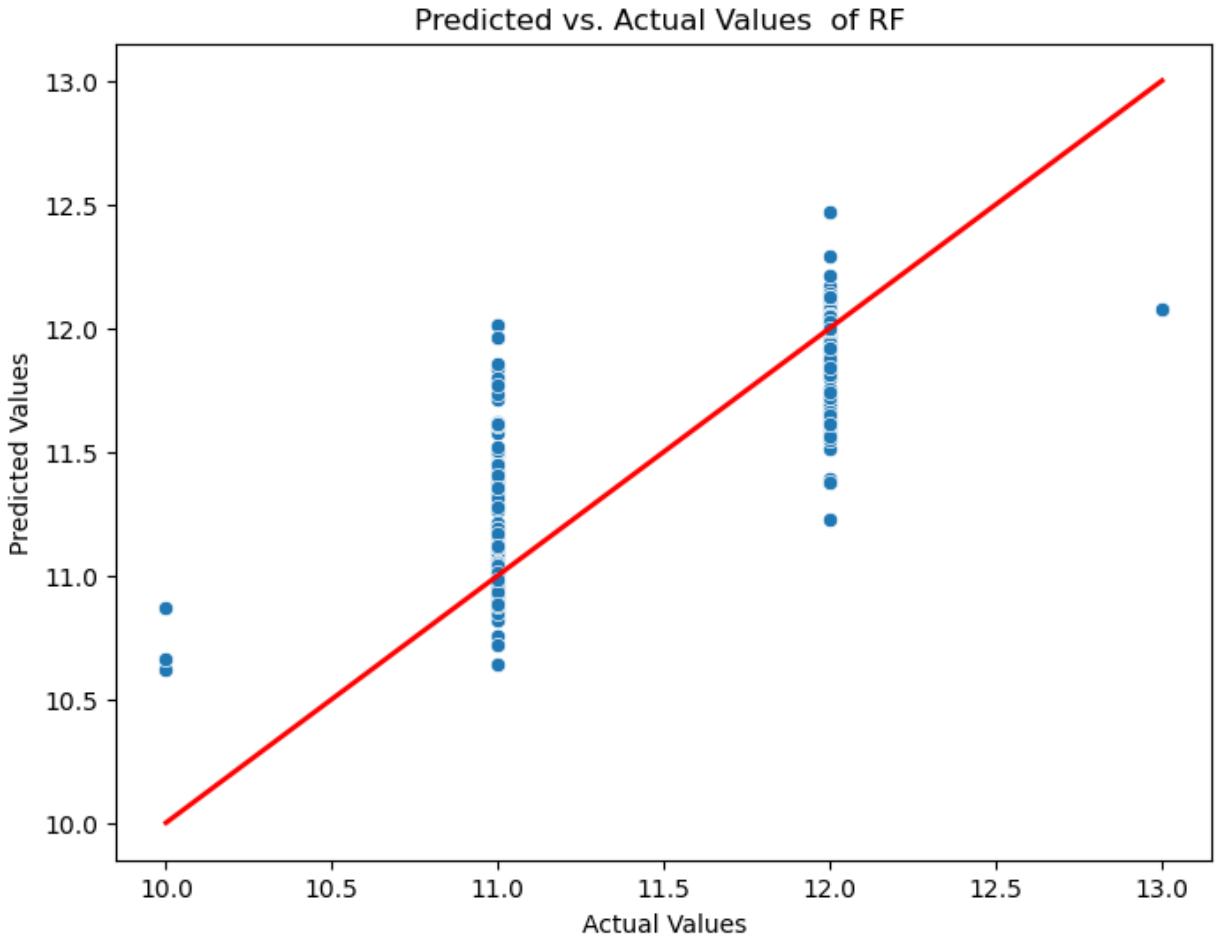


In [126…
```
residuals = y_test - y_pred_xgb
print("residualsOR  eror in RF MODEL :",residuals)
```

```
residualsOR  eror in RF MODEL : Id
1042    0.622159
1131   -0.265768
1003   -0.024603
1325   -1.009092
534    -0.617208
          ...
900     0.019813
1335    0.114232
1417   -0.117501
721     0.004433
255    -0.355408
Name: SalePrice, Length: 291, dtype: float64
```

