In [1]:
```
#import warnings
import warnings
warnings.filterwarnings('ignore')
```

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

In [3]:
```
# Import required libaries for Ridge, Lasso and GridSearchCV
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV
```

In [4]:
```
# Setting option to display all the columns and rows in dataset
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows',None)
```

In [5]:
```
#loading the dataset
df=pd.read_csv('train.csv')
```

In [6]:
```
df.head()
```

Out[6]:

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

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     588 non-null    object 
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object 
 28  ExterCond      1460 non-null   object 
 29  Foundation     1460 non-null   object 
 30  BsmtQual       1423 non-null   object 
 31  BsmtCond       1423 non-null   object 
 32  BsmtExposure   1422 non-null   object 
 33  BsmtFinType1   1423 non-null   object 
 34  BsmtFinSF1     1460 non-null   int64  
 35  BsmtFinType2   1422 non-null   object 
 36  BsmtFinSF2     1460 non-null   int64  
 37  BsmtUnfSF      1460 non-null   int64  
 38  TotalBsmtSF    1460 non-null   int64  
 39  Heating        1460 non-null   object 
 40  HeatingQC      1460 non-null   object 
 41  CentralAir     1460 non-null   object 
 42  Electrical     1459 non-null   object 
 43  1stFlrSF       1460 non-null   int64  
 44  2ndFlrSF       1460 non-null   int64  
 45  LowQualFinSF   1460 non-null   int64  
 46  GrLivArea      1460 non-null   int64  
 47  BsmtFullBath   1460 non-null   int64  
 48  BsmtHalfBath   1460 non-null   int64  
 49  FullBath       1460 non-null   int64  
 50  HalfBath       1460 non-null   int64  
 51  BedroomAbvGr   1460 non-null   int64  
 52  KitchenAbvGr   1460 non-null   int64  
 53  KitchenQual    1460 non-null   object 
 54  TotRmsAbvGrd   1460 non-null   int64  
```

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

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Id                 0
MSSubClass         0
MSZoning           0
LotFrontage      259
LotArea            0
Street             0
Alley           1369
LotShape           0
LandContour        0
Utilities          0
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        0
Exterior2nd        0
MasVnrType       872
MasVnrArea         8
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinSF1         0
BsmtFinType2      38
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
Heating            0
HeatingQC          0
CentralAir         0
Electrical         1
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        0
TotRmsAbvGrd       0
Functional         0
Fireplaces         0
FireplaceQu      690
GarageType        81
GarageYrBlt       81
```

```
GarageFinish       81
GarageCars          0
GarageArea          0
GarageQual         81
GarageCond         81
PavedDrive          0
WoodDeckSF          0
OpenPorchSF         0
EnclosedPorch       0
3SsnPorch           0
ScreenPorch         0
PoolArea            0
PoolQC           1453
Fence            1179
MiscFeature      1406
MiscVal             0
MoSold              0
YrSold              0
SaleType            0
SaleCondition       0
SalePrice           0
dtype: int64
```

In [9]:
```
df.isnull().any().sum()
```

Out[9]:
```
19
```

In [10]:
```
null_columns=df.columns[df.isnull().any()]
null_columns
```

Out[10]:
```
Index(['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence',
       'MiscFeature'],
      dtype='object')
```

In [11]:
```
null_columns.shape
```

Out[11]:
```
(19,)
```

In [12]:
```
null_count=df[null_columns].isnull().sum().sort_values(ascending=False)
null_per=(df[null_columns].isnull().sum()*100/df.shape[0]).sort_values(ascending
```

In [13]:
```
null_data=pd.concat([null_count,null_per],axis=1,keys=['Count', 'Percentage'])
```

In [14]:
```
null_data
```

Out[14]:

	Count	Percentage
PoolQC	1453	99.520548
MiscFeature	1406	96.301370
Alley	1369	93.767123
Fence	1179	80.753425
MasVnrType	872	59.726027
FireplaceQu	690	47.260274
LotFrontage	259	17.739726
GarageType	81	5.547945
GarageYrBlt	81	5.547945
GarageFinish	81	5.547945
GarageQual	81	5.547945
GarageCond	81	5.547945
BsmtFinType2	38	2.602740
BsmtExposure	38	2.602740
BsmtFinType1	37	2.534247
BsmtCond	37	2.534247
BsmtQual	37	2.534247
MasVnrArea	8	0.547945
Electrical	1	0.068493

In [15]:
```
df.drop(columns=null_data[ null_data['Percentage'] > 15].index, inplace=True)
```

In [16]:
```
df.shape
```

Out[16]: (1460, 74)

-->Use a countplot to determine which values in the columns are the most frequent.

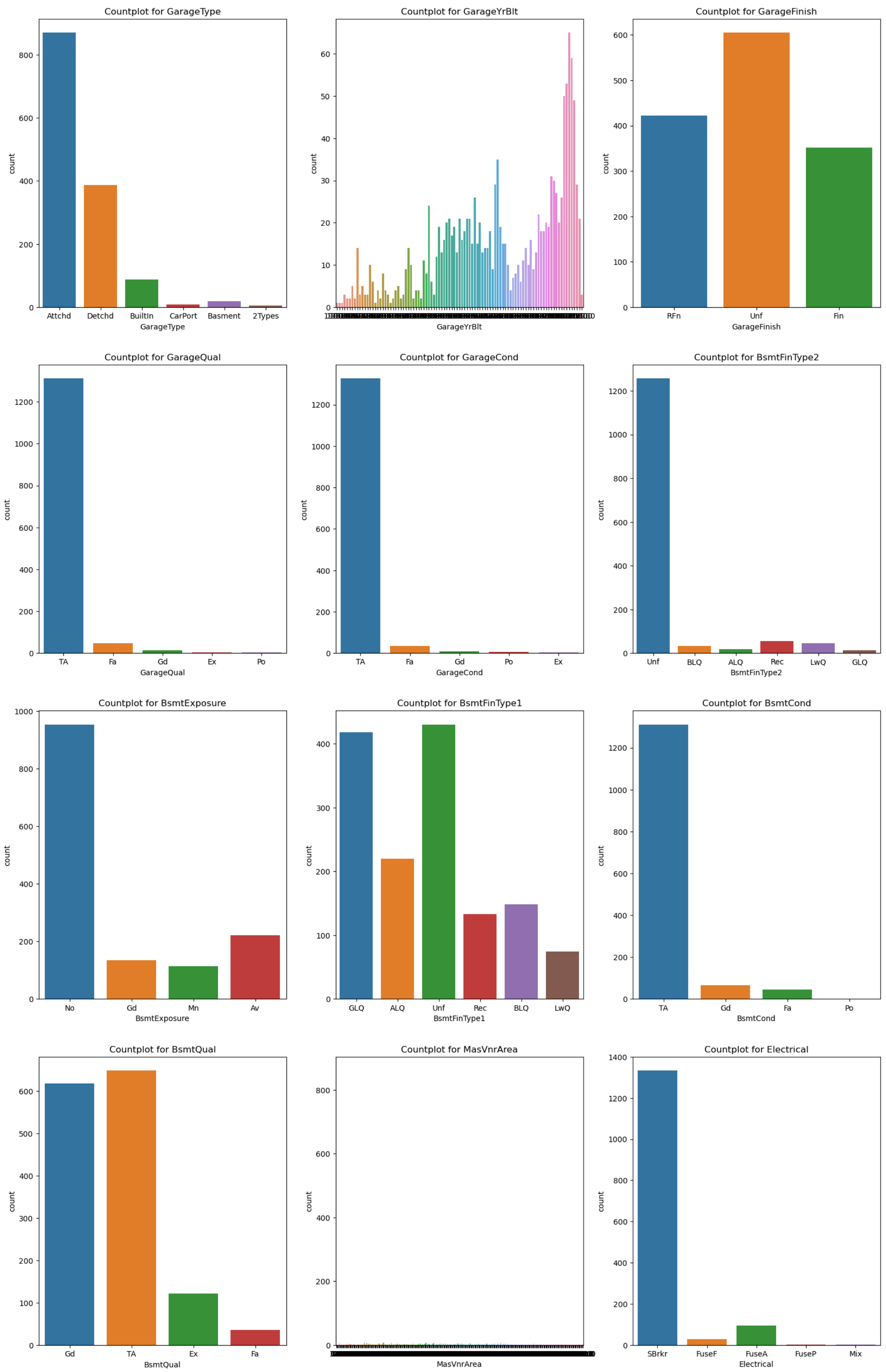
In [17]:
```
null_data = null_data[null_data['Percentage'] < 15]
null_data
```