In [133…
```
# Predicted vs. Actual Plot using Seaborn
plt.figure(figsize=(8, 6))
```

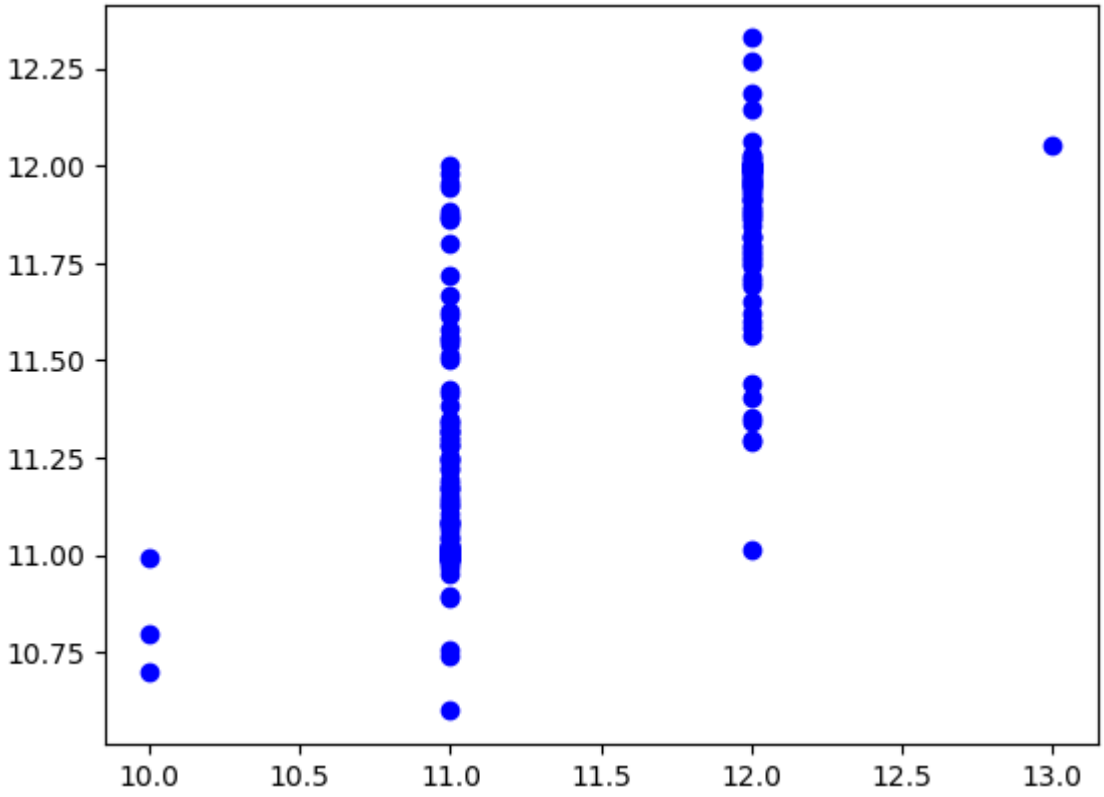
```
sns.scatterplot(x=y_test, y=y_pred_xgb)
sns.lineplot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', line
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Predicted vs. Actual Values  of RF ')
plt.show()
```



SUMMARY :

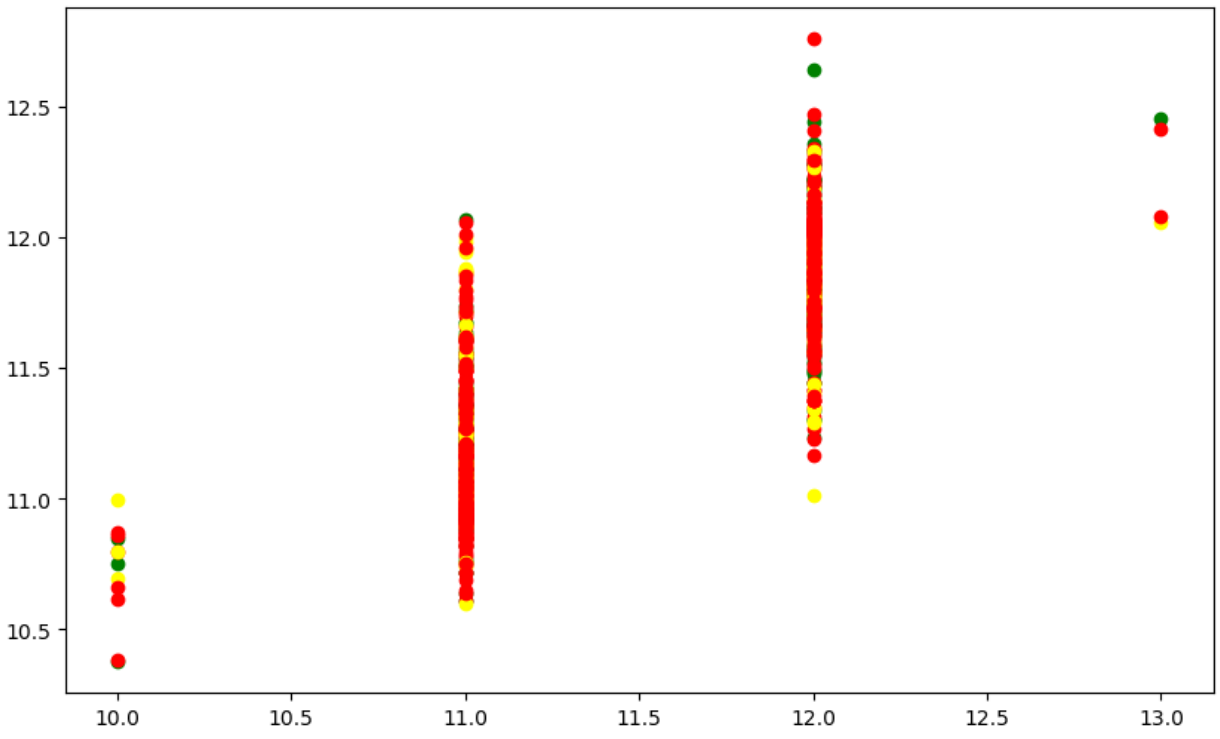
In [134…
```
plt.scatter(y_test, y_pred_rf, color='blue', label='Linear Regression')
```

Out[134]:
```
<matplotlib.collections.PathCollection at 0x8b7db050d0>
```



In [135…

```
# Plot the line graph for each regression model
plt.figure(figsize=(10, 6))

plt.scatter(y_test, y_pred_lasso, color='green', label='Lasso Regression')
plt.scatter(y_test, y_pred_ridge, color='red', label='Ridge Regression')
plt.scatter(y_test, y_pred_rf, color="yellow",label='Random Forest Regression')
plt.scatter(y_test, y_pred_xgb, color="red",label='XGBoost Regression')
plt.show()
```



In [136…
```
results_df = pd.DataFrame({"Actual Prices": y_test,"Lasso Regression":y_pred_lasso,
                           "Ridge Regression":y_pred_ridge,"Random Forest":y_pred_rf,"X
```