Out[17]:

	Count	Percentage
GarageType	81	5.547945
GarageYrBlt	81	5.547945
GarageFinish	81	5.547945
GarageQual	81	5.547945
GarageCond	81	5.547945
BsmtFinType2	38	2.602740
BsmtExposure	38	2.602740
BsmtFinType1	37	2.534247
BsmtCond	37	2.534247
BsmtQual	37	2.534247
MasVnrArea	8	0.547945
Electrical	1	0.068493

In [18]:

```
%matplotlib inline
plt.rcParams['figure.figsize']=20,40
for i,var in enumerate(null_data.index,start=1):
    plt.subplot(5,3,i)
    sns.countplot(x=var,data=df)
    plt.title(f"Countplot for {var}")
```



We will use the following method to conduct imputation for these columns:

- If the column is `categorical`, the missing values will be replaced using `mode()`.
- If the column contains `numerical values`, the missing values will be replaced using `median()`.

- If the value `NA` in the column has a meaningful value (GarageType = NA, for example, denotes "No Garage"). These settings will be changed to `None` .

In [19]:
```
#meaningfull categorical columns
meaning_col=['GarageType','GarageFinish','GarageQual','GarageCond','BsmtExposure
for feature in meaning_col:
    df[feature].fillna('None',inplace=True)
```

In [20]:
```
df['GarageYrBlt'].fillna(df['GarageYrBlt'].median(), inplace=True)
```

In [21]:
```
df['MasVnrArea'].fillna(df['MasVnrArea'].median(), inplace=True)
```

In [22]:
```
df['Electrical'].fillna(df['Electrical'].mode()[0], inplace=True)
```

In [23]:
```
df.isnull().any().sum()
```

Out[23]: 0

Dropping unimportant columns

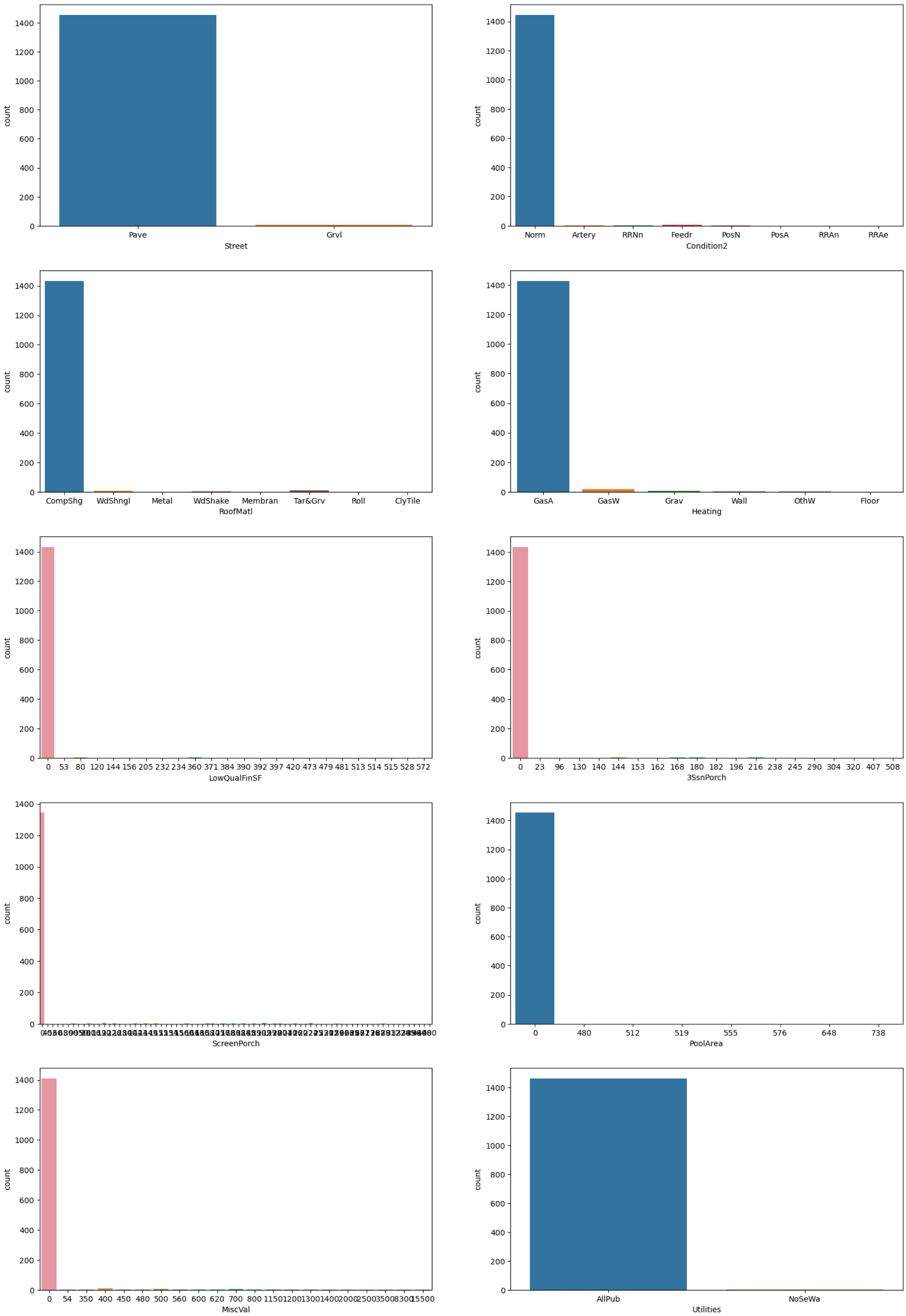
The columns will be removed in accordance with:

1. The model has less room to learn if the variance in the column is really low. These columns are going away.
2. A few columns, such `Id` , are meaningless because they don't offer any insightful information. We'll get rid of these too

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

Out[24]: (1460, 74)

In [25]:
```
%matplotlib inline
plt.rcParams['figure.figsize']=20,30
#unimportant columns
unin_col=['Street','Condition2','RoofMatl','Heating','LowQualFinSF','3SsnPorch',
for i,var in enumerate(unin_col,start=1):
    plt.subplot(5,2,i)
    sns.countplot(x=var,data=df)
    df.drop(columns=var,inplace=True)
```



In [26]:
```
df.drop(columns='Id',inplace=True)
```

In [27]:
```
df.shape
```

Out[27]: (1460, 63)

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 63 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   MSSubClass     1460 non-null   int64  
 1   MSZoning       1460 non-null   object 
 2   LotArea        1460 non-null   int64  
 3   LotShape       1460 non-null   object 
 4   LandContour    1460 non-null   object 
 5   LotConfig      1460 non-null   object 
 6   LandSlope      1460 non-null   object 
 7   Neighborhood   1460 non-null   object 
 8   Condition1     1460 non-null   object 
 9   BldgType       1460 non-null   object 
 10  HouseStyle     1460 non-null   object 
 11  OverallQual    1460 non-null   int64  
 12  OverallCond    1460 non-null   int64  
 13  YearBuilt      1460 non-null   int64  
 14  YearRemodAdd   1460 non-null   int64  
 15  RoofStyle      1460 non-null   object 
 16  Exterior1st    1460 non-null   object 
 17  Exterior2nd    1460 non-null   object 
 18  MasVnrArea     1460 non-null   float64
 19  ExterQual      1460 non-null   object 
 20  ExterCond      1460 non-null   object 
 21  Foundation     1460 non-null   object 
 22  BsmtQual       1460 non-null   object 
 23  BsmtCond       1460 non-null   object 
 24  BsmtExposure   1460 non-null   object 
 25  BsmtFinType1   1460 non-null   object 
 26  BsmtFinSF1     1460 non-null   int64  
 27  BsmtFinType2   1460 non-null   object 
 28  BsmtFinSF2     1460 non-null   int64  
 29  BsmtUnfSF      1460 non-null   int64  
 30  TotalBsmtSF    1460 non-null   int64  
 31  HeatingQC      1460 non-null   object 
 32  CentralAir     1460 non-null   object 
 33  Electrical     1460 non-null   object 
 34  1stFlrSF       1460 non-null   int64  
 35  2ndFlrSF       1460 non-null   int64  
 36  GrLivArea      1460 non-null   int64  
 37  BsmtFullBath   1460 non-null   int64  
 38  BsmtHalfBath   1460 non-null   int64  
 39  FullBath       1460 non-null   int64  
 40  HalfBath       1460 non-null   int64  
 41  BedroomAbvGr   1460 non-null   int64  
 42  KitchenAbvGr   1460 non-null   int64  
 43  KitchenQual    1460 non-null   object 
 44  TotRmsAbvGrd   1460 non-null   int64  
 45  Functional     1460 non-null   object 
 46  Fireplaces     1460 non-null   int64  
 47  GarageType     1460 non-null   object 
 48  GarageYrBlt    1460 non-null   float64
 49  GarageFinish   1460 non-null   object 
 50  GarageCars     1460 non-null   int64  
 51  GarageArea     1460 non-null   int64  
 52  GarageQual     1460 non-null   object 
 53  GarageCond     1460 non-null   object 
 54  PavedDrive     1460 non-null   object 
```

```
 55  WoodDeckSF     1460 non-null   int64
 56  OpenPorchSF    1460 non-null   int64
 57  EnclosedPorch  1460 non-null   int64
 58  MoSold         1460 non-null   int64
 59  YrSold         1460 non-null   int64
 60  SaleType       1460 non-null   object
 61  SaleCondition  1460 non-null   object
 62  SalePrice      1460 non-null   int64
dtypes: float64(2), int64(29), object(32)
memory usage: 718.7+ KB
```

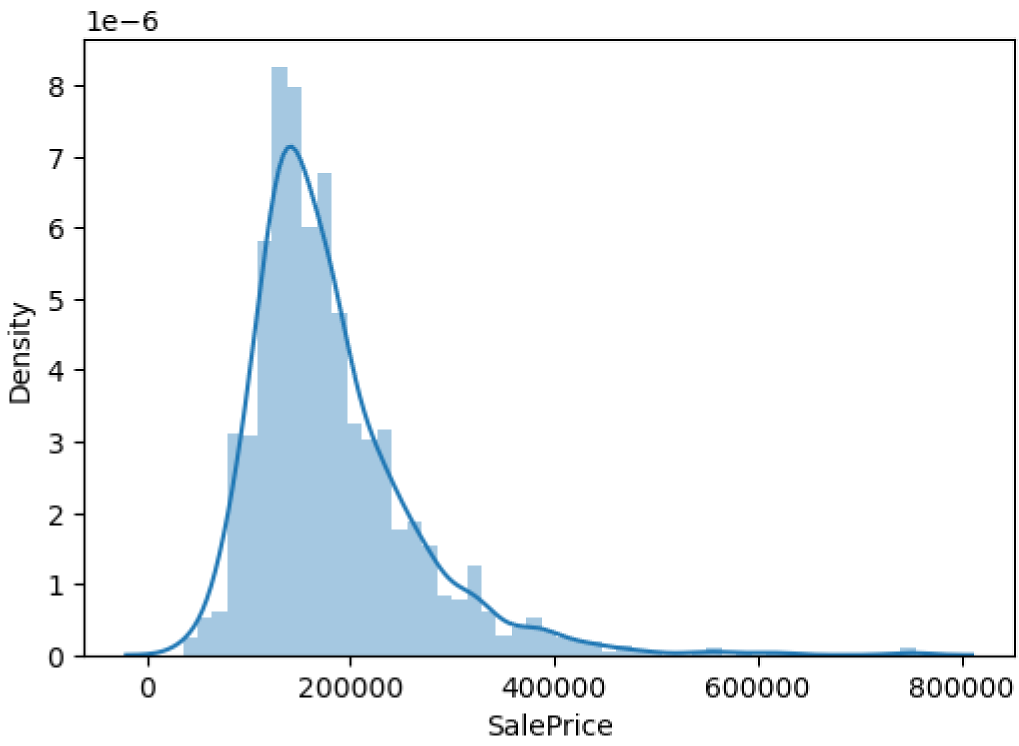
In [29]:
```
# Getting categorical variables
cat_var = df.select_dtypes(include='object').columns
cat_var
```

Out[29]:
```
Index(['MSZoning', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope',
       'Neighborhood', 'Condition1', 'BldgType', 'HouseStyle', 'RoofStyle',
       'Exterior1st', 'Exterior2nd', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional',
       'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

In [30]:
```
# Getting numerical variables
num_var = df.select_dtypes(exclude='object').columns
num_var
```

Out[30]:
```
Index(['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'BsmtFullBath',
       'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt', 'GarageCars', 'GarageArea',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', 'MoSold', 'YrSold',
       'SalePrice'],
      dtype='object')
```

In [31]:
```
%matplotlib inline
plt.rcParams['figure.figsize']=6,4
sns.distplot(df['SalePrice'])
plt.show()
```



- The distribution is skewed to the right, as can be seen by looking at it (i.e., outliers on data with high Sales Price). This suggests that there are anomalies present.

In [32]:
```
df['SalePrice'].skew()
```
Out[32]: 1.8828757597682129

The general guideline for determining skewness:

1. In the range of -0.5 to 0.5, the skewness indicates a reasonably symmetric set of data.
2. The data are substantially skewed if the skewness falls between -1 and –0.5 or between 0.5 and 1.
3. The data are significantly skewed if the skewness is greater than 1 or less than -1.

In [33]:
```
df['SalePrice'].kurtosis()
```
Out[33]: 6.536281860064529

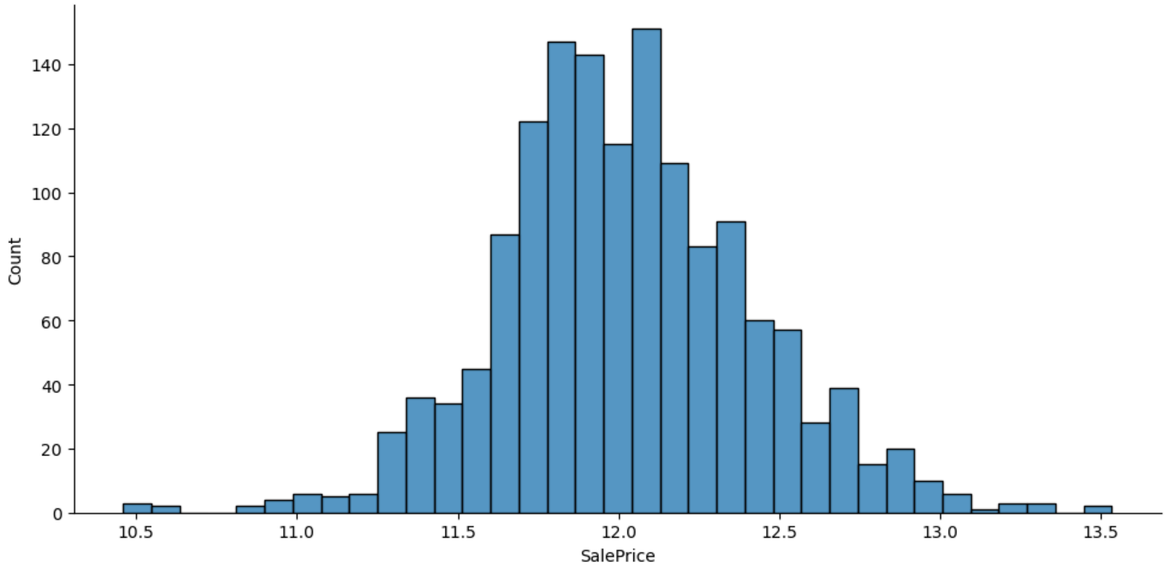
- Kurtosis calculates the distribution's tail-heaviness.
- Kurtosis value for a normal distribution is 3.
- The tail is heavier when the kurtosis value rises, and vice versa.

In our case as the kurtosis value is more than ~6.5, distribution tail is heavier

Handling SalePrice high skewness and kurtosis

- To handle this, we will perform Log Transformation on "SalePrice" column. This will transform the variable and make it as normally distributed as possible. Basically it reduces the skewness in the data

In [34]:
```
%matplotlib inline
plt.rcParams['figure.figsize']=6,4
sns.displot(np.log(df['SalePrice']), aspect=2)
plt.show()
```



- The data now roughly conforms to a normal distribution.

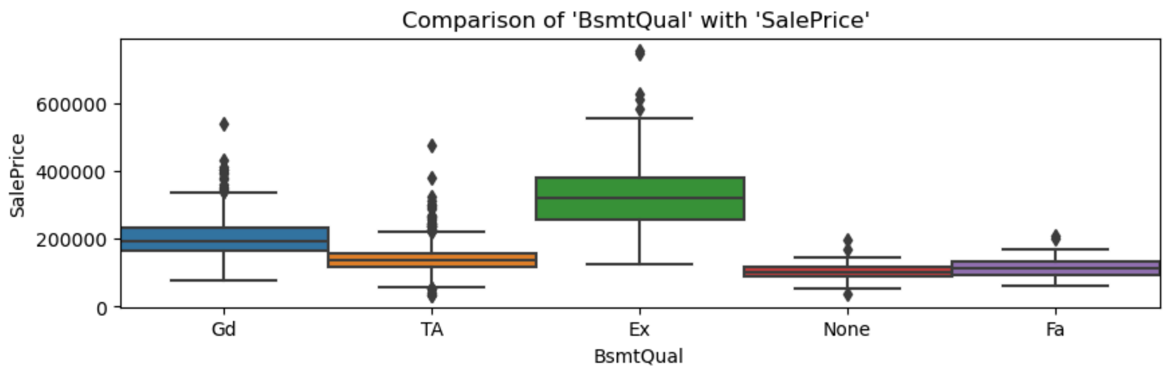
In [35]:
```
df['Transformed_SalePrice'] = np.log(df['SalePrice'])
```

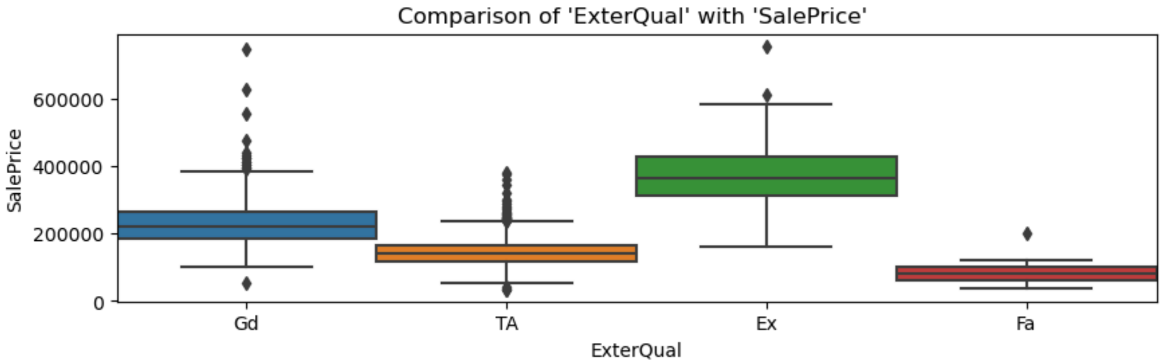
In [36]:
```
df.shape
```

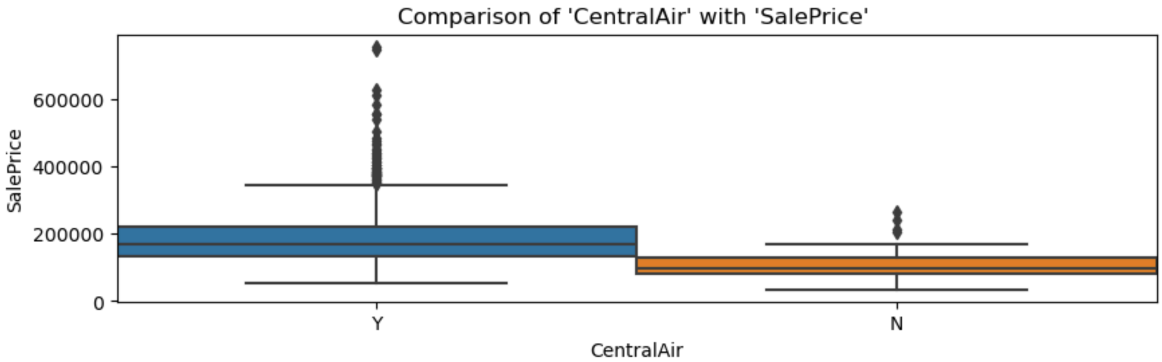
Out[36]: (1460, 64)

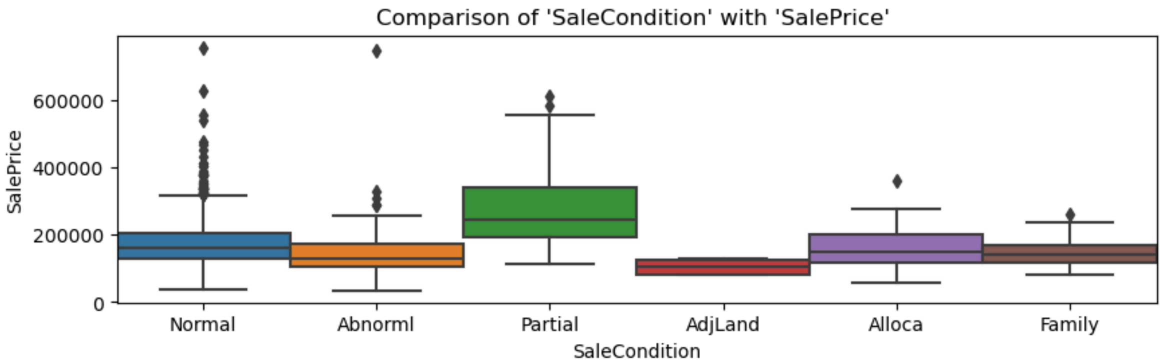
EDA- Exploratory Data Analysis

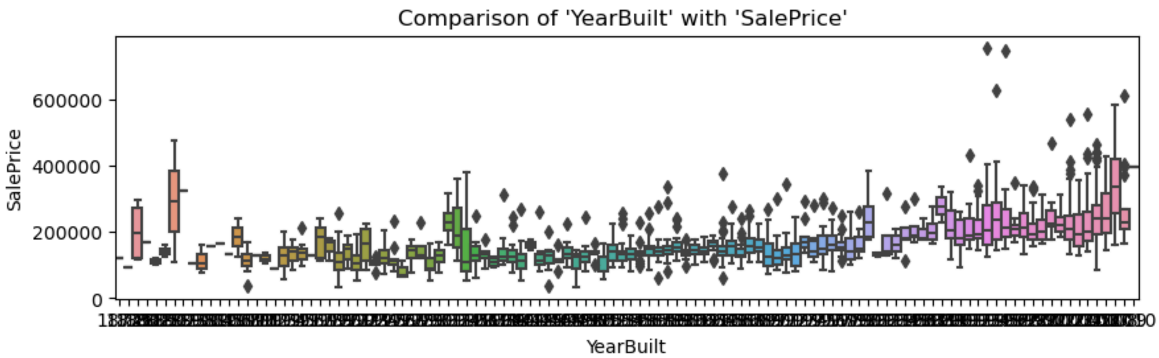
In [37]:
```
%matplotlib inline
plt.rcParams['figure.figsize']=10,15
a2=['BsmtQual','ExterQual','CentralAir','SaleCondition','YearBuilt']
for i,var in enumerate(a2,start=1):
    plt.subplot(5,1,i)
    sns.boxplot(x=var, y='SalePrice', data=df, width=1)
    plt.title(f"Comparison of '{var}' with 'SalePrice' ")
    plt.show()
```







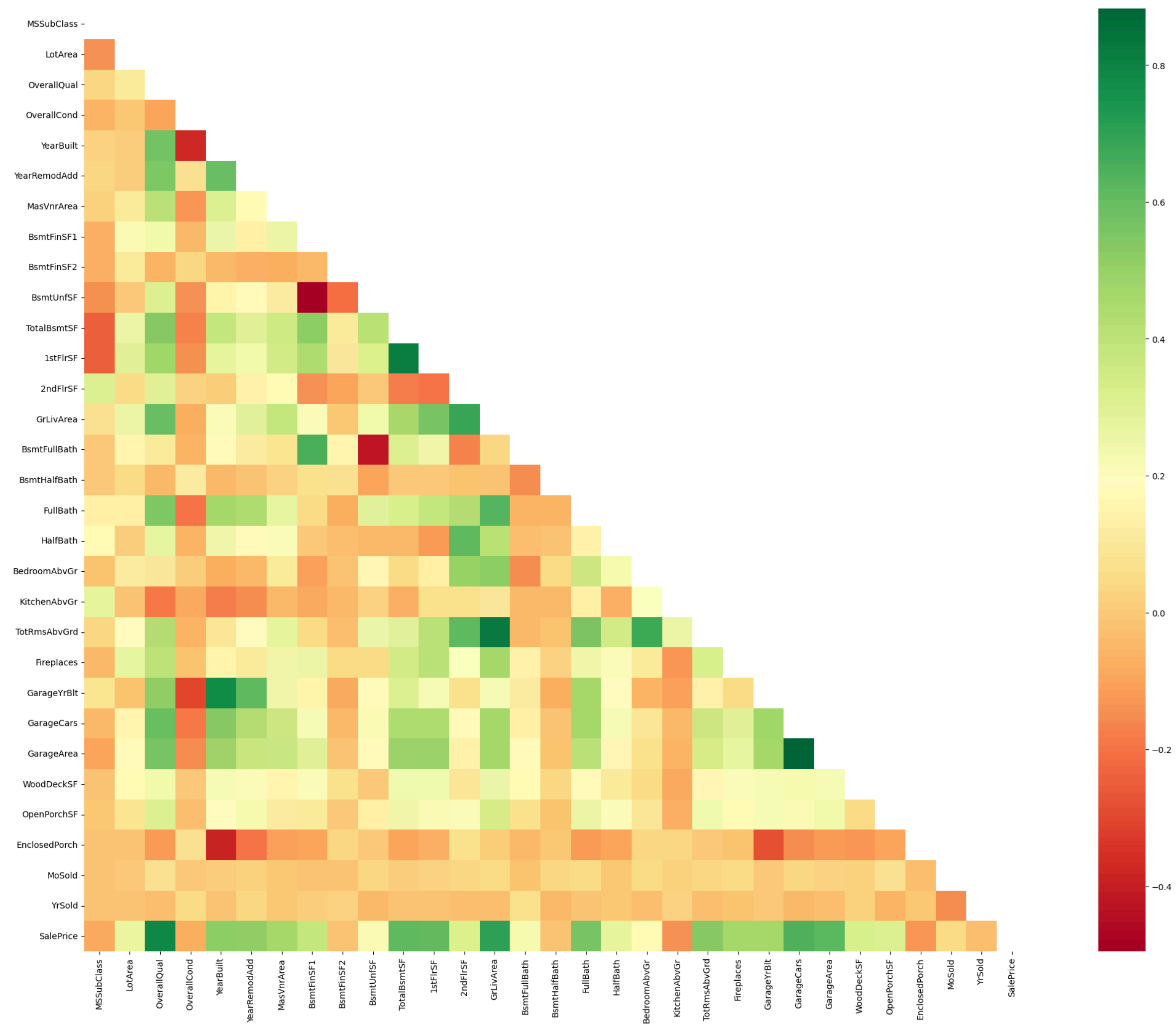




- As Basement quality increases from Fair to Excellent, we see a corresponding increase in SalePrice
- As Exterior quality increases from Fair to Excellent, we see a corresponding increase in SalePrice
- Houses with Central Air conditioning have a higher median price compared to the houses that don't have Central Air conditioning
- Houses that are partially completed have a higher median Saleprice compared to other categories. This might be because partially completed houses are usually new houses under construction.

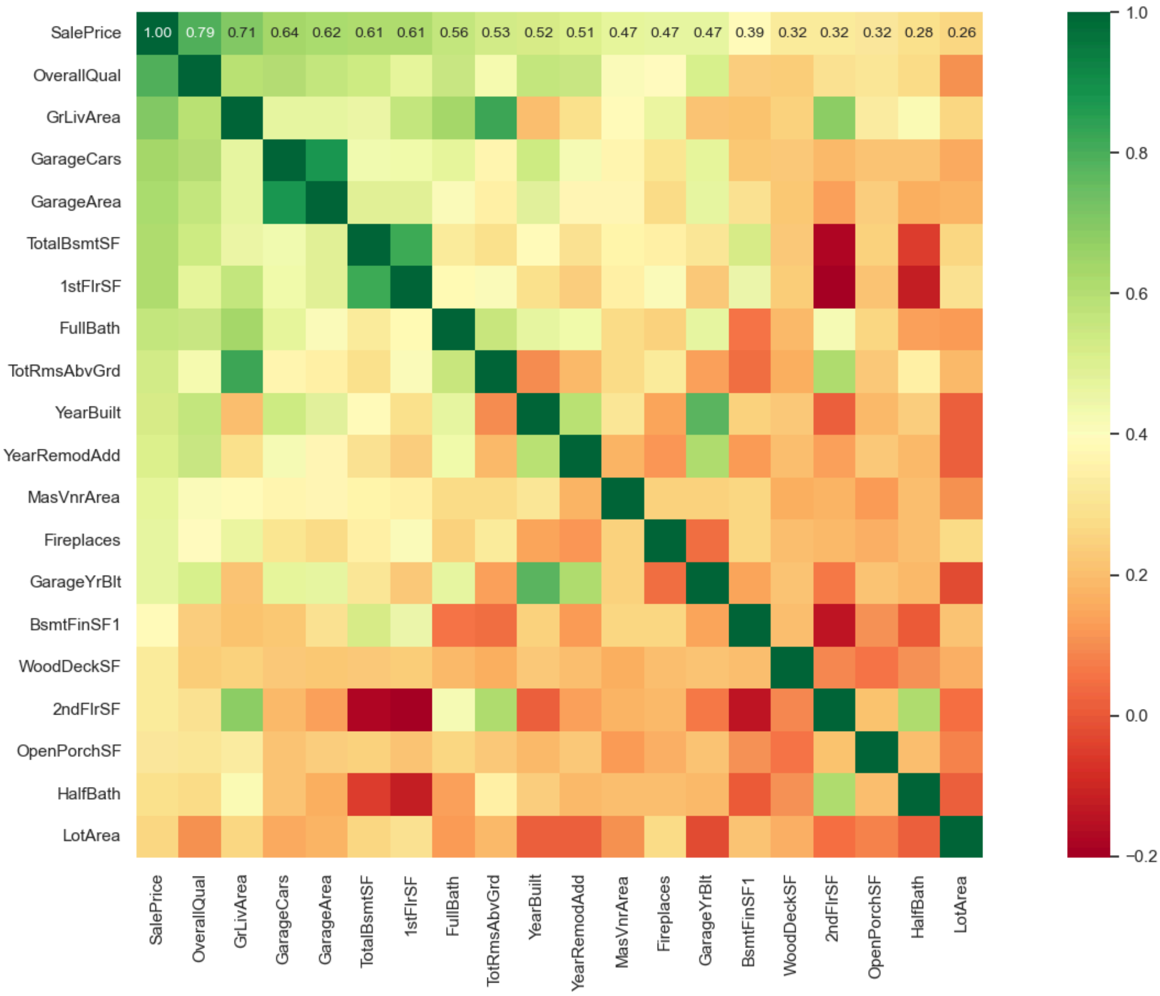
- As the house age increases, we can see that the median SalePrice drops but there are few cases where the SalePrice goes up as well

In [38]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Assuming df is your DataFrame

# Drop the 'Transformed_SalePrice' column and select only numeric columns
numeric_df = df.drop(columns='Transformed_SalePrice', axis=1).select_dtypes(excl

corr = numeric_df.corr()

# Create a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=[30, 20])

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, cmap='RdYlGn', annot=True, square=True, mask=mask)

plt.show()
```

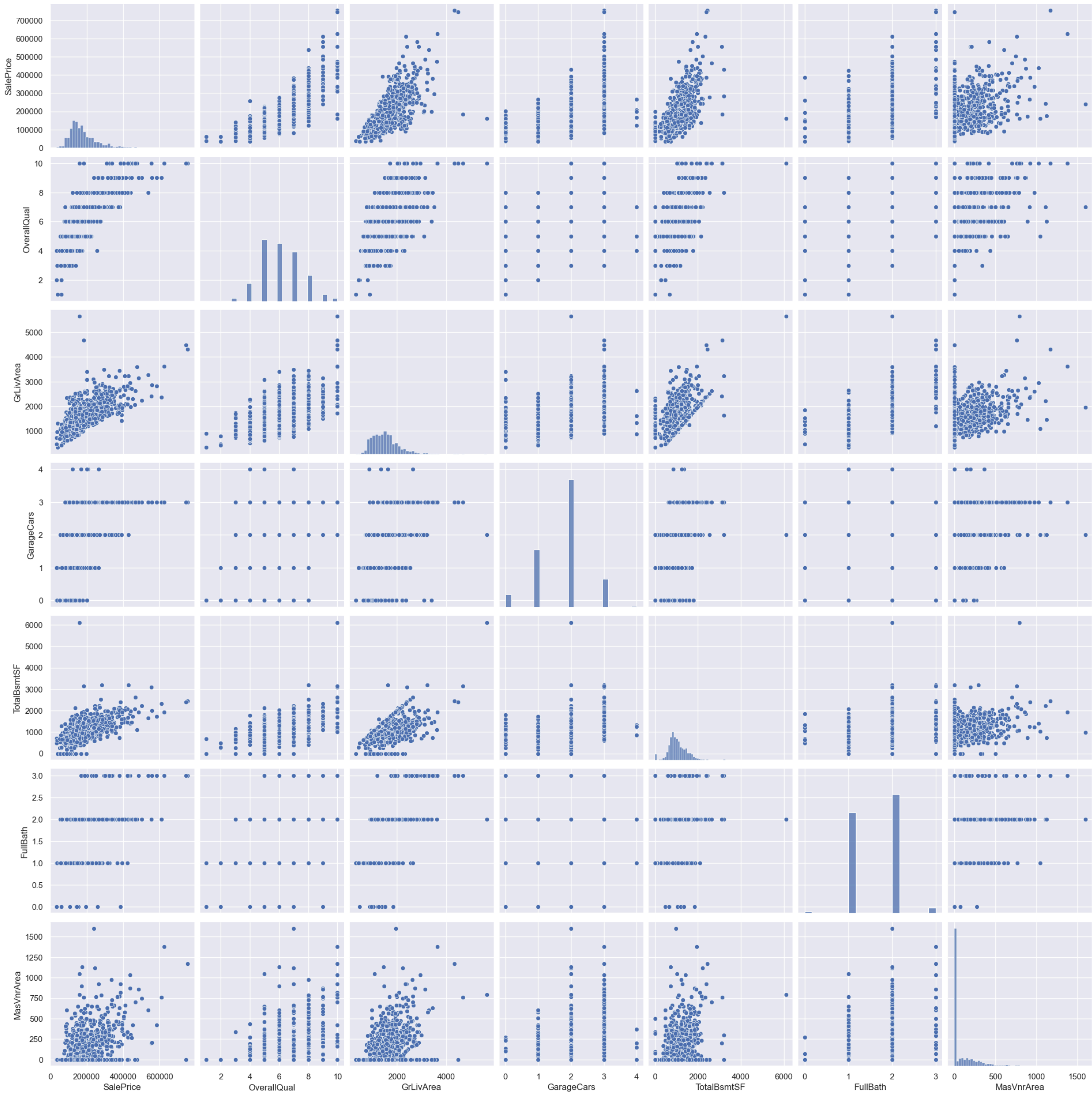


In [39]:
```
plt.figure(figsize=[20,10])
```

```
k = 20 # number of variables for a heatmap
cols = corr.nlargest(k,'SalePrice')['SalePrice'].index
corrmatrix = np.corrcoef(df[cols].values.T)
sns.set(font_scale=1)
hm = sns.heatmap(corrmatrix, cbar=True, annot=True, square=True, fmt='.2f', annc
plt.show()
```




```
In [40]: columns = ['SalePrice','OverallQual','GrLivArea','GarageCars','TotalBsmtSF','Ful
sns.pairplot(df[columns], size=3)
plt.show()
```



In [41]:
```
df.shape
```

Out[41]: (1460, 64)

In [42]:
```
from sklearn.preprocessing import LabelEncoder
df_categorical=df.select_dtypes(include='object').columns
lc=LabelEncoder()
for i in df_categorical:
    df[i]=lc.fit_transform(df[i])
```

In [43]:
```
df[df_categorical].head()
```

Out[43]:

	MSZoning	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition
0	3	3	3	4	0	5	
1	3	3	3	2	0	24	
2	3	0	3	4	0	5	
3	3	0	3	0	0	6	
4	3	0	3	2	0	15	

In [44]:
```
df.dtypes
```

Out[44]:
```
MSSubClass          int64
MSZoning            int32
LotArea             int64
LotShape            int32
LandContour         int32
LotConfig           int32
LandSlope           int32
Neighborhood        int32
Condition1          int32
BldgType            int32
HouseStyle          int32
OverallQual         int64
OverallCond         int64
YearBuilt           int64
YearRemodAdd        int64
RoofStyle           int32
Exterior1st         int32
Exterior2nd         int32
MasVnrArea        float64
ExterQual           int32
ExterCond           int32
Foundation          int32
BsmtQual            int32
BsmtCond            int32
BsmtExposure        int32
BsmtFinType1        int32
BsmtFinSF1          int64
BsmtFinType2        int32
BsmtFinSF2          int64
BsmtUnfSF           int64
TotalBsmtSF         int64
HeatingQC           int32
CentralAir          int32
Electrical          int32
1stFlrSF            int64
2ndFlrSF            int64
GrLivArea           int64
BsmtFullBath        int64
BsmtHalfBath        int64
FullBath            int64
HalfBath            int64
BedroomAbvGr        int64
KitchenAbvGr        int64
KitchenQual         int32
TotRmsAbvGrd        int64
Functional          int32
Fireplaces          int64
GarageType          int32
GarageYrBlt       float64
GarageFinish        int32
GarageCars          int64
GarageArea          int64
GarageQual          int32
GarageCond          int32
PavedDrive          int32
WoodDeckSF          int64
OpenPorchSF         int64
EnclosedPorch       int64
MoSold              int64
YrSold              int64
```

```
SaleType                  int32
SaleCondition             int32
SalePrice                 int64
Transformed_SalePrice   float64
dtype: object
```

In [45]:
```
df.shape
```

Out[45]: (1460, 64)

In [46]:
```
#Divide data into X and y for building the model
x=df.drop(['SalePrice','Transformed_SalePrice'], axis=1)
y=df['Transformed_SalePrice']
```

In [47]:
```
x.head()
```

Out[47]:

	MSSubClass	MSZoning	LotArea	LotShape	LandContour	LotConfig	LandSlope	Ne
0	60	3	8450	3	3	4	0	
1	20	3	9600	3	3	2	0	
2	60	3	11250	0	3	4	0	
3	70	3	9550	0	3	0	0	
4	60	3	14260	0	3	2	0	

In [48]:
```
y.head()
```

Out[48]:
```
0    12.247694
1    12.109011
2    12.317167
3    11.849398
4    12.429216
Name: Transformed_SalePrice, dtype: float64
```

In [49]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
```

In [50]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

Building base Model

In [51]:
```
from sklearn.linear_model import (
LinearRegression,Ridge,Lasso,ElasticNet,SGDRegressor,HuberRegressor
)
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import PolynomialFeatures
```

```python
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPRegressor
import lightgbm as lgb
import xgboost as xgb
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error
```

In [52]:
```python
#Define the models
models={
    'Linear Regression':LinearRegression(),
    'Robust Regression':HuberRegressor(),
    'Ridge Regression':Ridge(),
    'ElasticNet Regressor': ElasticNet(),
    'Lasso Regression':Lasso(),
    'Polynomial Regression':Pipeline([
        ('poly',PolynomialFeatures(degree=2)),
        ('linear',LinearRegression())
    ]),
    'SGD Regressor':SGDRegressor(),
    'ANN':MLPRegressor(hidden_layer_sizes=(100,),max_iter=1000),
    'Random Forest Regressor':RandomForestRegressor(),
    'Support vector Regressor':SVR(),
    'LGBM':lgb.LGBMRegressor(),
    'XGBoost':xgb.XGBRFRegressor(),
    'KNN Regressor':KNeighborsRegressor()
}
results=[]
```

In [ ]:
```python
for name,model in models.items():
    model.fit(x_train,y_train)
    y_pred=model.predict(x_test)
    y_pred_train=model.predict(x_train)
    r2_train = r2_score(y_train, y_pred_train)
    r2_test=r2_score(y_test, y_pred)
    mse_train = mean_squared_error(y_train, y_pred_train)
    mse_test=mean_squared_error(y_test, y_pred)
    mae_train=mean_absolute_error(y_train, y_pred_train)
    mae_test=mean_absolute_error(y_test, y_pred)
    results.append({
        'Name of the model':name,
        'r2_train':r2_train,
        'r2_test':r2_test,
        'mse_train':mse_train,
        'mse_test':mse_test,
        'mae_train':mae_train,
        'mae_test':mae_test
    })
df1=pd.DataFrame(results)
```

In [ ]:
```python
df1
```

To determine which model is the best, we should consider the balance between training and testing performance, the complexity of the model, and the overall error metrics. Here are the considerations for each model:

Key Metrics for Evaluation:

- **R^2 Score**: Indicates the proportion of variance explained by the model. Higher is generally better, but it should be similar for both training and test sets to avoid overfitting.
- **MSE (Mean Squared Error)**: Measures the average squared difference between predicted and actual values. Lower is better.
- **MAE (Mean Absolute Error)**: Measures the average absolute difference between predicted and actual values. Lower is better.

Model Performance Summary:

1. **Linear Regression**:
   - **R^2 Train**: 0.898227
   - **R^2 Test**: -1.153681e+22 (significantly negative)
   - **MSE Train**: 0.016442
   - **MSE Test**: 1.782565e+21
   - **MAE Train**: 0.086249
   - **MAE Test**: 3.276174e+10
   - **Conclusion**: Severe overfitting, poor generalization. Not a good model.
2. **KNeighbor Regressor**:
   - **R^2 Train**: 0.869135
   - **R^2 Test**: 0.793835
   - **MSE Train**: 0.021143
   - **MSE Test**: 0.031849
   - **MAE Train**: 0.101664
   - **MAE Test**: 0.128370
   - **Conclusion**: Balanced performance, good generalization. A good model.
3. **Support Vector Regressor**:
   - **R^2 Train**: 0.965355
   - **R^2 Test**: 0.864931
   - **MSE Train**: 0.005597
   - **MSE Test**: 0.020870
   - **MAE Train**: 0.063692
   - **MAE Test**: 0.098854
   - **Conclusion**: Slight overfitting, but overall good performance. A good model.
4. **Random Forest Regressor**:
   - **R^2 Train**: 0.980918
   - **R^2 Test**: 0.889454
   - **MSE Train**: 0.003083
   - **MSE Test**: 0.017081
   - **MAE Train**: 0.036678
   - **MAE Test**: 0.089254
   - **Conclusion**: Slight overfitting, but excellent performance. A good model.
5. **Decision Tree Regressor**:

- **R^2 Train**: 1.000000
- **R^2 Test**: 0.750913
- **MSE Train**: 0.000000
- **MSE Test**: 0.038487
- **MAE Train**: 0.000000
- **MAE Test**: 0.141652
- **Conclusion**: Severe overfitting, poor generalization. Not a good model.

Best Model Selection:

- **Random Forest Regressor**: It shows the highest R^2 on the test set, indicating the best generalization performance. Although there is slight overfitting, it is within acceptable limits, and the overall error metrics are low.
- **Support Vector Regressor**: Also a good option with balanced performance, though slightly lower than the Random Forest.

Conclusion:

The Random Forest Regressor is the best model based on the balance between training and test performance, as well as low error metrics. It shows excellent generalization and maintains low errors, making it a robust choice for your problem.

In [ ]:
```
x.columns
```

In [ ]:
```
# to build the front end i need to identify the key attributes.
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(x_train, y_train)
importances = model.feature_importances_
features = x.columns
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance': importa
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascen
```

In [ ]:
```
feature_importance_df
```

In [ ]:
```
feature_importance_df.to_csv('feature importance.csv',index=False)
```