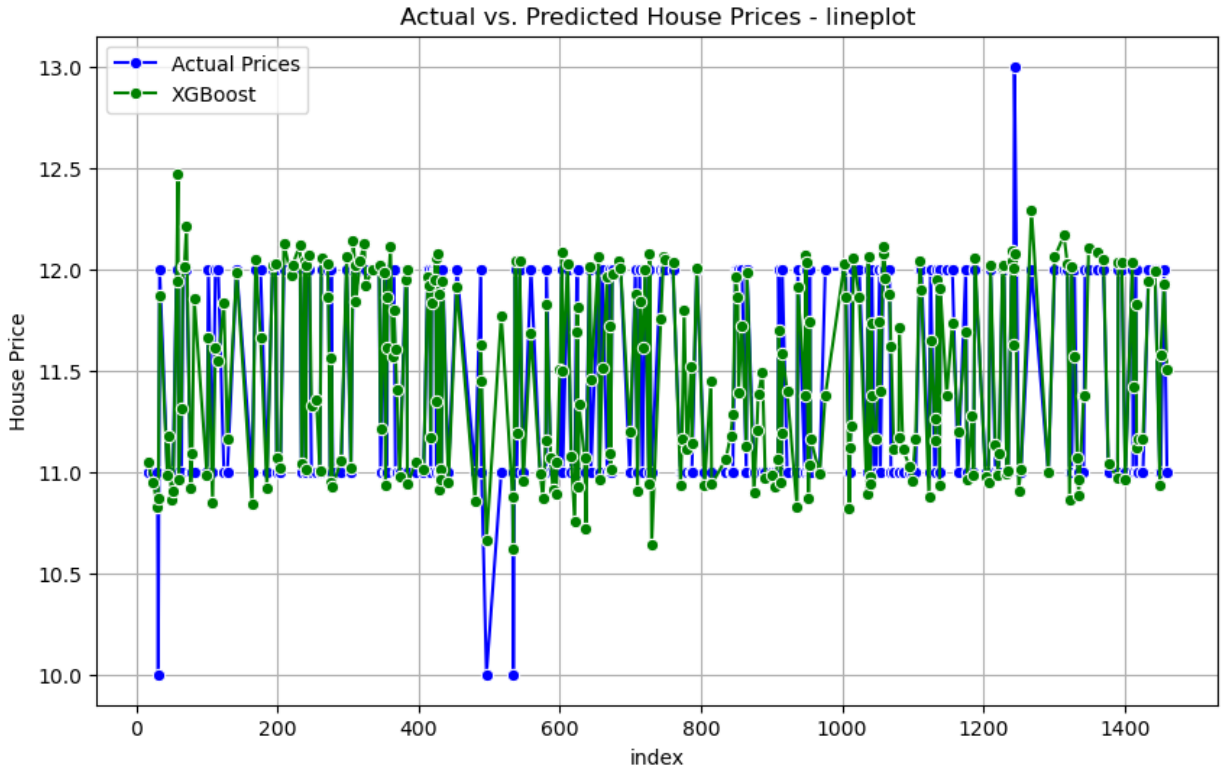
In [137…
```
results_df # check sale price of each model
```

Out[137]:

Id	Actual Prices	Lasso Regression	Ridge Regression	Random Forest	XGBoost
1042	12	11.302802	11.313512	11.342633	11.377841
1131	11	11.335586	11.330792	11.299693	11.265768
1003	12	12.036721	12.045984	11.999990	12.024603
1325	11	12.070155	12.053365	11.999990	12.009092
534	10	10.376540	10.383608	10.696667	10.617208
...	...	...	...	...	...
900	11	11.178835	11.130034	11.000000	10.980187
1335	11	11.038412	10.983363	11.010000	10.885768
1417	11	11.173994	11.103541	11.065918	11.117501
721	12	11.890699	11.917043	11.961337	11.995567
255	11	11.191311	11.129818	11.244116	11.355408

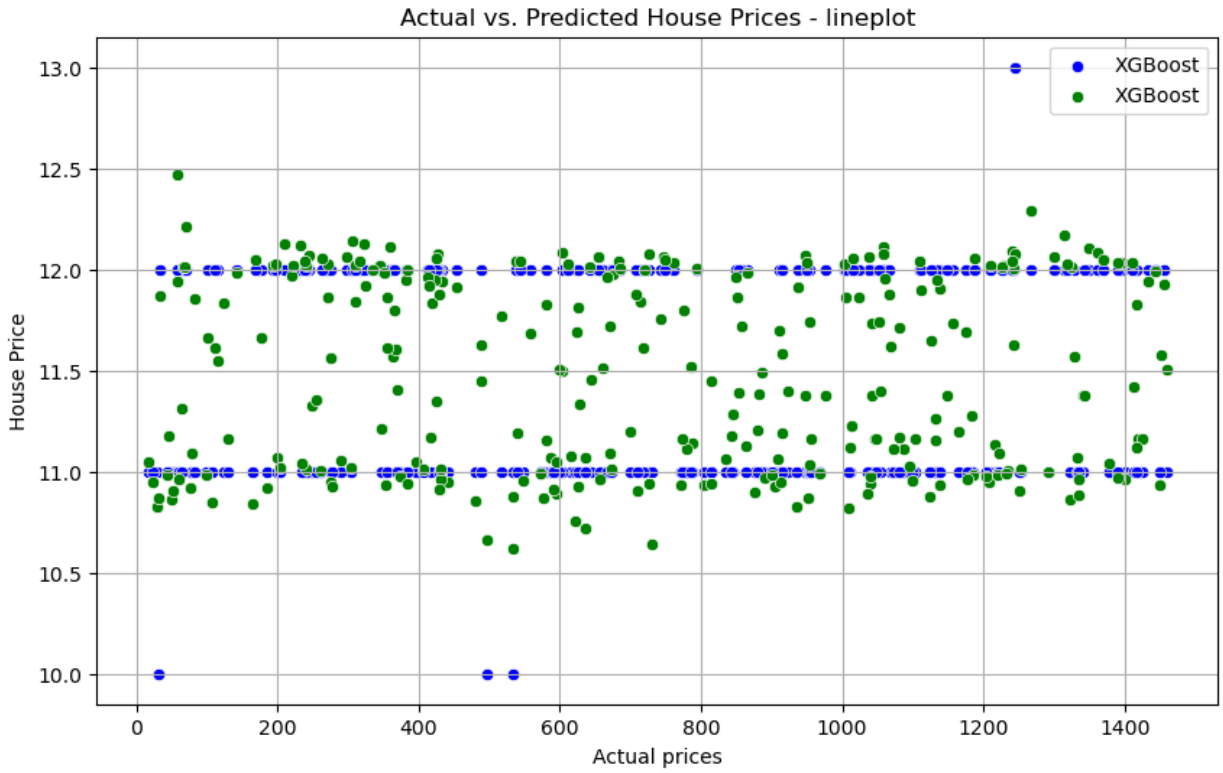
291 rows × 5 columns

In [138…
```
plt.figure(figsize=(10, 6))
#sns.lineplot(data=results_df,x='results_df.index',y='Actual Prices',label='Actual Pri
#sns.lineplot(data=results_df,x='results_df.index',y='Actual Prices',label='Actual Pri

sns.lineplot(data=results_df['Actual Prices'],label='Actual Prices',marker='o',color='
sns.lineplot(data=results_df['XGBoost'],label='XGBoost',marker='o',color='g',linestyle
plt.xlabel('index')
plt.ylabel('House Price')
plt.title('Actual vs. Predicted House Prices - lineplot')
plt.legend()
plt.grid(True)
plt.show()
```



In [139…

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=results_df['Actual Prices'],label='XGBoost',marker='o',color='b',
sns.scatterplot(data=results_df['XGBoost'],label='XGBoost',marker='o',color='g',linest
plt.xlabel('Actual prices')
plt.ylabel('House Price')
plt.title('Actual vs. Predicted House Prices - lineplot')
plt.legend()
plt.grid(True)
plt.show()
```



In the data analysis process, an exploratory data analysis (EDA) was conducted to examine the dataset. During this phase, certain low-impact data points were identified and subsequently removed from consideration. In the feature engineering (FE) phase, new data was generated and incorporated into the dataset to enhance its predictive capabilities. Following the FE stage, a suitable model was selected based on the characteristics of the data. Through rigorous experimentation, appropriate parameters were determined for the chosen model, ensuring optimal performance. Subsequently, the model was built using the finalized dataset and the identified parameters. Perform additional feature engineering (FE) experiments to enhance the quality of the features. Explore the implementation of XGBoost and RandomForestRegressor models, considering their suitability for the dataset. Fine-tune the model parameters to achieve more accurate predictions.

Summary :

- 1) from above results XGboost regressor model is best model
- 2) then Random forest regressor model .
- 3) The ideal ‘r2_score’ of a build should be more than 0.73 (at least > 0.60).

In [ ]: