



# ***HOUSING PRICE PREDICTION***

**Submitted by:  
Shubham Sahu**

## ***ACKNOWLEDGMENT***

I would like to thank Flip Robo Technologies for providing me with the opportunity to work on this project from which I have learned a lot. I am also grateful to Miss Khushboo Garg for her constant guidance and support.

Some of the reference sources are as follows:

- Internet
- Coding Ninjas
- Medium.com
- Analytics Vidhya
- StackOverflow

## **TABLE OF CONTENTS**

<b>ACKNOWLEDGMENT</b> .....	2
<b>INTRODUCTION</b> .....	1
BUSINESS PROBLEM FRAMING .....	1
CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM .....	1
REVIEW OF LITERATURE .....	2
MOTIVATION FOR THE PROBLEM UNDERTAKEN .....	2
<b>ANALYTICAL PROBLEM FRAMING</b> .....	3
MATHEMATICAL/ ANALYTICAL MODELING OF THE PROBLEM .....	3
DATA SOURCES AND THEIR FORMATS .....	4
DATA PREPROCESSING DONE .....	9
DATA INPUTS- LOGIC- OUTPUT RELATIONSHIPS .....	16
HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED .....	17
<b>MODEL/S DEVELOPMENT AND EVALUATION</b> .....	22
IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS) .....	22
TESTING OF IDENTIFIED APPROACHES (ALGORITHMS) .....	23
RUN AND EVALUATE SELECTED MODELS .....	24
KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION .....	26
VISUALIZATIONS .....	26
INTERPRETATION OF THE RESULTS .....	69
<b>CONCLUSION</b> .....	70
KEY FINDINGS AND CONCLUSIONS OF THE STUDY .....	70
LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE ...	70
LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK .....	71

## ***INTRODUCTION***

### ***BUSINESS PROBLEM FRAMING***

This is a real estate problem where a US based housing company named Surprise Housing has decided to invest in Australian Market. Their agenda is to buy houses in Australia at prices below their actual value in the market and sell them at high prices to gain profit. To do this this company uses data analytics to decide in which property they must invest.

Company has collected the data of previously sold houses in Australia and with the help of this data they want to know to the value of prospective properties to decide whether it will suitable to invest in the properties or not.

To know the value of properties company has provided data to us to do data analysis and to extract the information of attributes which are important to predict the price of the houses. They want a machine learning model which can predict the price of houses and also the significance of each important attribute in house prediction i.e, how and to what intensity each variable impacts the price of the house.

### ***CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM***

In real estate the value of property usually increases with time as seen in many countries. One of the causes for this is due to rising population.

The value of property also depends on the proximity of the property, its size its neighbourhood and audience for which the property is subjected to be sold. For example if audience is mainly concerned of commercial purpose. Then the property which is located in densely populated area will be sold very fast and at high prices compared to the one located at remote place. Similarly if audience is concerned only on living place then property with less dense area having large area with all services will be sold at higher prices.

The company is looking at prospective properties to buy houses to enter the market. We are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not.

## ***REVIEW OF LITERATURE***

Houses are one of the necessary needs of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of the major contributors in the world's economy.

A US-based housing company named Surprise Housing has decided to enter the Australian market. The company uses data analytics to purchase houses at a price below their actual values and flip them at a higher price.

We are required to build a model using Machine Learning in order to predict the actual value of the prospective properties and decide whether to invest in them or not.

With its great weather, cosmopolitan cities, diverse natural landscapes and relaxed lifestyle, it's no wonder that Australia remains a top pick for expats.

Living cost in Australia for one person: \$2,835 per month. Average living expenses for a couple: \$4,118 per month. Average monthly living expenses for a family of 4: \$5,378. Australia currently has the 16th highest cost of living in the world, with the USA and UK well behind at 21st and 33rd place respectively. Sydney and Melbourne are popular choices for expats moving to Australia. House pricing in some of the top Australian cities:-

Sydney - median house price A\$1,142,212

Adelaide- median house price A\$542,947

Hobart (smaller city)- median house price A\$530,570.

## ***MOTIVATION FOR THE PROBLEM UNDERTAKEN***

To understand real world problems where Machine Learning and Data Analysis can be applied to help organizations in various domains to make better decisions with the help of which they can gain profit or can be escaped from any loss which otherwise could be possible without the study of data. One of such domain is Real Estate.

Houses are one of the necessary need of each and every person around the globe and therefore housing and real estate market is one of the markets which is one of

the major contributors in the world's economy. It is a very large market and there are various companies working in the domain. Data science comes as a very important tool to solve problems in the domain to help the companies increase their overall revenue, profits, improving their marketing strategies and focusing on changing trends in house sales and purchases. Predictive modelling, Market mix modelling, recommendation systems are some of the machine learning techniques used for achieving the business goals for housing companies. Our problem is related to one such housing company.

## ***ANALYTICAL PROBLEM FRAMING***

### ***MATHEMATICAL/ ANALYTICAL MODELING OF THE PROBLEM***

In this project we have performed various mathematical and statistical analysis such as we checked description or statistical summary of the data using describe, checked correlation using corr and also visualized it using heatmap. Then we have used Z-Score to plot outliers and remove them.

In [18]: *# Let's check the statistical summary of our dataset*

```
housing_train.describe()
```

Out[18]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2
count	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000	1168.000000
mean	724.136130	56.767979	70.807363	10484.749144	6.104452	5.595890	1970.930651	1984.758562	101.696918	444.726027	46.647260
std	416.159877	41.940650	22.440317	8957.442311	1.390153	1.124343	30.145255	20.785185	182.218483	462.664785	163.520016
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1875.000000	1950.000000	0.000000	0.000000	0.000000
25%	360.500000	20.000000	60.000000	7621.500000	5.000000	5.000000	1954.000000	1966.000000	0.000000	0.000000	0.000000
50%	714.500000	50.000000	70.000000	9522.500000	6.000000	5.000000	1972.000000	1983.000000	0.000000	385.500000	0.000000
75%	1079.500000	70.000000	79.250000	11515.500000	7.000000	6.000000	2000.000000	2004.000000	160.000000	714.500000	0.000000
max	1460.000000	190.000000	313.000000	164660.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	1474.000000

From this statistical analysis we make some of the interpretations that,

- Maximum standard deviation of 8957.44 is observed in LotArea column.
- Maximum SalePrice of a house observed is 755000 and minimum is 34900.

- In the columns Id, MSSubclass, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfsF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, HalfBath, TotRmsAbvGrd, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, Miscval, salePrice mean is considerably greater than median so the columns are positively skewed.
- In the columns FullBath, BedroomAbvGr, Fireplaces, Garagecars, GarageArea, YrSold Median is greater than mean so the columns are negatively skewed.
- In the columns Id, MSSubClass, LotFrontage, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtHalfBath, BedroomAbvGr, ToRmsAbvGrd, GarageArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal, SalePrice there is considerable difference between the 75 percentile and maximum so outliers are present.

## ***DATA SOURCES AND THEIR FORMATS***

The variable features of this problem statement are as :

MSSubClass: Identifies the type of dwelling involved in the sale

MSZoning: Identifies the general zoning classification of the sale

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

Alley: Type of alley access to property

LotShape: General shape of property

LandContour: Flatness of the property

Utilities: Type of utilities available

LotConfig: Lot configuration

LandSlope: Slope of property

Neighborhood: Physical locations within Ames city limits

Condition1: Proximity to various conditions

Condition2: Proximity to various conditions (if more than one is present)

BldgType: Type of dwelling

HouseStyle: Style of dwelling

OverallQual: Rates the overall material and finish of the house

OverallCond: Rates the overall condition of the house

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

RoofMatl: Roof material

Exterior1st: Exterior covering on house

Exterior2nd: Exterior covering on house (if more than one material)

MasVnrType: Masonry veneer type

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

ExterCond: Evaluates the present condition of the material on the exterior

Foundation: Type of foundation

BsmtQual: Evaluates the height of the basement

BsmtCond: Evaluates the general condition of the basement

BsmtExposure: Refers to walkout or garden level walls



BsmtFinType1: Rating of basement finished area

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

HeatingQC: Heating quality and condition

CentralAir: Central air conditioning

Electrical: Electrical system

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

GarageType: Garage location

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

GarageCars: Size of garage in car capacity

GarageArea: Size of garage in square feet

GarageQual: Garage quality

GarageCond: Garage condition

PavedDrive: Paved driveway

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

Fence: Fence quality

MiscFeature: Miscellaneous feature not covered in other categories

MiscVal: \$Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

## SaleCondition: Condition of sale

In [7]: *# Let's check the information of our dataset*

```
housing_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 1168 entries, 0 to 1167
```

```
Data columns (total 81 columns):
```

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

```

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

```

```
In [6]: # Let's check the data types of our columns
```

```
housing_train.dtypes
```

```

Out[6]: Id                int64
MSSubClass                int64
MSZoning                  object
LotFrontage               float64
LotArea                   int64
...
MoSold                   int64
YrSold                   int64
SaleType                 object
SaleCondition            object
SalePrice                 int64
Length: 81, dtype: object

```

## **DATA PREPROCESSING DONE**

After loading all the required libraries we loaded the data into our jupyter notebook.

```

In [1]: # Let's import all the required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
pd.pandas.set_option('display.max_columns',None)

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy import stats

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression,Lasso,Ridge,Elastic
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.model_selection import GridSearchCV

#importing warnings
import warnings
warnings.filterwarnings('ignore')

```

```

In [2]: # Let's Load our dataset

housing_train=pd.read_csv("Housing train.csv")
housing_train

```

```

Out[2]:

```

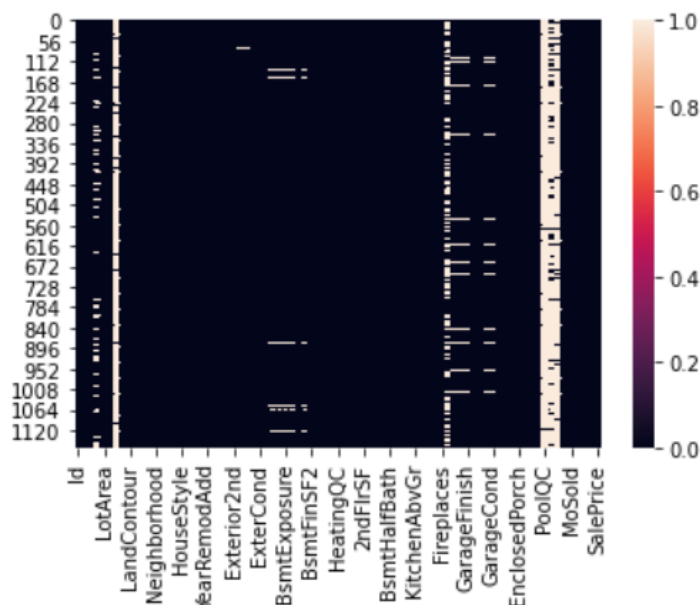
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	L
0	127	120	RL	NaN	4928	Pave	NaN	IR1	
1	889	20	RL	95.0	15865	Pave	NaN	IR1	
2	793	60	RL	92.0	9920	Pave	NaN	IR1	
3	110	20	RL	105.0	11751	Pave	NaN	IR1	
4	422	20	RL	NaN	16635	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	

Feature Engineering has been used for cleaning of the data. Some unused columns have been deleted and even some columns have been bifurcated which was used in the prediction. We first done data cleaning. We first looked percentage of values missing in columns then we imputed missing values.

```
In [10]: # Let's check the missing values of top 30 columns
housing_train.isnull().sum().sort_values(ascending = False).head(30)
```

```
Out[10]: PoolQC          1161
MiscFeature       1124
Alley             1091
Fence             931
FireplaceQu       551
LotFrontage       214
GarageType         64
GarageCond         64
GarageYrBlt        64
GarageFinish       64
GarageQual         64
BsmtExposure       31
BsmtFinType2       31
BsmtFinType1       30
BsmtCond           30
BsmtQual           30
MasVnrArea         7
MasVnrType         7
Exterior2nd        0
Exterior1st        0
OverallCond        0
ExterQual          0
ExterCond          0
Foundation         0
RoofMatl           0
```

```
In [12]: # Let's plot the heat map for our missing values
sns.heatmap(housing_train.isnull());
```



In [13]: *# Let's check the percentage of missing values of each column*

```
def missing_values_table(housing_train):
    mis_val = housing_train.isnull().sum()
    mis_val_percent = 100 * housing_train.isnull().sum() / len(housing_train)
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
    mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})
    mis_val_table_ren_columns = mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
    print ("Your selected dataframe has " + str(housing_train.shape[1]) + " columns.\n"
          "There are " + str(mis_val_table_ren_columns.shape[0]) +
          " columns that have missing values.")
    return mis_val_table_ren_columns
missing_values_table(housing_train)
```

Your selected dataframe has 81 columns.  
There are 18 columns that have missing values.

Out[13]:

	Missing Values	% of Total Values
PoolQC	1161	99.4
MiscFeature	1124	96.2
Alley	1091	93.4
Fence	931	79.7
FireplaceQu	551	47.2

Out[13]:

	Missing Values	% of Total Values
PoolQC	1161	99.4
MiscFeature	1124	96.2
Alley	1091	93.4
Fence	931	79.7
FireplaceQu	551	47.2
LotFrontage	214	18.3
GarageType	64	5.5
GarageYrBlt	64	5.5
GarageFinish	64	5.5
GarageQual	64	5.5
GarageCond	64	5.5
BsmtExposure	31	2.7
BsmtFinType2	31	2.7
BsmtCond	30	2.6
BsmtFinType1	30	2.6
BsmtQual	30	2.6
MasVnrArea	7	0.6
MasVnrType	7	0.6

```
In [14]: # Let's fill the missing values in categorical columns as NA
```

```
columns = ["FireplaceQu", "GarageType", "GarageFinish", "GarageQual", "GarageCond", "BsmtExposure", "BsmtFinType2", "BsmtCond", '
housing_train[columns] = housing_train[columns].fillna('NA')
```

```
In [15]: # Let's fill the missing values in MasVnrType with None
```

```
housing_train['MasVnrType'] = housing_train['MasVnrType'].fillna('None')
```

```
In [16]: # Let's fill the missing values in GarageYrBlt with 0
```

```
housing_train['GarageYrBlt'] = housing_train['GarageYrBlt'].fillna('0')
```

```
In [17]: # Let's Imputing the missing values and replace it with the median
```

```
housing_train['LotFrontage'].fillna(housing_train['LotFrontage'].median(),inplace=True)
housing_train['MasVnrArea'].fillna(housing_train['MasVnrArea'].median(),inplace=True)
```

```
In [8]: # Let's explore the categorical columns
```

```
for column in housing_train.columns:
    if housing_train[column].dtypes == object:
        print(str(column) + ' : ' + str(housing_train[column].unique())
        print(housing_train[column].value_counts())
        print('\n')
```

```
MSZoning : ['RL' 'RM' 'FV' 'RH' 'C (all)']
```

```
RL          928
```

```
RM          163
```

```
FV           52
```

```
RH           16
```

```
C (all)       9
```

```
Name: MSZoning, dtype: int64
```

```
Street : ['Pave' 'Grv1']
```

```
Pave       1164
```

```
Grv1         4
```

```
Name: Street, dtype: int64
```

```
Alley : [nan 'Grv1' 'Pave']
```

```
Grv1       41
```

```
Pave       36
```

```
Name: Alley, dtype: int64
```

We observed that there is only one unique value present in Utilities so will be dropping this column. Then we encoded all the categorical columns into numerical columns using dummy variables.



```

In [8]: # Let's explore the categorical columns

for column in housing_train.columns:
    if housing_train[column].dtypes == object:
        print(str(column) + ' : ' + str(housing_train[column].unique()))
        print(housing_train[column].value_counts())
        print('\n')

```

Name: mszoning, dtype: int64

Street : ['Pave' 'Grv1']  
Pave 1164  
Grv1 4  
Name: Street, dtype: int64

Alley : [nan 'Grv1' 'Pave']  
Grv1 41  
Pave 36  
Name: Alley, dtype: int64

LotShape : ['IR1' 'Reg' 'IR2' 'IR3']  
Reg 740  
IR1 390  
IR2 32  
IR3 6

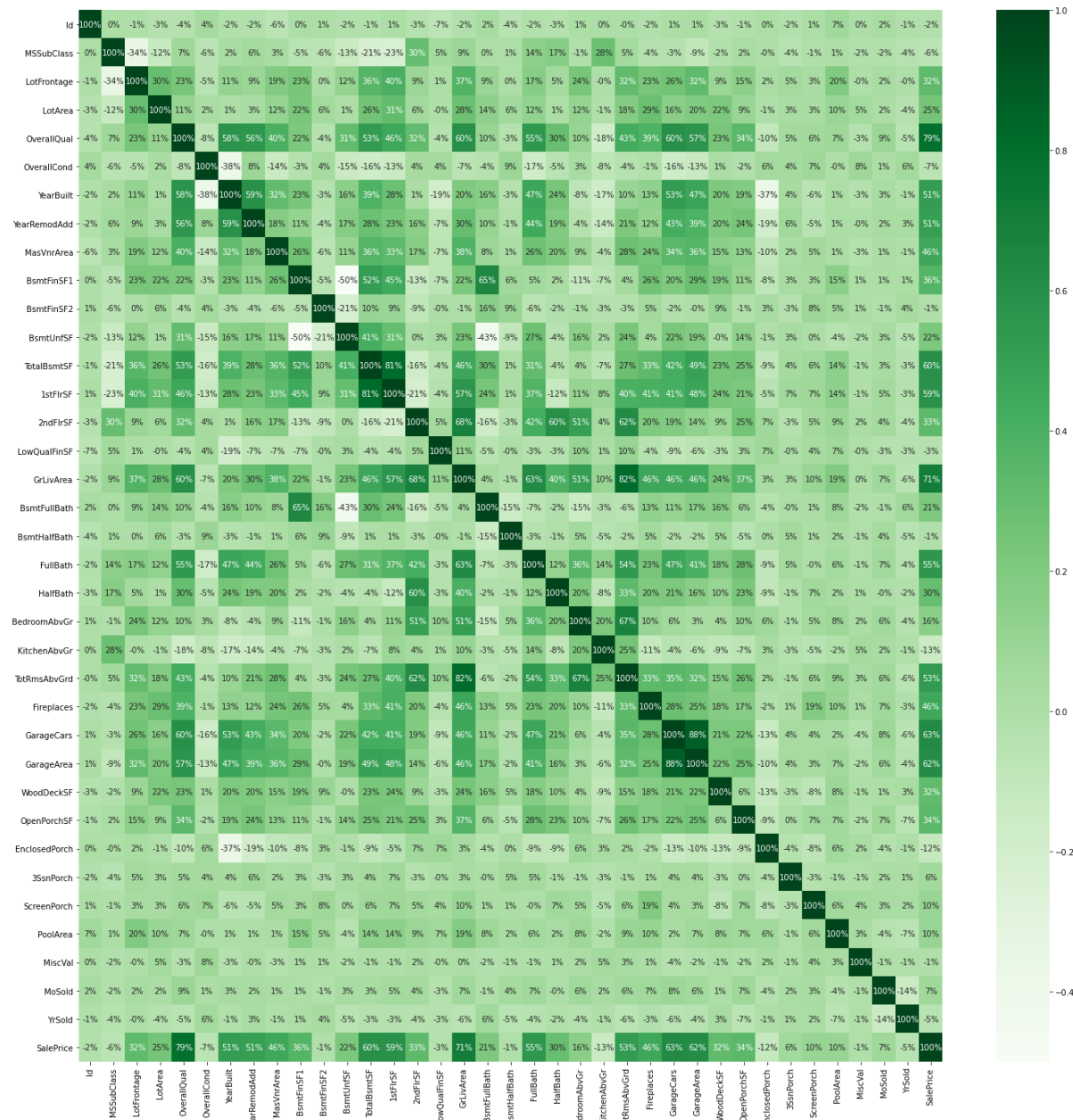
Then we checked the correlation with the help of heatmap.

```

In [20]: # Let's plot the heat map

plt.figure(figsize=(24,24))
sns.heatmap(housing_train_cor,annot=True,fmt='.0%',cmap='Greens')
plt.show()

```



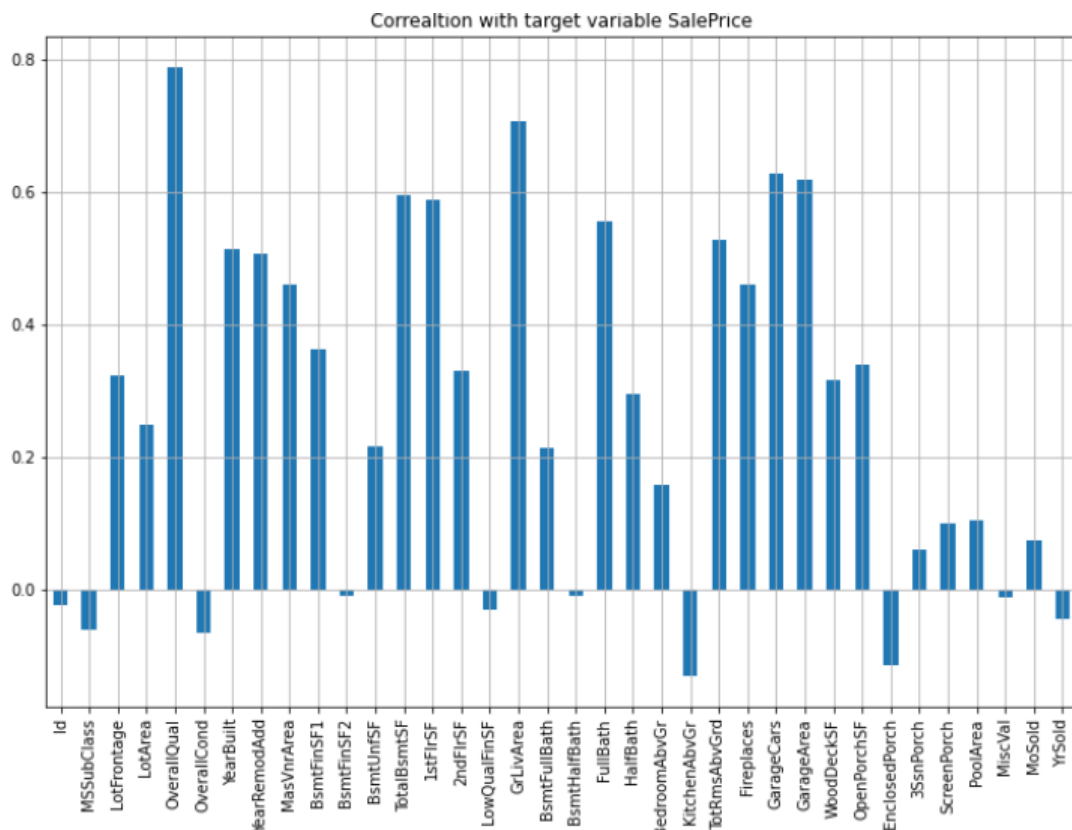
While checking the heatmap of correlation we observed that:

- SalePrice is highly positively correlated with the columns OverallQual, YearBuilt, YearRemodAdd, TotalBsmntSF, 1stFlrSF, GrLivArea, FullBath, TotRmsAbvGrd, GarageCars, GarageArea.
- SalePrice is negatively correlated with OverallCond, KitchenAbvGr, EnclosedPorch, YrSold.
- We observe multicollinearity in between columns so we will be using Principal Component Analysis(PCA).

- No correlation has been observed between the column Id and other columns so we will be dropping this column.

## DATA INPUTS- LOGIC- OUTPUT RELATIONSHIPS

Here we check the correlation



between all our feature variables with target variable label

```
In [21]: # Let's check the correlation with target variable 'SalePrice'

plt.figure(figsize=(12,8))
housing_train.drop('SalePrice', axis=1).corrwith(housing_train['SalePrice']).plot(kind='bar',grid=True)
plt.xticks(rotation='vertical')
plt.title("Correaltion with target variable SalePrice");
```

1. The column OverallQual is most positively correlated with SalePrice.
2. The column KitchenAbvGrd is most negatively correlated with SalePrice.

Set of assumptions related to the problem under consideration

By looking into the target variable label we assumed that it was a Regression type of problem.

We observed multicollinearity in between columns so we assumed that we will be using Principal Component Analysis (PCA).

We also observed that only one single unique value was present in Utilities column so we assumed that we will be dropping these columns.

## ***HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED***

### ***HARDWARE:***

[View basic information about your computer](#)

Windows edition

Windows 10 Home Single  
Language

© 2019 Microsoft Corporation.  
All rights reserved.



Windows 10

System

Processor:	Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz
Installed memory (RAM):	8.00 GB (7.80 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	Shubham
Full computer name:	Shubham
Computer description:	
Workgroup:	WORKGROUP

[Change settings](#)

Windows activation

Windows is activated [Read the Microsoft Software License Terms](#)

Product ID: 00327-60000-00000-AA367

[Change product key](#)

### ***SOFTWARE:***

Jupyter Notebook (Anaconda 3) – Python 3.7.6

Microsoft Excel 2010

### **LIBRARIES:**

The tools, libraries and packages we used for accomplishing this project are pandas, numpy, matplotlib, seaborn, scipy stats, sklearn.decomposition pca, sklearn standardscaler, GridSearchCV, joblib.

```
In [1]: # Let's import all the required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
pd.pandas.set_option('display.max_columns',None)

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy import stats

from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV,cross_val_score
from sklearn.model_selection import GridSearchCV
```

### ***From sklearn.preprocessing import StandardScaler***

As these columns are different in scale, they are standardized to have common scale while building machine learning model. This is useful when you want to compare data that correspond to different units.

***from sklearn.preprocessing import Label Encoder***

Label Encoder and One Hot Encoder. These two encoders are parts of the SciKit Learn library in Python, and they are used to convert categorical data, or text data, into numbers, which our predictive models can better understand.

***from sklearn.model\_selection import train\_test\_split, cross\_val\_score***

Train\_test\_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually. By default, Sklearn train\_test\_split will make random partitions for the two subsets.

Through pandas library we loaded our csv file 'Data file' into dataframe and performed data manipulation and analysis.

With the help of numpy we worked with arrays.

With the help of matplotlib and seaborn we did plot various graphs and figures and done data visualization.

With scipy stats we treated outliers through winsorization technique.

With sklearn.decomposition's pca package we reduced the number of feature variables from 256 to 100 by plotting scree plot with their Eigenvalues and chose the number of columns on the basis of their nodes.

With sklearn's standard scaler package we scaled all the feature variables onto single scale.

## MODEL TRAINING

```
In [64]: housing_train_x=housing_train_cap.drop(columns=['SalePrice'],axis=1)
y=housing_train_cap['SalePrice']
```

```
In [65]: #Scaling input variables

sc=StandardScaler()
x=sc.fit_transform(housing_train_x)
x=pd.DataFrame(x,columns=housing_train_x.columns)
```

## PCA

```
In [66]: # Let's explore the PCA

covar_matrix = PCA(n_components = len(x.columns))
covar_matrix.fit(x)
```

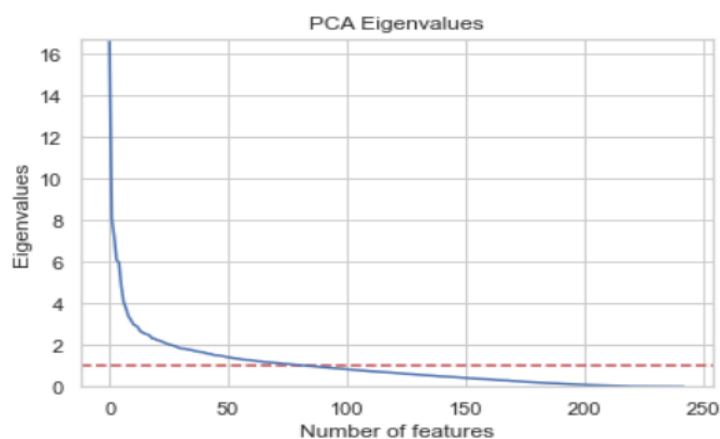
```
Out[66]: PCA(n_components=243)
```

```
In [67]: # Let's plot the PCA componenets

plt.ylabel('Eigenvalues')
plt.xlabel('Number of features')
plt.title('PCA Eigenvalues')
plt.ylim(0,max(covar_matrix.explained_variance_))
plt.style.context('seaborn-whitegrid')
plt.axhline(y=1, color='r', linestyle='--')
```

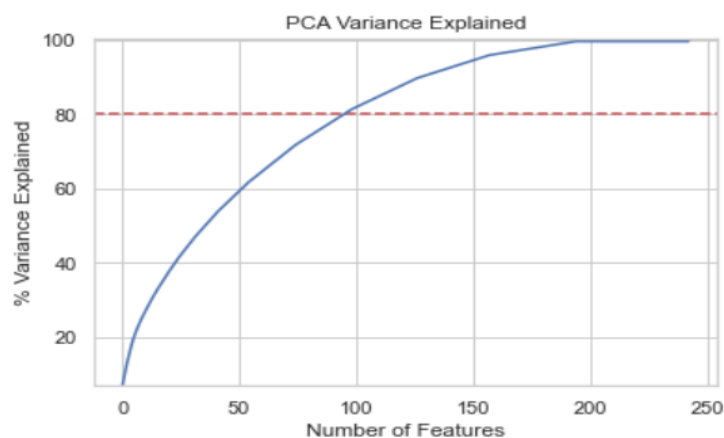
```
In [67]: # Let's plot the PCA componenets

plt.ylabel('Eigenvalues')
plt.xlabel('Number of features')
plt.title('PCA Eigenvalues')
plt.ylim(0,max(covar_matrix.explained_variance_))
plt.style.context('seaborn-whitegrid')
plt.axhline(y=1, color='r', linestyle='--')
plt.plot(covar_matrix.explained_variance_)
plt.show()
```



```
In [68]: variance = covar_matrix.explained_variance_ratio_
var=np.cumsum(np.round(covar_matrix.explained_variance_ratio_, decima

plt.ylabel('% Variance Explained')
plt.xlabel('Number of Features')
plt.title('PCA Variance Explained')
plt.ylim(min(var),100.5)
plt.style.context('seaborn-whitegrid')
plt.axhline(y=80, color='r', linestyle='--')
plt.plot(var)
plt.show()
```



```
In [69]: pca=PCA(n_components=90)
xpca=pca.fit_transform(x)
x=xpca
```

```
In [70]: pd.DataFrame(data=x)
```

Out[70]:

	0	1	2	3	4	5	6	
0	0.024209	-1.896947	0.132640	0.813270	-2.206811	-1.804833	1.036208	1.1
1	-2.247517	-4.219125	2.434139	2.469253	5.428170	2.217708	4.360840	-0.5
2	-3.177182	-0.067218	0.034345	-0.530133	1.284218	-2.884045	1.488233	0.1
3	-2.108238	-3.530568	1.215632	2.012254	1.144286	0.329085	-3.080266	-0.1
4	-3.131157	-1.375629	0.344610	1.784063	0.114215	-0.337610	-0.860078	1.6
...	...	...	...	...	...	...	...	...
1163	3.795608	-2.918561	-1.472008	-0.273291	-2.503337	0.282884	-1.206214	-0.2
1164	4.015034	2.373341	10.993851	-4.930151	-3.243407	0.557196	0.472869	-1.4
1165	0.639942	-1.219614	-0.937151	-1.445215	-1.285738	-5.676654	0.848904	3.3
1166	6.935130	2.136400	-2.252290	-2.371354	2.506539	1.338418	-0.222883	-0.6
1167	-3.748656	1.997020	-0.459500	-0.736154	-0.689951	-2.325993	1.362231	-1.7

1168 rows × 90 columns



***from sklearn.linear\_model import LogisticRegression***

The library sklearn can be used to perform logistic regression in a few lines as shown using the LogisticRegression class. It also supports multiple features. It requires the input values to be in a specific format hence they have been reshaped before training using the fit method.

***from sklearn.tree import DecisionTreeClassifier***

Decision Tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and number of attributes in the given data. The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy

***from sklearn.ensemble import RandomForestClassifier***

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max\_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Through GridSearchCV we were able to find the right parameters for hyperparameter tuning. Through joblib we saved our model in csv format.

## ***MODEL/S DEVELOPMENT AND EVALUATION***

### ***IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS)***

We first converted all our categorical variables to numeric variables with the help of dummy variables to checkout and dropped the columns which we felt were unnecessary.

We observed skewness in data so we tried to remove the skewness through treating outliers with winsorization technique.

The data was improper scaled so we scaled the feature variables on a single scale using sklearn's StandardScaler package.

There were too many (256) feature variables in the data so we reduced it to 100 with the help of Principal Component Analysis(PCA) by plotting Eigenvalues and taking the number of nodes as our number of feature variables.

### ***TESTING OF IDENTIFIED APPROACHES (ALGORITHMS)***

The algorithms we used for the training and testing are as follows:-

- Linear Regression
- Lasso
- Ridge
- Elastic Net
- SVR
- KNeighbors Regressor
- Decision Tree Regressor
- Random Forest Regressor
- Ada Boost Regressor
- Gradient Boosting Regressor

```
In [73]: model=[LinearRegression(),  
                DecisionTreeRegressor(),  
                KNeighborsRegressor(),  
                SVR(),  
                Lasso(),  
                Ridge(),  
                ElasticNet(),  
                RandomForestRegressor(),  
                AdaBoostRegressor(),  
                GradientBoostingRegressor()]
```

## RUN AND EVALUATE SELECTED MODELS

```
In [73]: model=[LinearRegression(),
                DecisionTreeRegressor(),
                KNeighborsRegressor(),
                SVR(),
                Lasso(),
                Ridge(),
                ElasticNet(),
                RandomForestRegressor(),
                AdaBoostRegressor(),
                GradientBoostingRegressor()
            ]
for m in model:
    m.fit(x_train,y_train)
    print('score of',m,'is:',m.score(x_train,y_train))
    predm=m.predict(x_test)
    print('Error:')
    print('Mean absolute error:',mean_absolute_error(y_test,predm))
    print('Mean squared error:',mean_squared_error(y_test,predm))
    print('Root Mean Squared Error:',np.sqrt(mean_squared_error(y_test,predm)))
    print("r2_score:",r2_score(y_test,predm))
    print('*****')
    print('\n')
```

```
score of LinearRegression() is: 0.8228495368700252
Error:
Mean absolute error: 21805.768654407417
Mean squared error: 1050342129.3284745
Root Mean Squared Error: 32408.982232221897
r2_score: 0.8399373085177295
*****
```

```
score of DecisionTreeRegressor() is: 1.0
Error:
Mean absolute error: 31359.418803418805
Mean squared error: 1874550145.2564104
Root Mean Squared Error: 43296.075402470495
r2_score: 0.7143354215830102
*****
```

```
score of KNeighborsRegressor() is: 0.7907231497562741
Error:
Mean absolute error: 26583.544444444447
Mean squared error: 1539525411.2545302
Root Mean Squared Error: 39236.78645422596
r2_score: 0.7653901771146742
*****
```

```
score of SVR() is: -0.045684746681192934
Error:
Mean absolute error: 58256.37313723461
Mean squared error: 6883587037.077791
Root Mean Squared Error: 82967.38538171364
r2_score: -0.04899673872128396
*****
```

```
score of Lasso() is: 0.8228495270862598
Error:
Mean absolute error: 21802.83997938824
Mean squared error: 1050198314.0557423
Root Mean Squared Error: 32406.76339987908
r2_score: 0.8399592246715113
*****
```

```
score of Ridge() is: 0.8228494764569273
Error:
Mean absolute error: 21798.74752559169
Mean squared error: 1050034922.0479769
Root Mean Squared Error: 32404.242346457922
r2_score: 0.8399841241435969
*****
```

```
score of ElasticNet() is: 0.8157053308542571
Error:
Mean absolute error: 20530.56921156668
Mean squared error: 1042532743.7144129
Root Mean Squared Error: 32288.27563860314
r2_score: 0.8411273886309673
*****
```

```
score of RandomForestRegressor() is: 0.9675889141114626
Error:
Mean absolute error: 21720.597521367523
Mean squared error: 1132352401.6918838
Root Mean Squared Error: 33650.44430155245
r2_score: 0.8274396807856363
*****
```

```
score of AdaBoostRegressor() is: 0.8284721097456915
Error:
Mean absolute error: 31348.140379197303
Mean squared error: 1712251988.095036
Root Mean Squared Error: 41379.366695190445
r2_score: 0.7390682006770667
*****
```

```
score of GradientBoostingRegressor() is: 0.9723064213023661
Error:
Mean absolute error: 21353.071870540363
Mean squared error: 1004719535.3991446
Root Mean Squared Error: 31697.311169863362
r2_score: 0.8468897814052067
*****
```

## **KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION**

We used the metric Root Mean Squared Error by selecting the Ridge Regressor model which was giving us best(minimum) RMSE score.

## **VISUALIZATIONS**

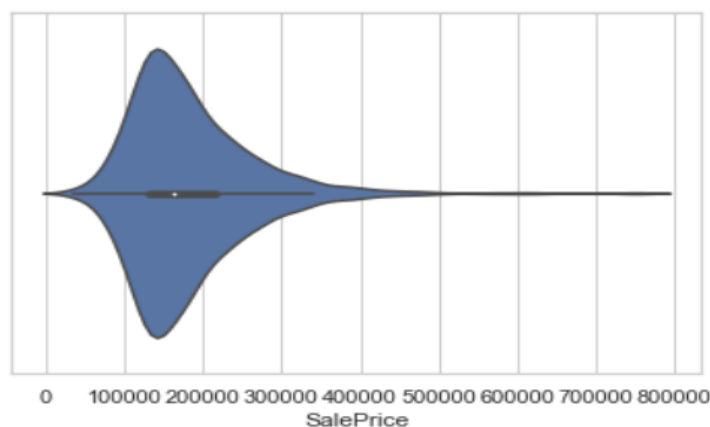
### **Data Visualization**

#### **Univariate Analysis**

```
In [22]: # Let's Check the target variable

sns.set(style='whitegrid')
sns.violinplot(housing_train['SalePrice'])
plt.show()

housing_train['SalePrice'].value_counts()
```



```
Out[22]: 140000    18
         135000    16
         155000    12
         139000    11
         160000    11
         ..
         126175     1
         204000     1
         186000     1
         369900     1
         105500     1
         Name: SalePrice, Length: 581, dtype: int64
```

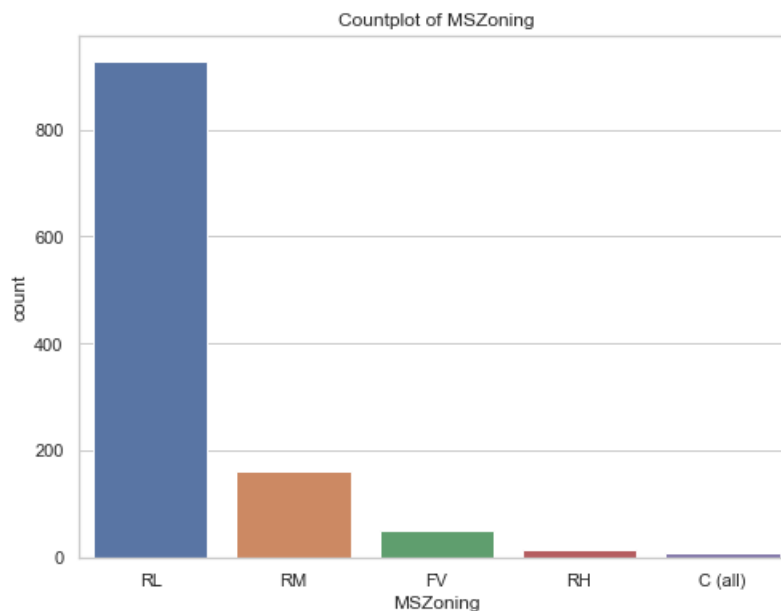
### Observation:

Maximum number of SalePrice lies between 140000 and 230000.

```
In [23]: # Let's check the column MSZoning

plt.subplots(figsize=(8,6))
sns.countplot(x="MSZoning", data=housing_train)
plt.title("Countplot of MSZoning")
plt.xlabel('MSZoning')
plt.ylabel("count")
plt.show()

housing_train['MSZoning'].value_counts()
```



```
Out[23]: RL      928
         RM      163
         FV       52
         RH       16
         C (all)    9
         Name: MSZoning, dtype: int64
```

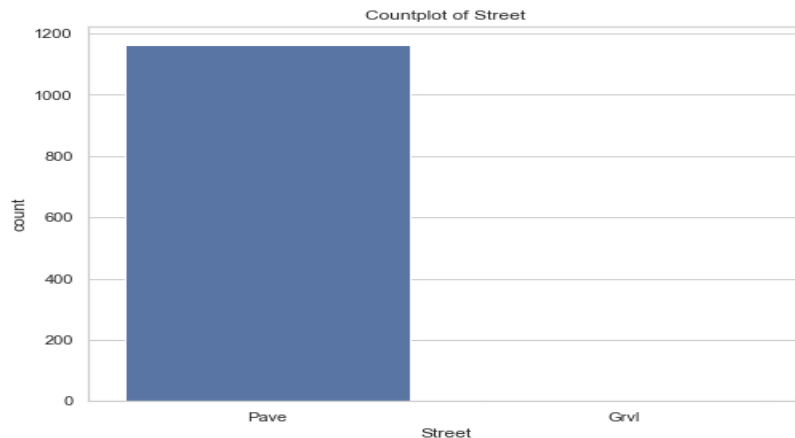
**Observation:**

Maximum, 928 number of MSZoning are RL.

In [24]: *# Let's check the column Street*

```
plt.subplots(figsize=(8,6))
sns.countplot(x="Street", data=housing_train)
plt.title("Countplot of Street")
plt.xlabel('Street')
plt.ylabel("count")
plt.show()

housing_train['Street'].value_counts()
```



```
Out[24]: Pave    1164
Grvl      4
Name: Street, dtype: int64
```

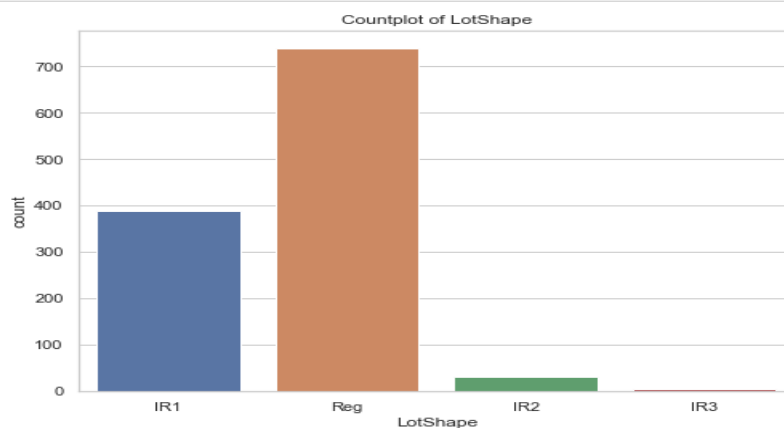
**Observation:**

Maximum, 1164 number of Street are Pave where as only 4 are Grvl.

In [25]: *# Let's check the column LotShape*

```
plt.subplots(figsize=(8,6))
sns.countplot(x="LotShape", data=housing_train)
plt.title("Countplot of LotShape")
plt.xlabel('LotShape')
plt.ylabel("count")
plt.show()

housing_train['LotShape'].value_counts()
```



```
Out[25]: Reg      740
IR1      390
IR2      32
IR3       6
Name: LotShape, dtype: int64
```

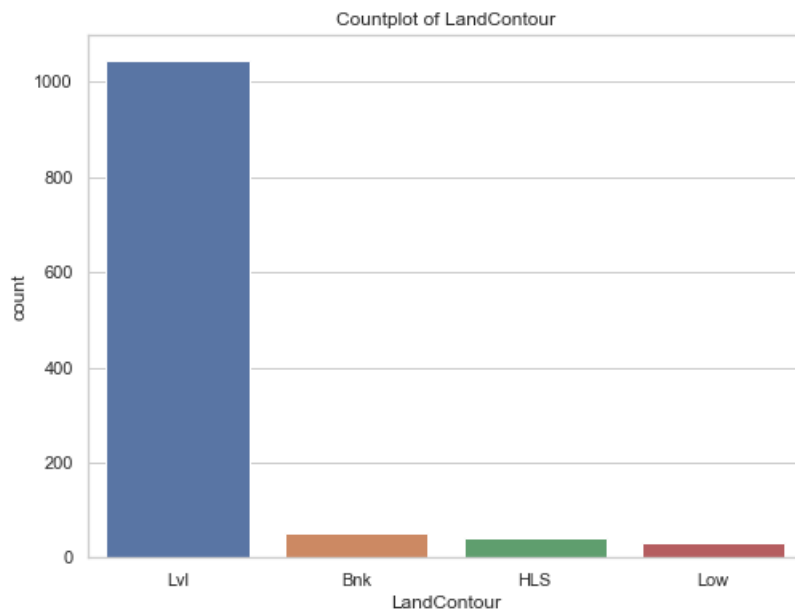
**Observation:**

Maximum, 740 number of LotShape are Reg.

```
In [26]: # Let's check the column LandContour

plt.subplots(figsize=(8,6))
sns.countplot(x="LandContour", data=housing_train)
plt.title("Countplot of LandContour")
plt.xlabel('LandContour')
plt.ylabel("count")
plt.show()

housing_train['LandContour'].value_counts()
```



```
Out[26]: Lvl      1046
         Bnk        50
         HLS        42
         Low        30
         Name: LandContour, dtype: int64
```

**Observation:**

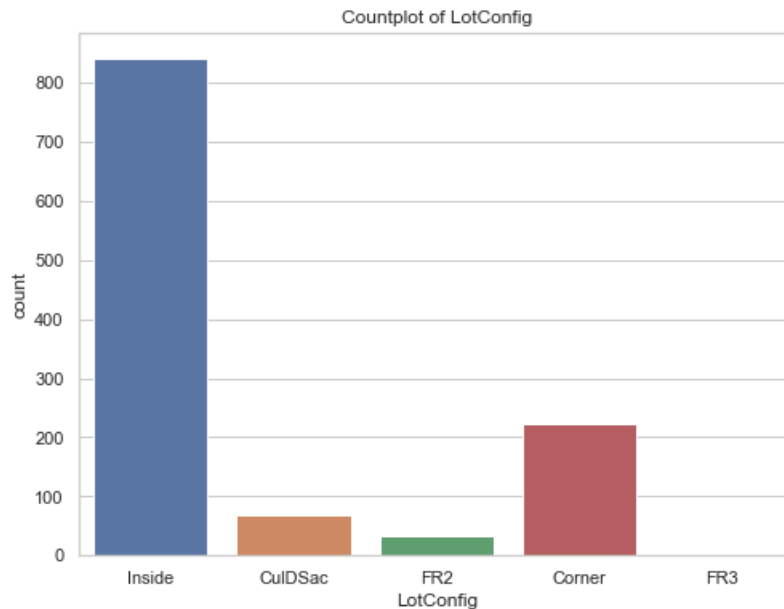
Maximum, 1046 number of LandContour are Lvl.



```
In [27]: # Let's check the column LotConfig

plt.subplots(figsize=(8,6))
sns.countplot(x="LotConfig", data=housing_train)
plt.title("Countplot of LotConfig")
plt.xlabel('LotConfig')
plt.ylabel("count")
plt.show()

housing_train['LotConfig'].value_counts()
```



```
Out[27]: Inside      842
        Corner      222
        CulDSac      69
        FR2         33
        FR3          2
        Name: LotConfig, dtype: int64
```

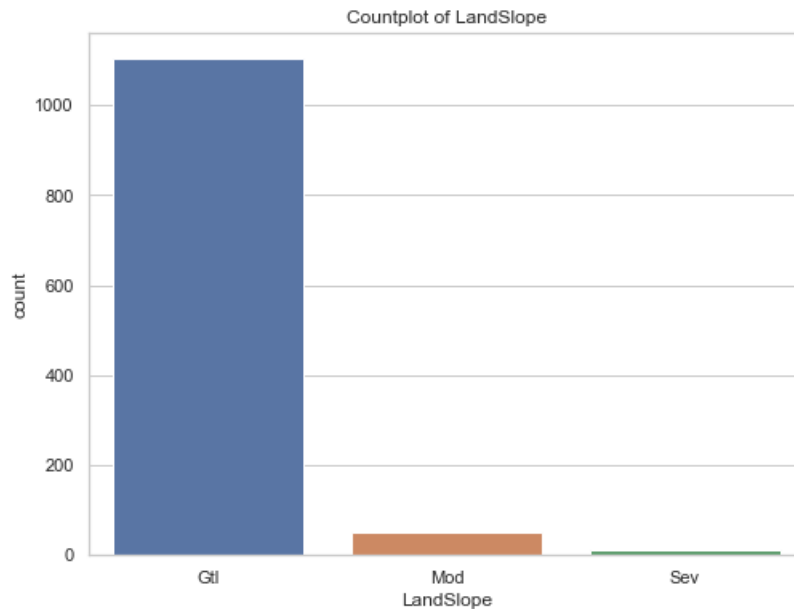
**Observation:**

Maximum, 842 number of LotConfig are Inside.

```
In [28]: # Let's check the column LandSlope

plt.subplots(figsize=(8,6))
sns.countplot(x="LandSlope", data=housing_train)
plt.title("Countplot of LandSlope")
plt.xlabel('LandSlope')
plt.ylabel("count")
plt.show()

housing_train['LandSlope'].value_counts()
```



```
Out[28]: Gtl    1105
Mod      51
Sev      12
Name: LandSlope, dtype: int64
```

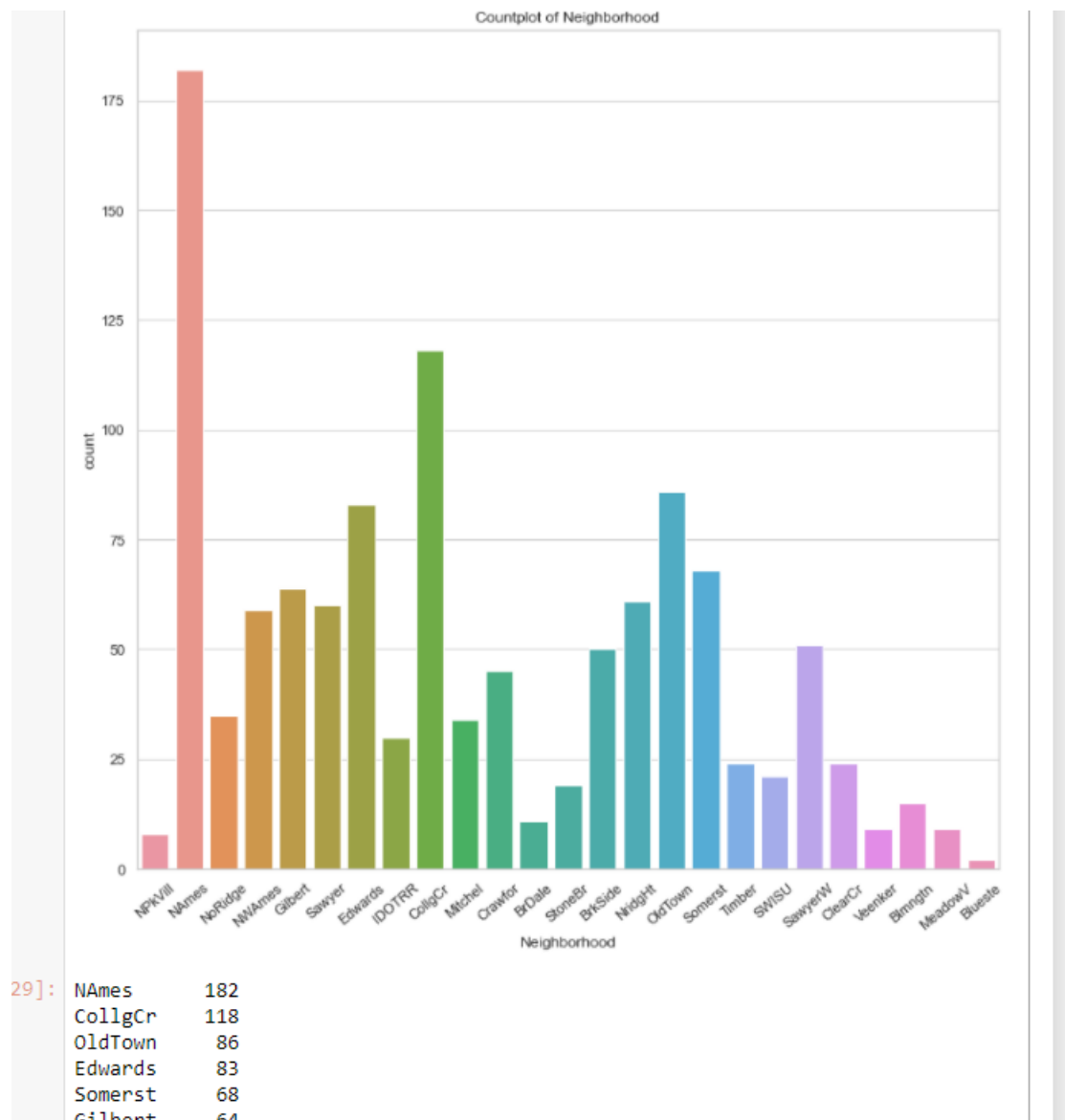
#### Observation:

Maximum, 1105 number of LandSlope are Gtl.

```
In [29]: # Let's check the column Neighborhood

plt.subplots(figsize=(12,12))
sns.countplot(x="Neighborhood", data=housing_train)
plt.title("Countplot of Neighborhood")
plt.xticks(rotation=40)
plt.xlabel('Neighborhood')
plt.ylabel("count")
plt.show()

housing_train['Neighborhood'].value_counts()
```



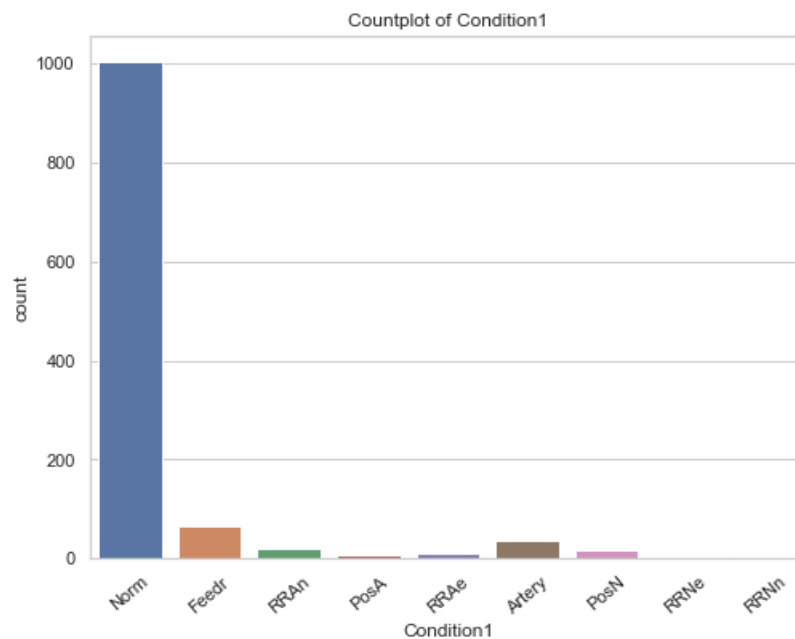
**Observation:**

Maximum, 182 number of Neighborhood are Names.

```
In [30]: # Let's check the column Condition1

plt.subplots(figsize=(8,6))
sns.countplot(x="Condition1", data=housing_train)
plt.title("Countplot of Condition1")
plt.xticks(rotation=40)
plt.xlabel('Condition1')
plt.ylabel("count")
plt.show()

housing_train['Condition1'].value_counts()
```



```
Out[30]: Norm      1005
         Feedr      67
         Artery     38
         RRAn       20
         PosN       17
         RRAe        9
```

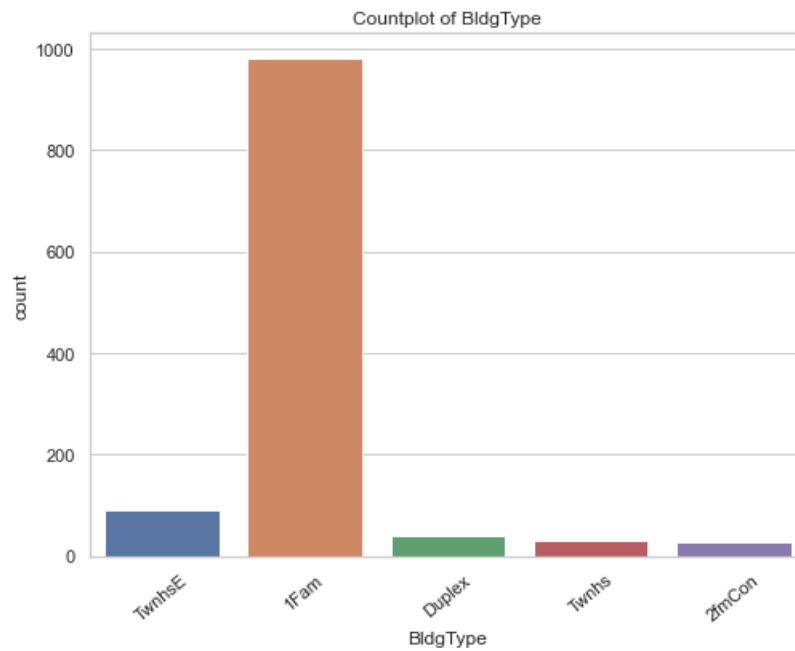
#### Observation:

Maximum, 1005 number of Condition1 is Norm.

```
In [31]: # Let's check the column BldgType

plt.subplots(figsize=(8,6))
sns.countplot(x="BldgType", data=housing_train)
plt.title("Countplot of BldgType")
plt.xticks(rotation=40)
plt.xlabel('BldgType')
plt.ylabel("count")
plt.show()

housing_train['BldgType'].value_counts()
```



```
Out[31]: 1Fam      981
Twnhse      90
Duplex      41
Twnhs       29
2fmCon       27
Name: BldgType, dtype: int64
```

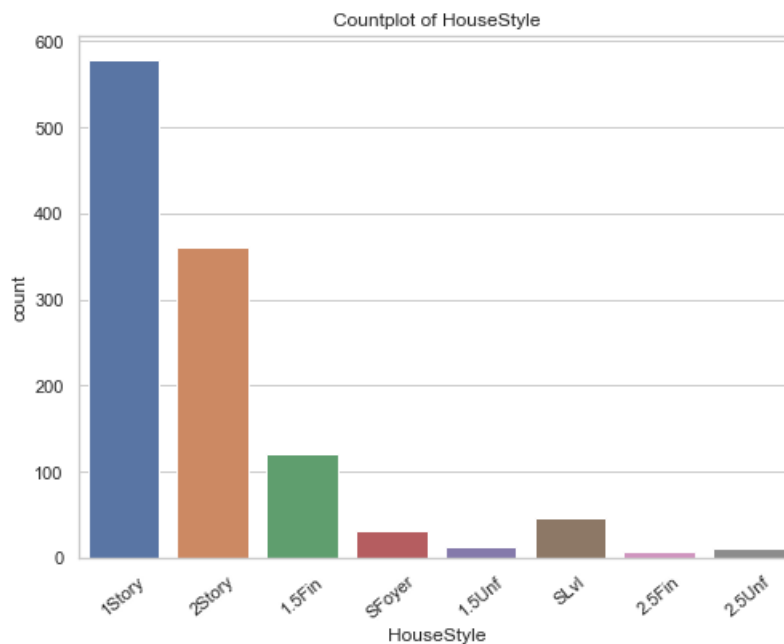
#### Observation:

Maximum, 981 number of BldgType are 1Fam.

```
In [32]: # Let's check the column HouseStyle

plt.subplots(figsize=(8,6))
sns.countplot(x="HouseStyle", data=housing_train)
plt.title("Countplot of HouseStyle")
plt.xticks(rotation=40)
plt.xlabel('HouseStyle')
plt.ylabel("count")
plt.show()

housing_train['HouseStyle'].value_counts()
```



```
Out[32]: 1Story    578
         2Story    361
         1.5Fin    121
         SLvl      47
         SFoyer     32
         1.5Unf     12
         2.5Unf     10
         2.5Fin      7
```

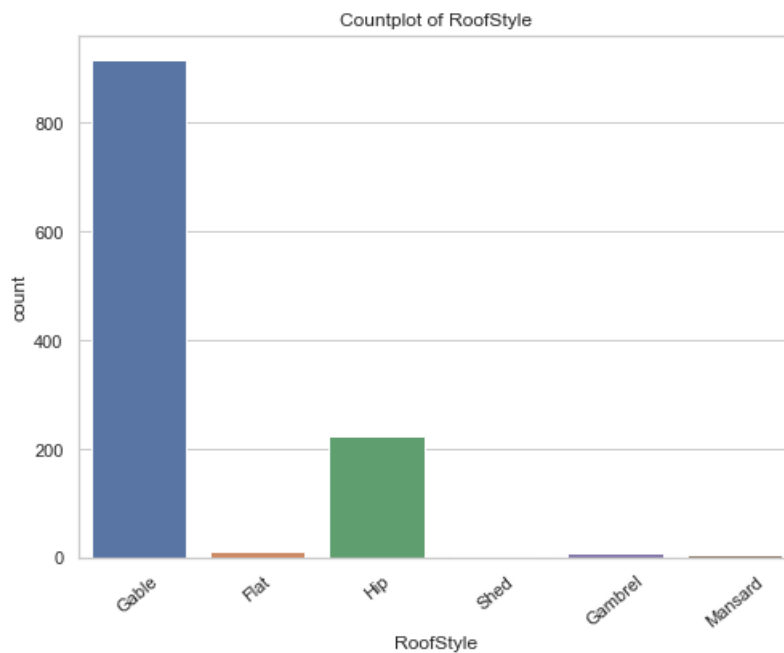
#### Observation:

1 Story has highest number of count followed by 2Story, 1.5Fin, SLvL etc

```
In [33]: # Let's check the column RoofStyle

plt.subplots(figsize=(8,6))
sns.countplot(x="RoofStyle", data=housing_train)
plt.title("Countplot of RoofStyle")
plt.xticks(rotation=40)
plt.xlabel('RoofStyle')
plt.ylabel("count")
plt.show()

housing_train['RoofStyle'].value_counts()
```



```
Out[33]: Gable    915
         Hip      225
         Flat      12
         Gambrel    9
         Mansard    5
         Shed       2
```

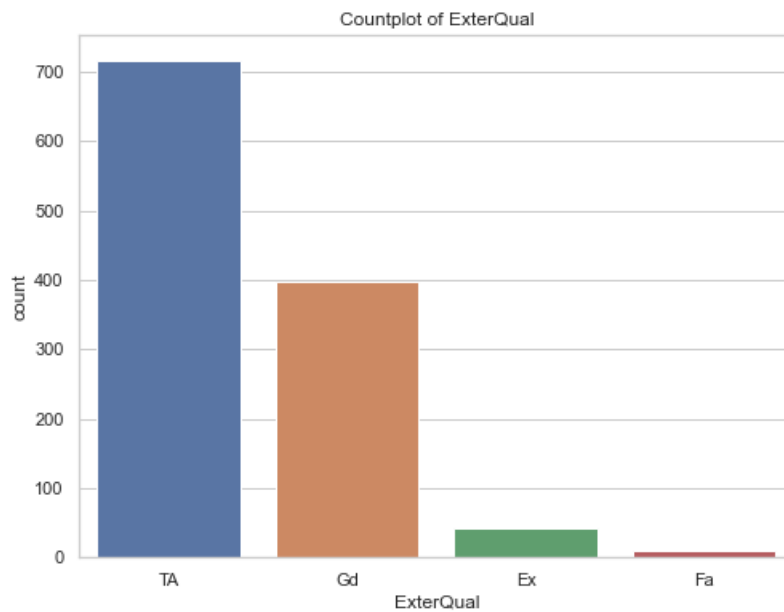
**Observation:**

Maximum, 915 number of RoofStyle are Gable.

```
In [34]: # Let's check the column ExterQual
```

```
plt.subplots(figsize=(8,6))
sns.countplot(x="ExterQual", data=housing_train)
plt.title("Countplot of ExterQual")
plt.xlabel('ExterQual')
plt.ylabel("count")
plt.show()

housing_train['ExterQual'].value_counts()
```



```
Out[34]: TA      717
        Gd      397
        Ex       43
        Fa       11
        Name: ExterQual, dtype: int64
```

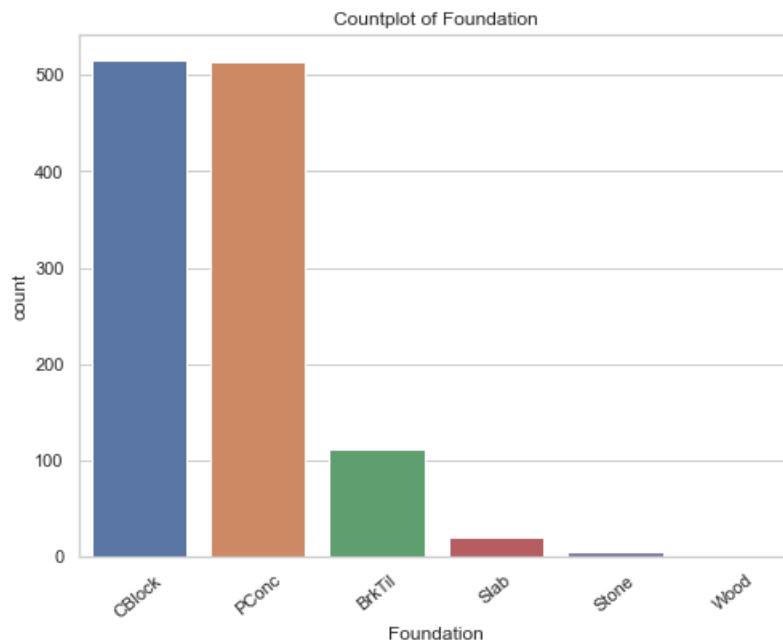
**Observation:** ¶

Maximum, 717 number of ExterQual is TA.



```
In [35]: # Let's checking the column Foundation
```

```
plt.subplots(figsize=(8,6))  
sns.countplot(x="Foundation", data=housing_train)  
plt.title("Countplot of Foundation")  
plt.xticks(rotation=40)  
plt.xlabel('Foundation')  
plt.ylabel("count")  
plt.show()  
  
housing_train['Foundation'].value_counts()
```



```
Out[35]: CBlock    516  
         PConc     513  
         BrkTil    112  
         Slab      21  
         Stone      5
```

---

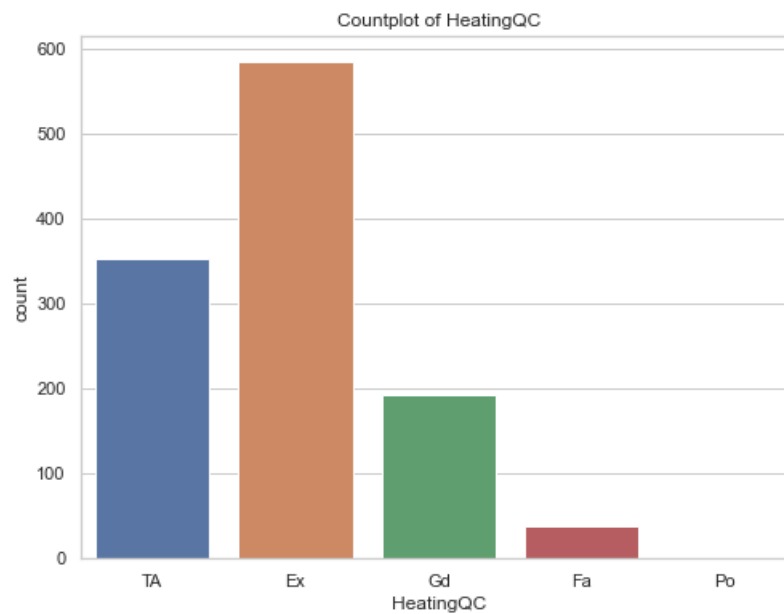
**Observation:**

Maximum, 516 number of Foundation are CBlock.

```
In [36]: # Let's check the column HeatingQC

plt.subplots(figsize=(8,6))
sns.countplot(x="HeatingQC", data=housing_train)
plt.title("Countplot of HeatingQC")
plt.xlabel('HeatingQC')
plt.ylabel("count")
plt.show()

housing_train['HeatingQC'].value_counts()
```



```
Out[36]: Ex      585
TA       352
Gd       192
Fa        38
Po         1
Name: HeatingQC, dtype: int64
```

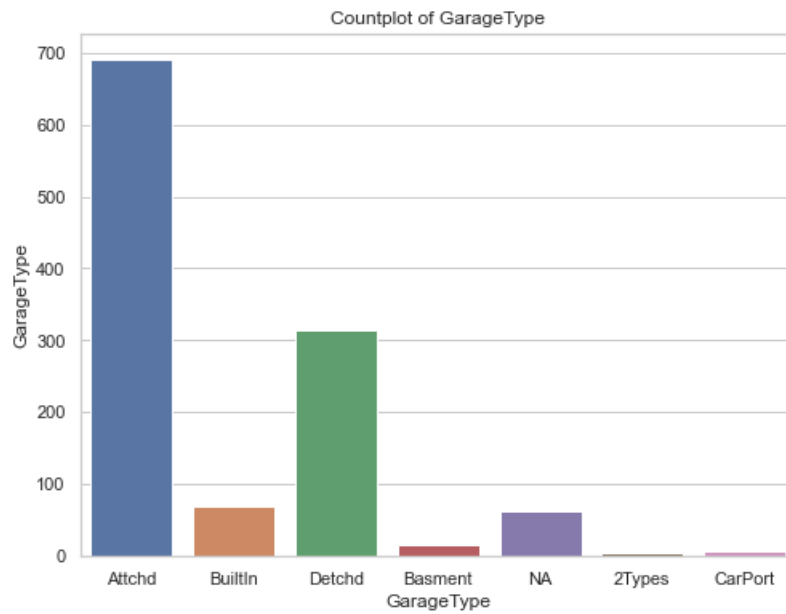
**Observation:**

Maximum, 585 number of HeatingQC is Ex.

```
In [37]: # Let's check the column GarageType
```

```
plt.subplots(figsize=(8,6))
sns.countplot(x="GarageType", data=housing_train)
plt.title("Countplot of GarageType")
plt.xlabel('GarageType')
plt.ylabel("GarageType")
plt.show()

housing_train['GarageType'].value_counts()
```



```
Out[37]: Attchd      691
Detchd      314
BuiltIn      70
NA          64
Basement     16
CarPort       8
```

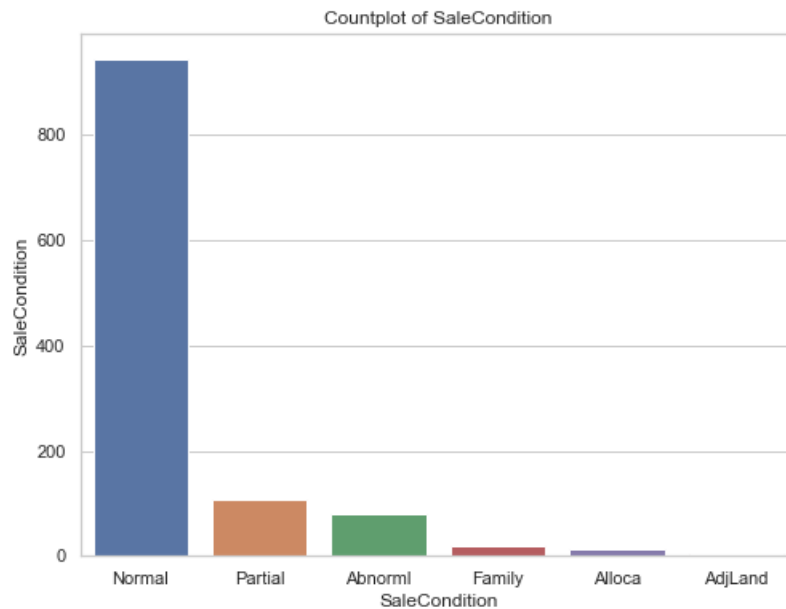
#### Observation:

Maximum, 691 number of GarageType are Attchd.

```
In [38]: # Let's check the column SaleCondition

plt.subplots(figsize=(8,6))
sns.countplot(x="SaleCondition", data=housing_train)
plt.title("Countplot of SaleCondition")
plt.xlabel('SaleCondition')
plt.ylabel("SaleCondition")
plt.show()

housing_train['SaleCondition'].value_counts()
```



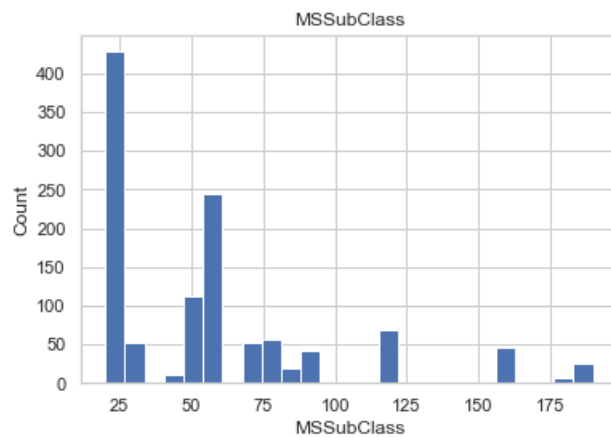
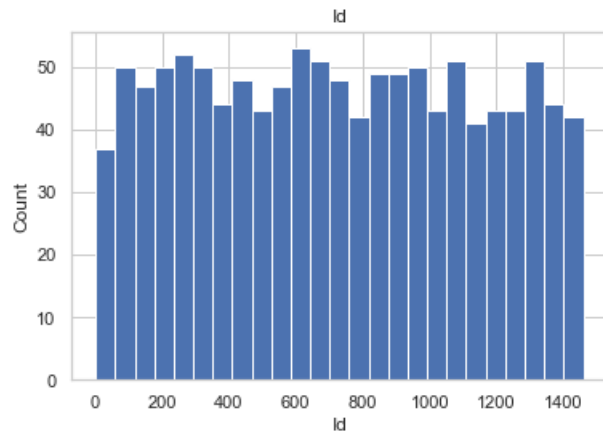
```
Out[38]: Normal      945
Partial      108
Abnorml      81
Family       18
Alloca       12
AdjLand       4
Name: SaleCondition, dtype: int64
```

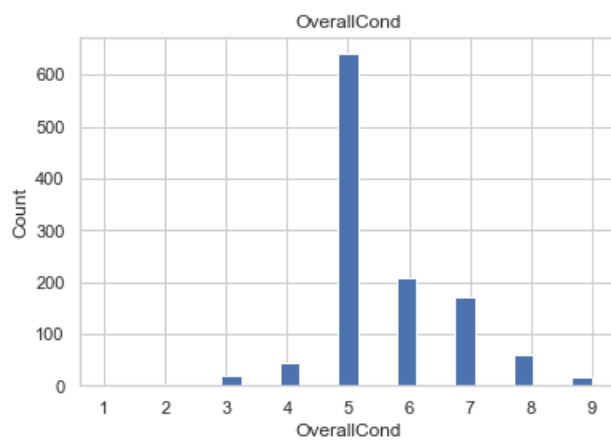
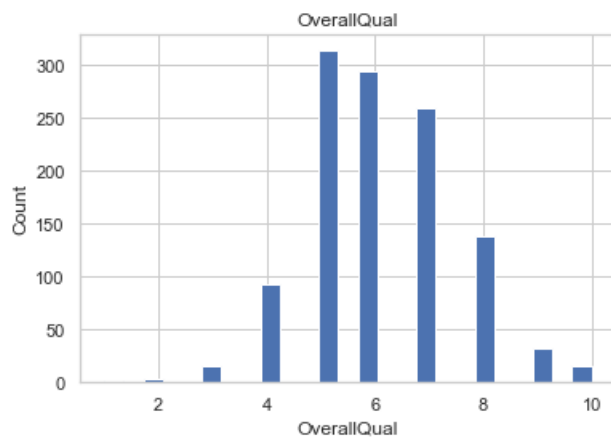
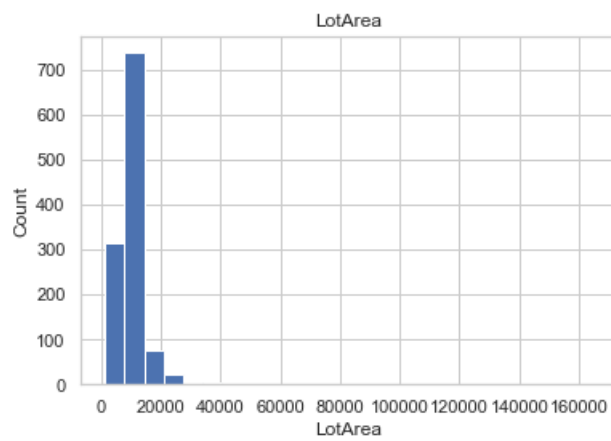
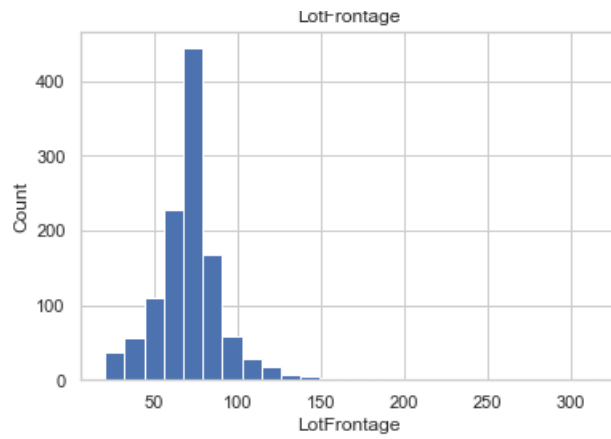
**Observation:**

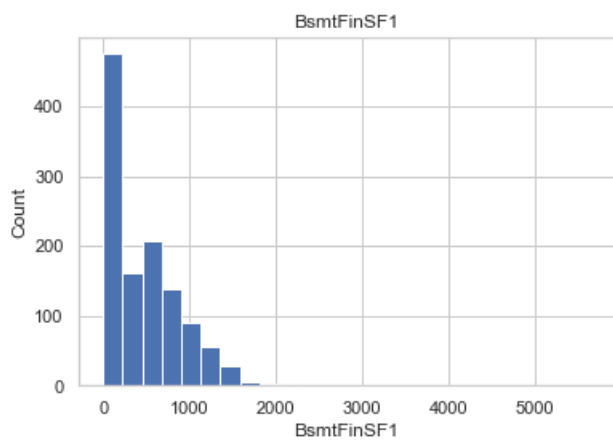
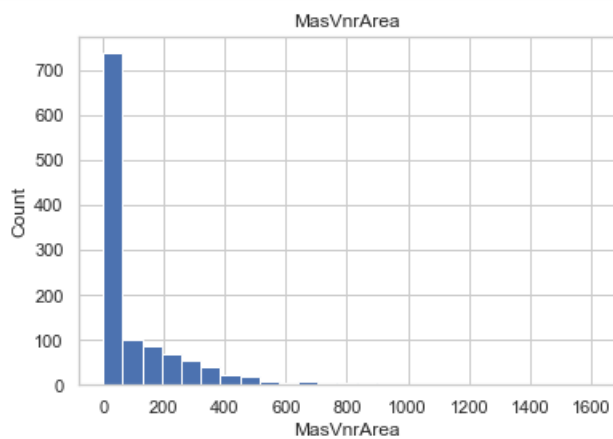
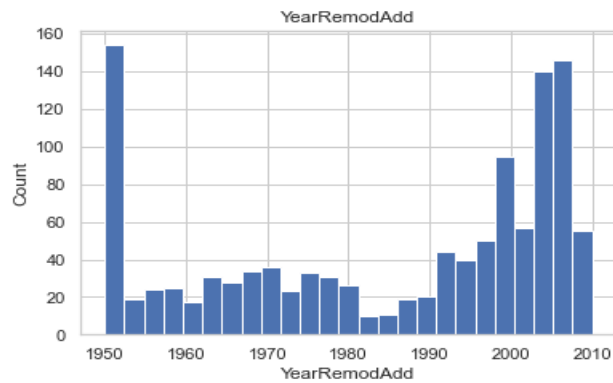
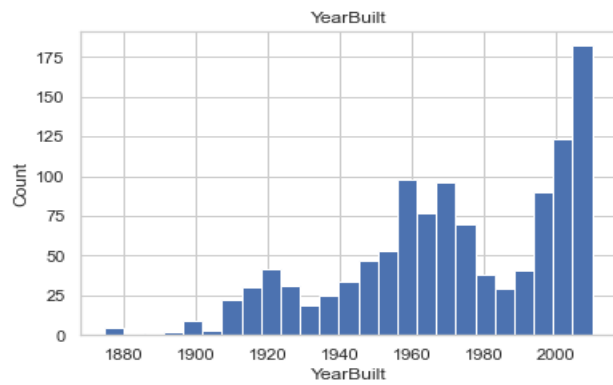
Maximum, 945 number of SaleCondition is normal.

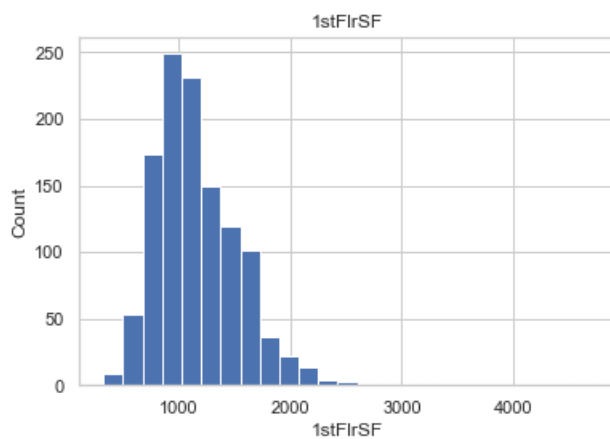
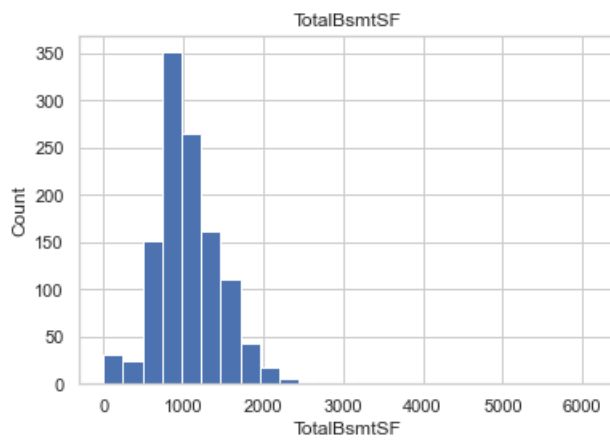
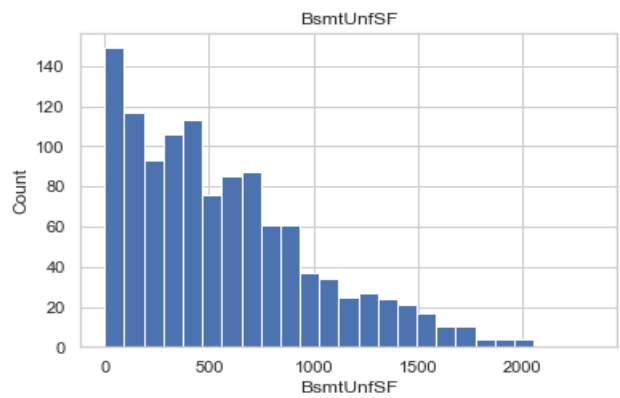
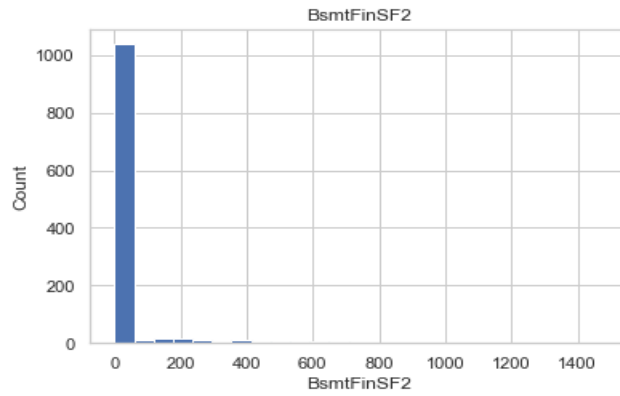
```
In [39]: # Let's plot the histogram of every numerical column
```

```
for col in housing_train.describe().columns:  
    data=housing_train.copy()  
    data[col].hist(bins=25)  
    plt.xlabel(col)  
    plt.ylabel("Count")  
    plt.title(col)  
    plt.show()
```

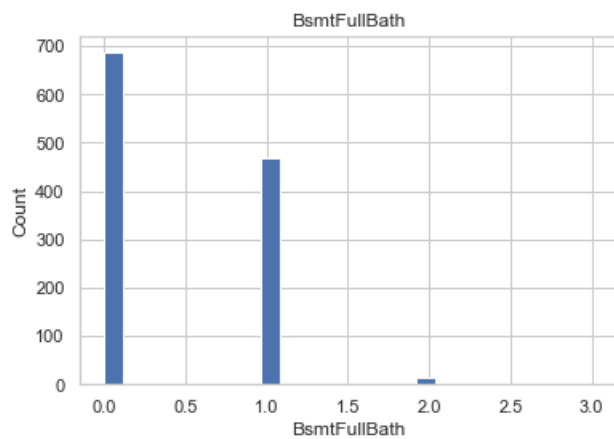
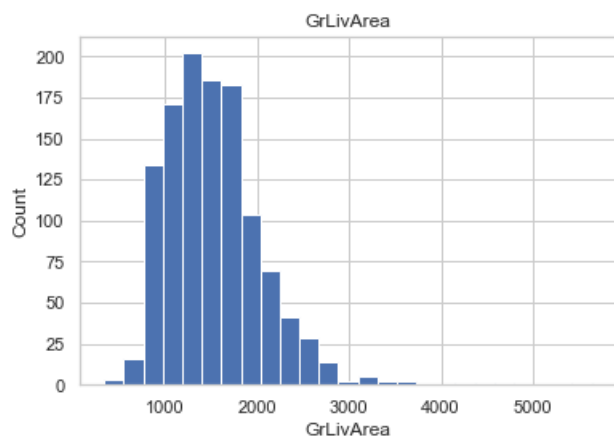
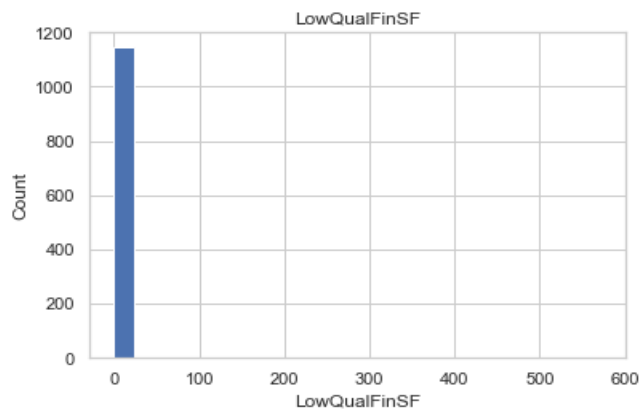
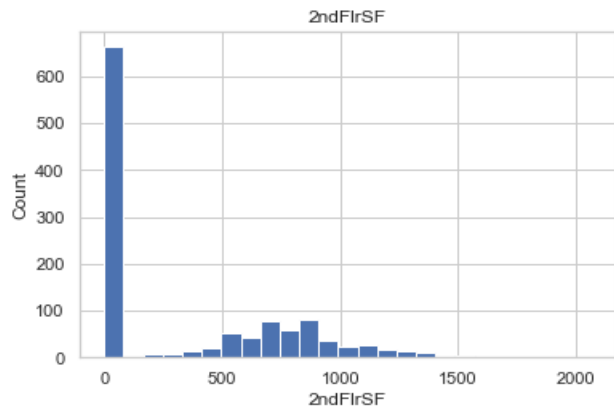


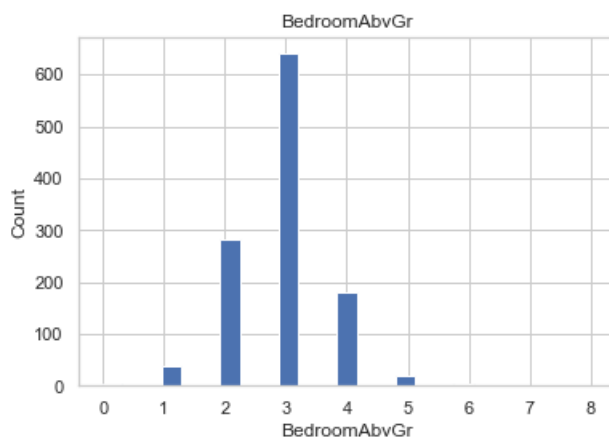
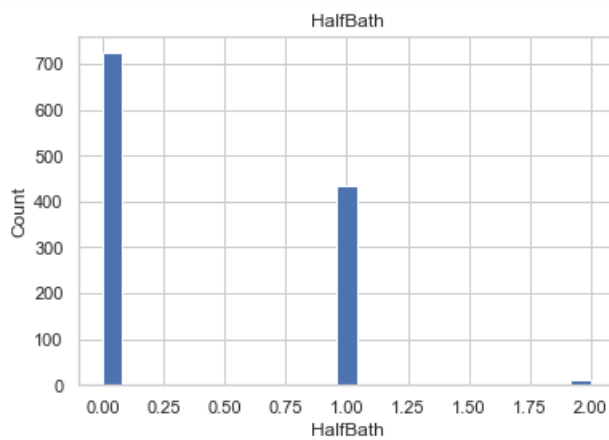
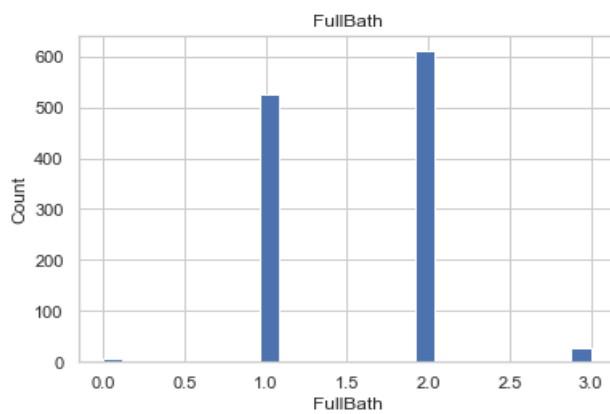
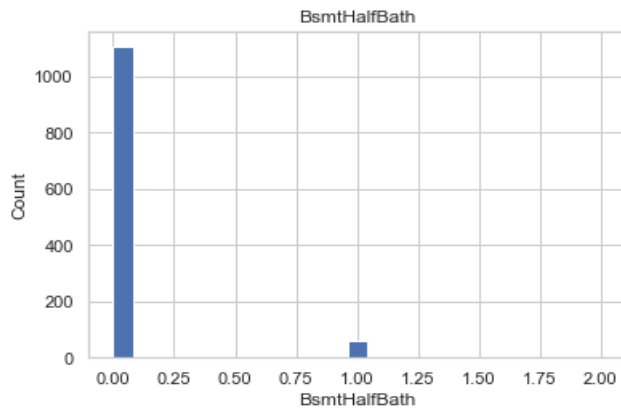


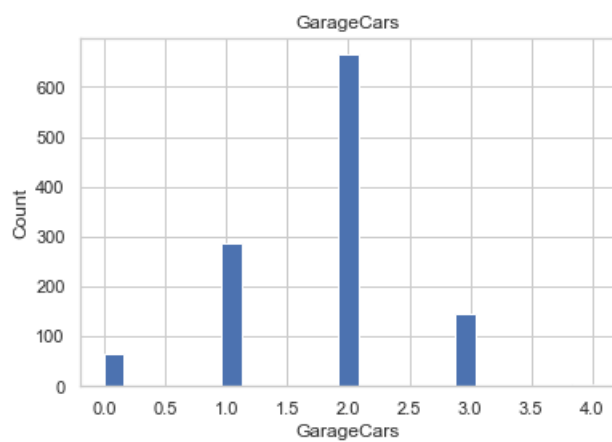
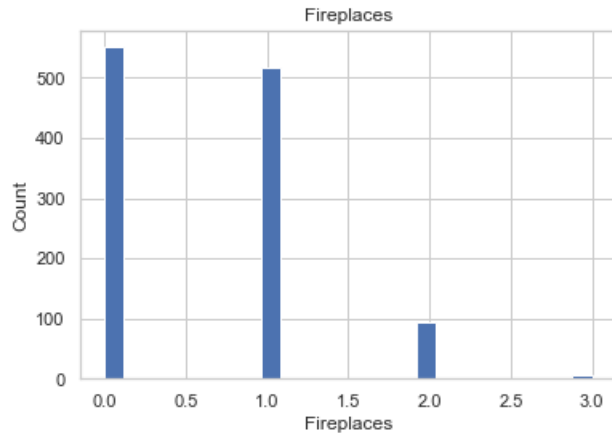
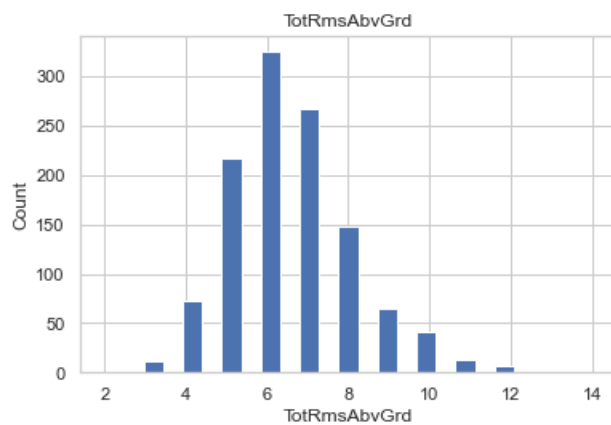
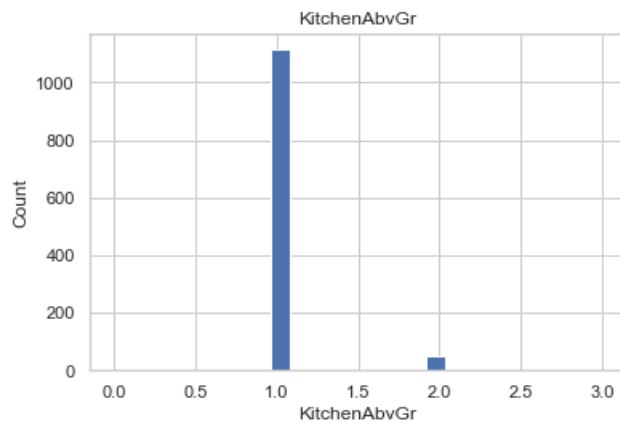


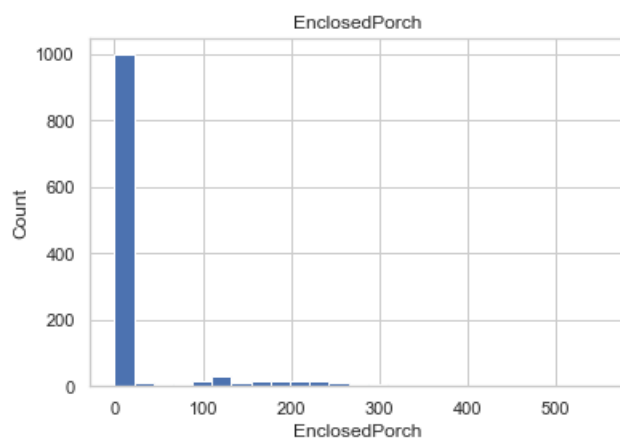
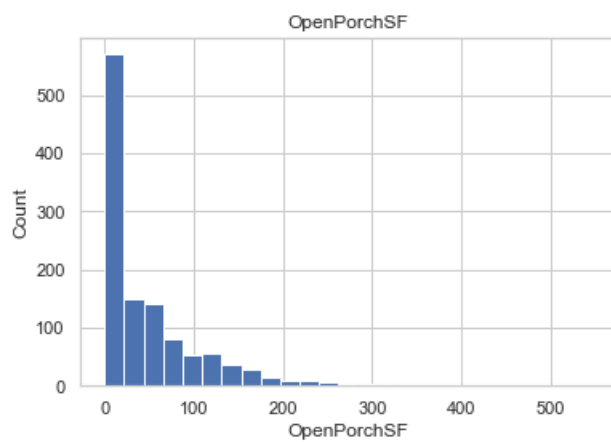
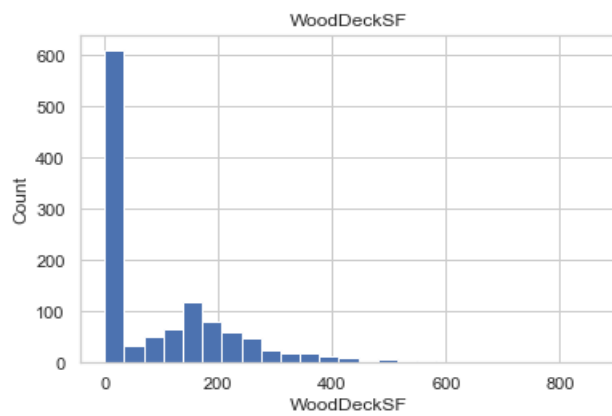
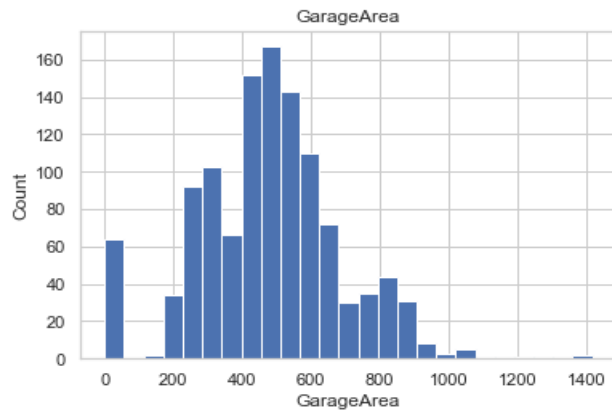


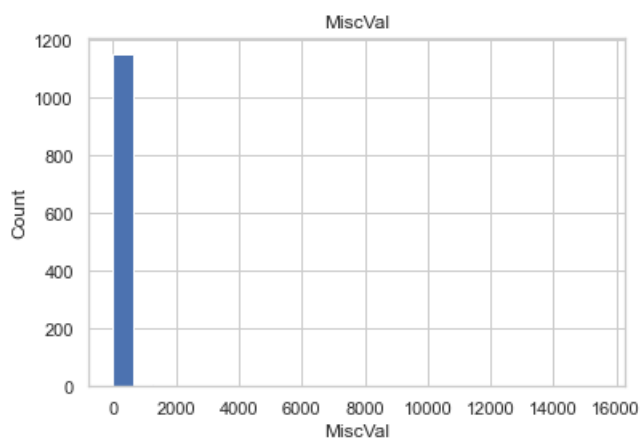
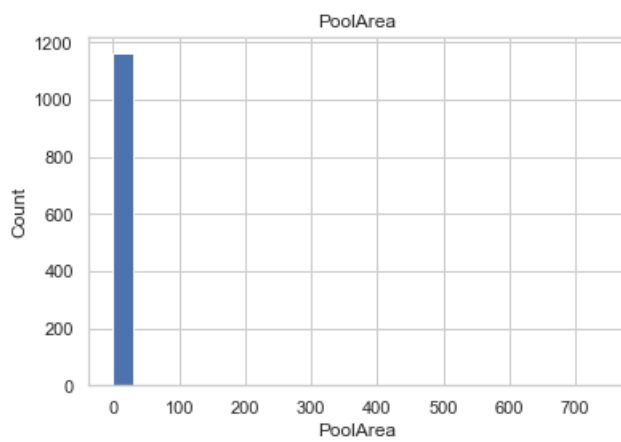
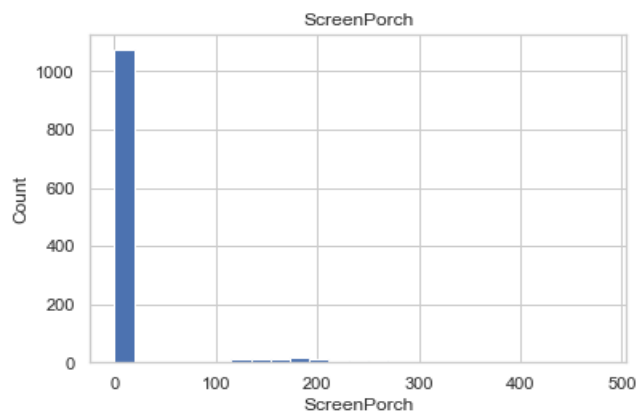
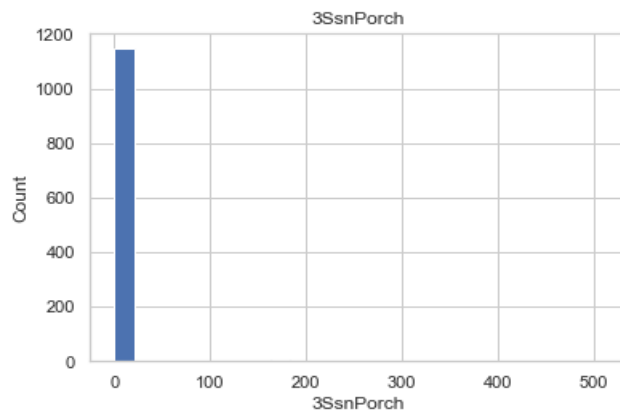


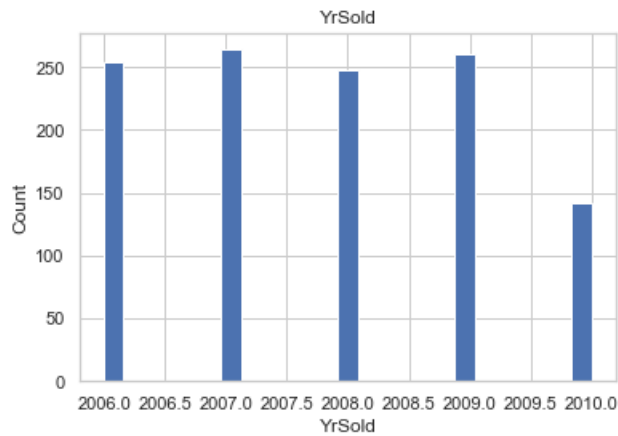
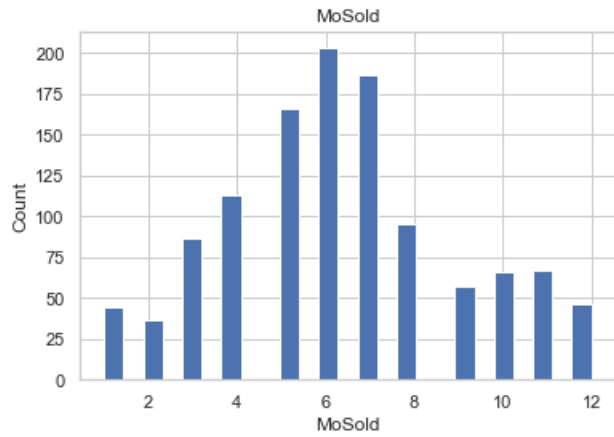








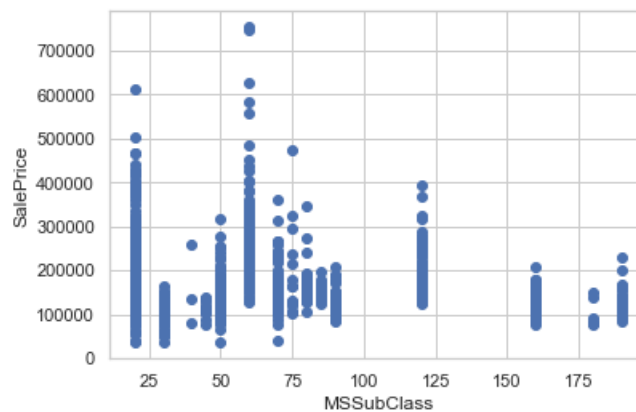
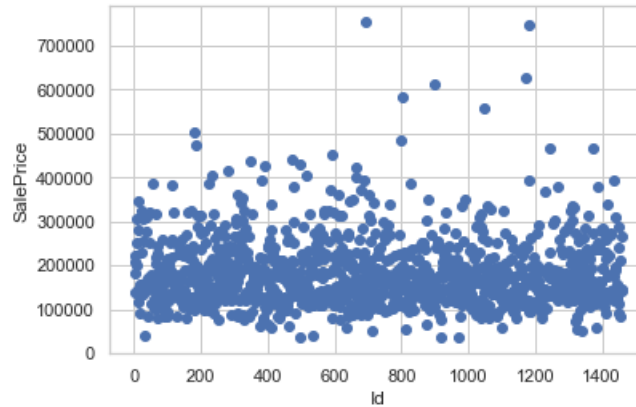


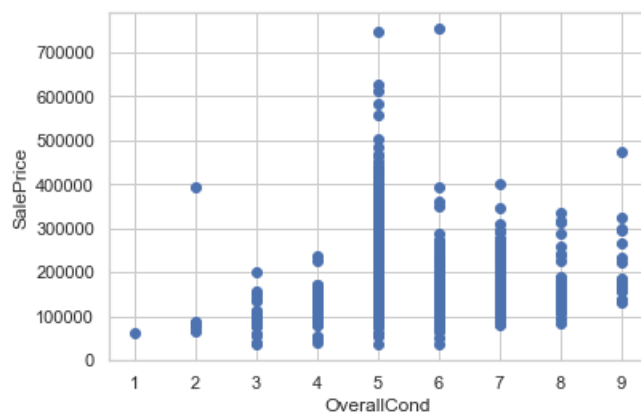
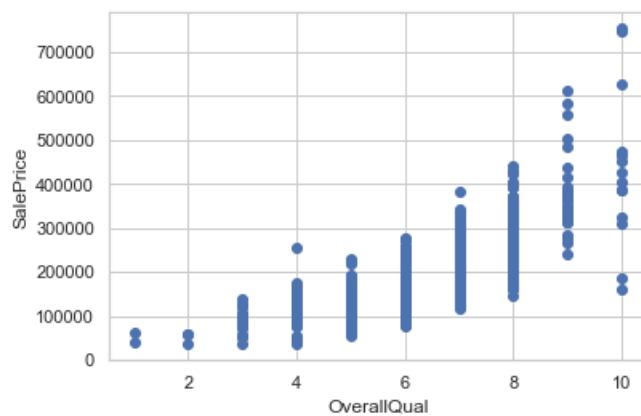
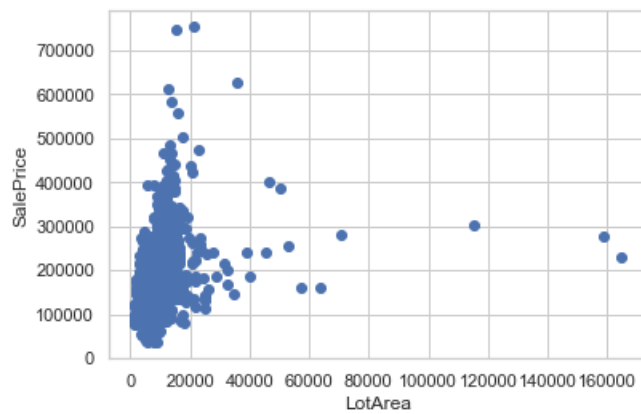
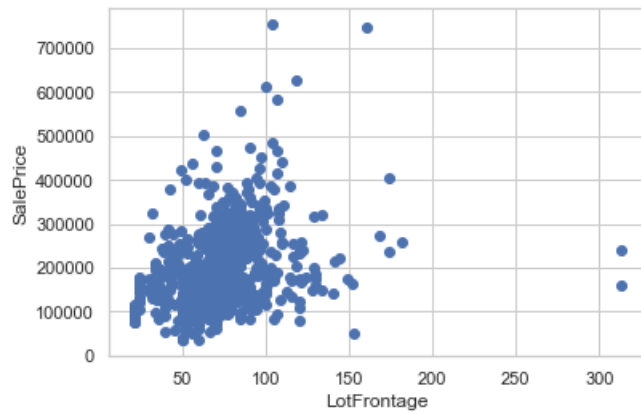


## Bivariate Analysis

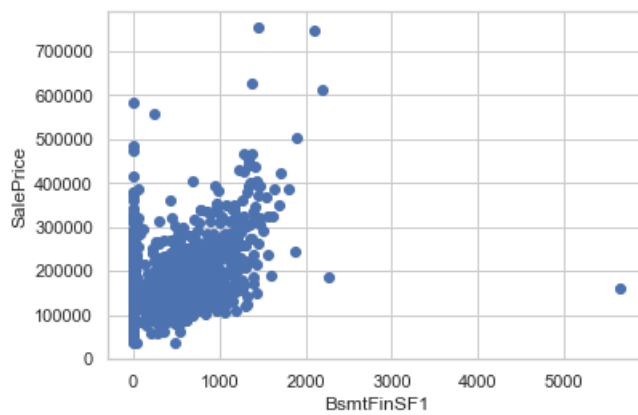
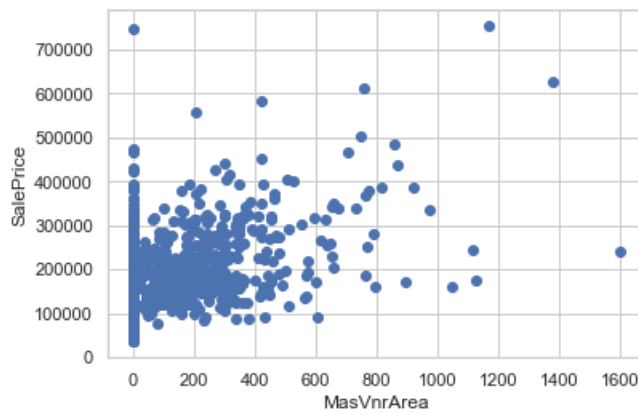
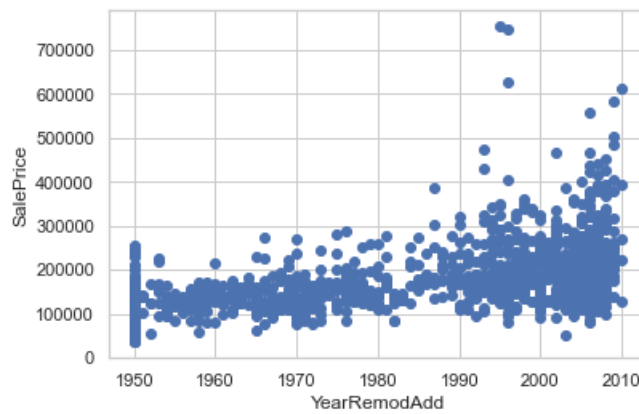
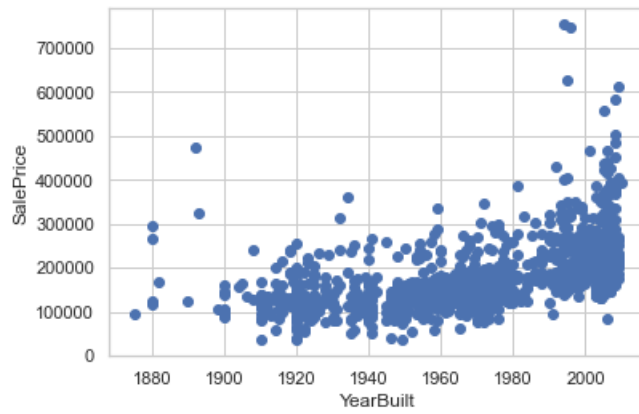
```
In [40]: # Let's plot the Scatter plot between all feature variables and target variable

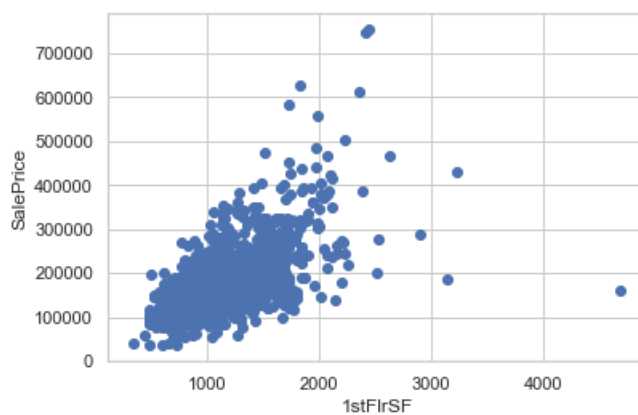
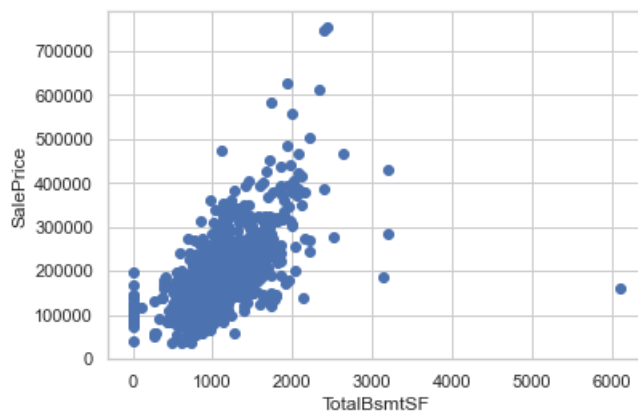
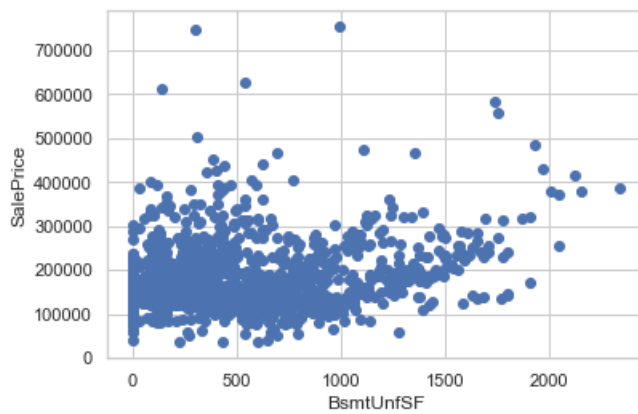
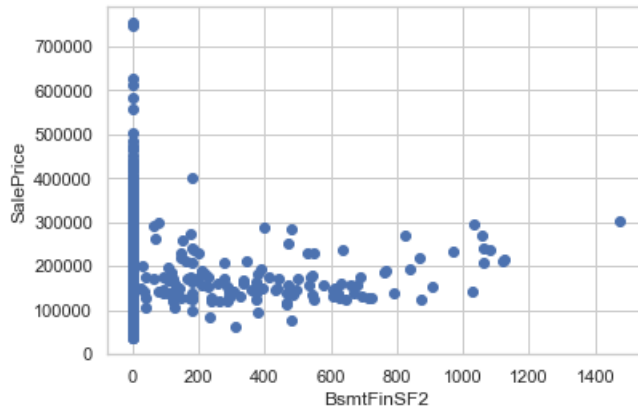
for col in housing_train.describe().columns:
    data=housing_train.copy()
    plt.scatter(data[col],data['SalePrice'])
    plt.xlabel(col)
    plt.ylabel('SalePrice')
    plt.show()
```

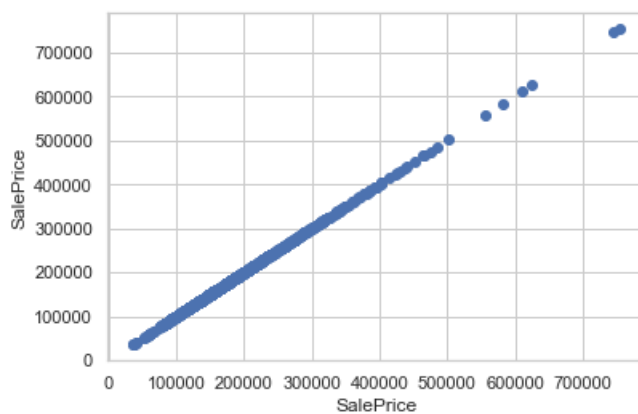
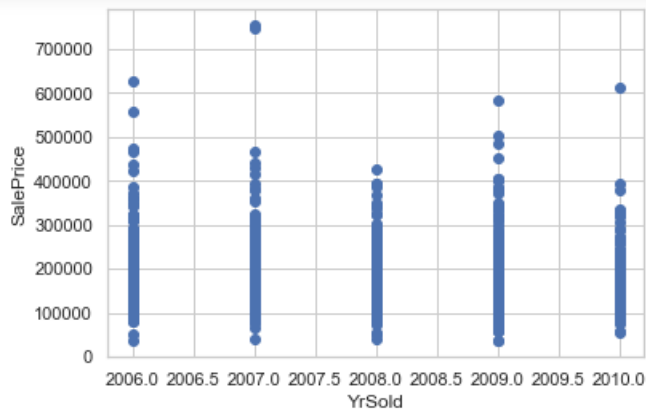
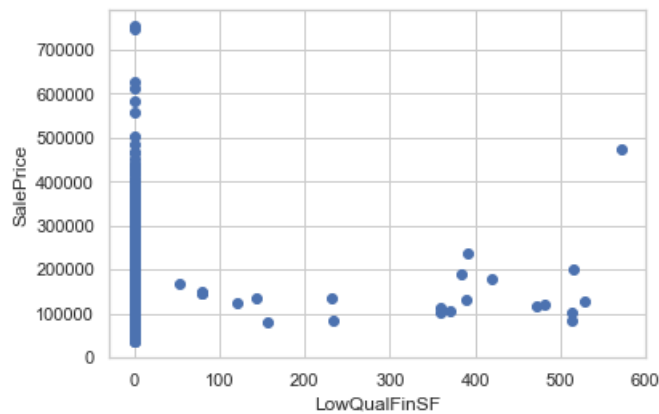
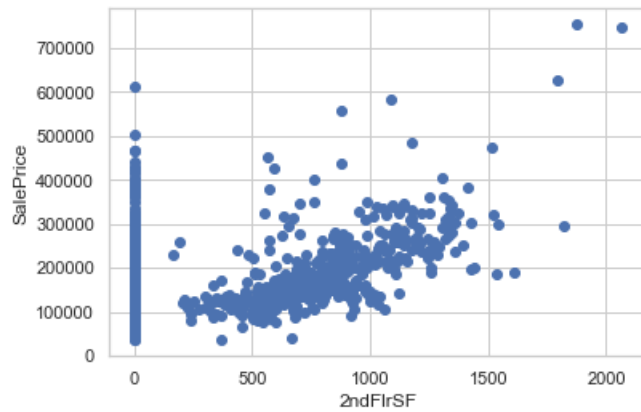








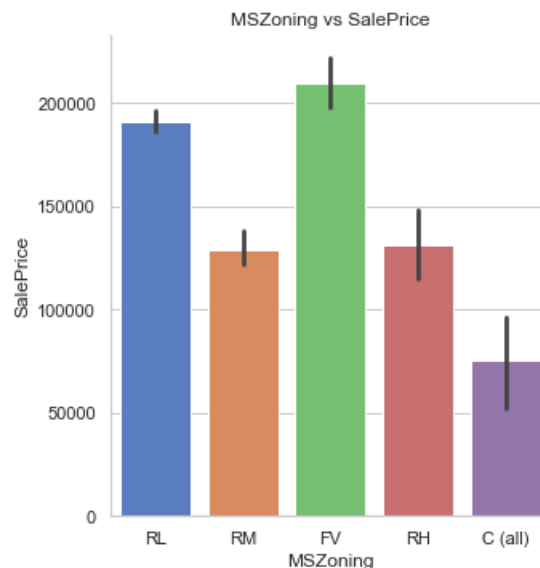




```
In [41]: # Let's plot the Factor plot of MSZoning vs SalePrice

plt.figure(figsize=(8,6))
sns.factorplot(x='MSZoning',y='SalePrice',data=housing_train,kind='bar',size=5,
plt.title('MSZoning vs SalePrice')
plt.ylabel('SalePrice')
plt.show()
print(housing_train.groupby('SalePrice')['MSZoning'].value_counts());
```

<Figure size 576x432 with 0 Axes>



SalePrice	MSZoning	
34900	C (all)	1
35311	C (all)	1
37900	RM	1
39300	RL	1
40000	C (all)	1
	..	
582933	RL	1
611657	RL	1
625000	RL	1
745000	RL	1

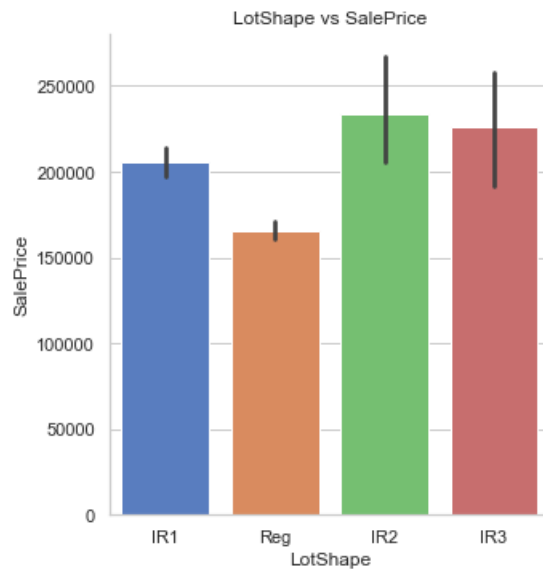
#### Observation:

SalePrice is maximum with FV MSZoning.

In [42]: # Let's plot the Factor plot of LotShape vs SalePrice

```
plt.figure(figsize=(8,6))
sns.factorplot(x='LotShape',y='SalePrice',data=housing_train,kind='bar',size=5,p
plt.title('LotShape vs SalePrice')
plt.ylabel('SalePrice')
plt.show();
print(housing_train.groupby('SalePrice')['LotShape'].value_counts());
```

<Figure size 576x432 with 0 Axes>



SalePrice	LotShape
34900	Reg
35311	Reg
37900	Reg
39300	Reg
40000	Reg
..	..
582933	Reg
611657	IR1
625000	IR1

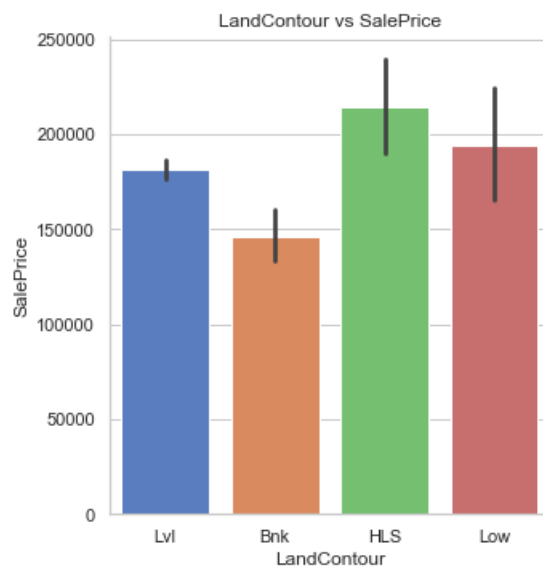
#### Observation:

SalePrice is maximum with IR2 LotShape.

```
In [43]: # Let's plot the Factor plot of LandContour vs SalePrice

plt.figure(figsize=(8,6))
sns.factorplot(x='LandContour',y='SalePrice',data=housing_train,kind='bar',size=
plt.title('LandContour vs SalePrice')
plt.ylabel('SalePrice')
plt.show()
print(housing_train.groupby('SalePrice')['LandContour'].value_counts())
```

<Figure size 576x432 with 0 Axes>



```
SalePrice  LandContour
34900      Lvl          1
35311      Lvl          1
37900      Lvl          1
39300      Low          1
40000      Lvl          1
..
582933     Lvl          1
611657     Lvl          1
625000     Lvl          1
745000     Lvl          1
-----
```

#### Observation:

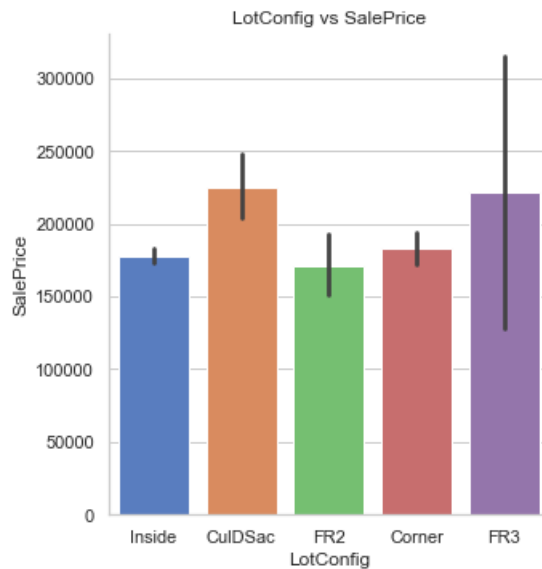
SalePrice is maximum with HLS LandContour.

```
In [44]: # Let's plot the Factor plot of LotConfig vs SalePrice

plt.figure(figsize=(8,6))
sns.factorplot(x='LotConfig',y='SalePrice',data=housing_train,kind='bar',size=5,
plt.title('LotConfig vs SalePrice')
plt.ylabel('SalePrice')
plt.show()

print(housing_train[['LotConfig']].value_counts())
```

<Figure size 576x432 with 0 Axes>



SalePrice	LotConfig	
34900	Inside	1
35311	Inside	1
37900	Inside	1
39300	Inside	1
40000	Inside	1
..		
582933	Inside	1
611657	Inside	1
625000	CulDSac	1

#### Observation:

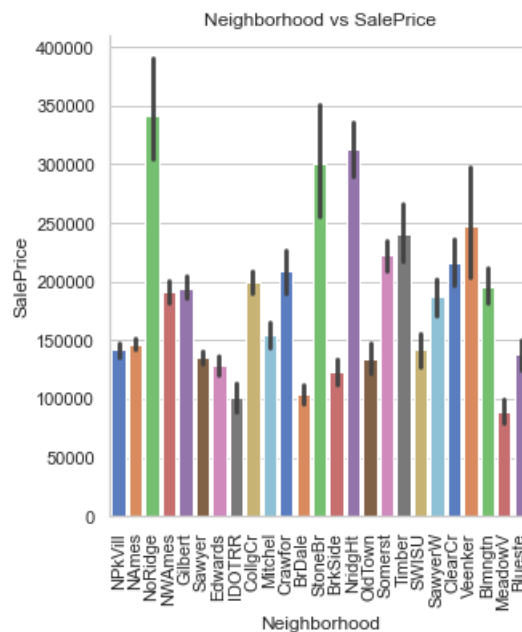
SalePrice is maximum with CulDSac LotConfig.

In [45]: # Let's plo the Factor plot of Neighborhood vs SalePrice

```
plt.figure(figsize=(16,16))
sns.factorplot(x='Neighborhood',y='SalePrice',data=housing_train,kind='bar',size
plt.title('Neighborhood vs SalePrice')
plt.xticks(rotation='vertical')
plt.ylabel('SalePrice')
plt.show()

print(housing_train.groupby('SalePrice')['Neighborhood'].value_counts())
```

<Figure size 1152x1152 with 0 Axes>



SalePrice	Neighborhood	
34900	IDOTRR	1
35311	IDOTRR	1
37900	OldTown	1
39300	BrkSide	1

#### Observation:

SalePrice is maximum with NoRidge Neighborhood.

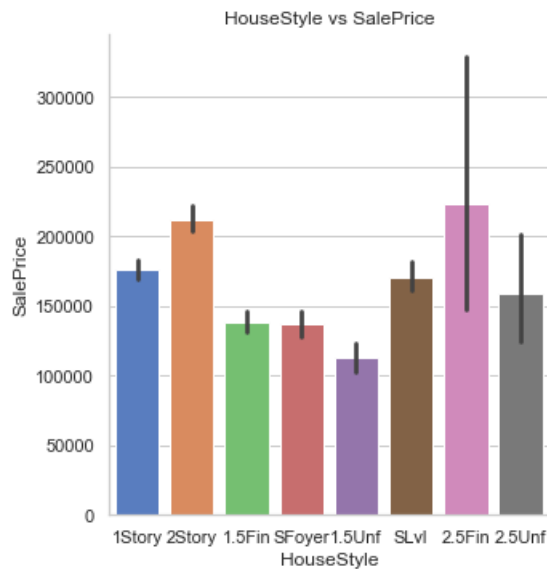


In [46]: # Let's plot the Factor plot of HouseStyle vs SalePrice

```
plt.figure(figsize=(8,6))
sns.factorplot(x='HouseStyle',y='SalePrice',data=housing_train,kind='bar',size=5)
plt.title('HouseStyle vs SalePrice')
plt.ylabel('SalePrice')
plt.show()

print(housing_train.groupby('SalePrice')['HouseStyle'].value_counts())
```

<Figure size 576x432 with 0 Axes>



SalePrice	HouseStyle	
34900	1Story	1
35311	1Story	1
37900	1.5Fin	1
39300	1Story	1
40000	2Story	1
		..
582933	2Story	1
.....	...	.

#### Observation:

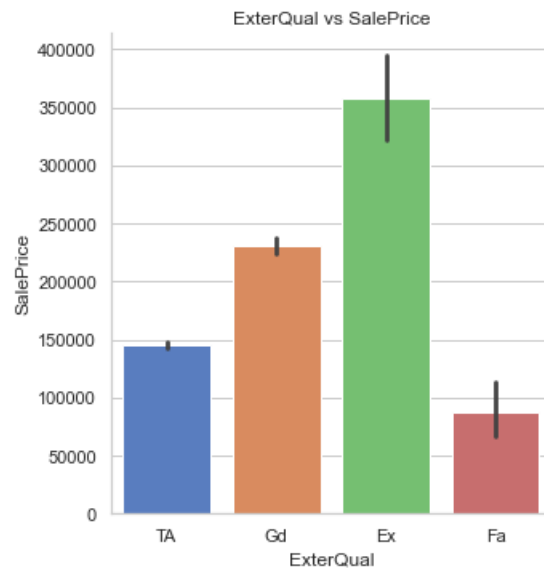
SalePrice is maximum with 2.5Fin HouseStyle.

```
In [47]: # Let's plot the Factor plot of ExterQual vs SalePrice

plt.figure(figsize=(8,6))
sns.factorplot(x='ExterQual',y='SalePrice',data=housing_train,kind='bar',size=5,
plt.title('ExterQual vs SalePrice')
plt.ylabel('SalePrice')
plt.show()

print(housing_train.groupby('SalePrice')['ExterQual'].value_counts())
```

<Figure size 576x432 with 0 Axes>



SalePrice	ExterQual	
34900	TA	1
35311	TA	1
37900	TA	1
39300	Fa	1
40000	TA	1

#### Observation:

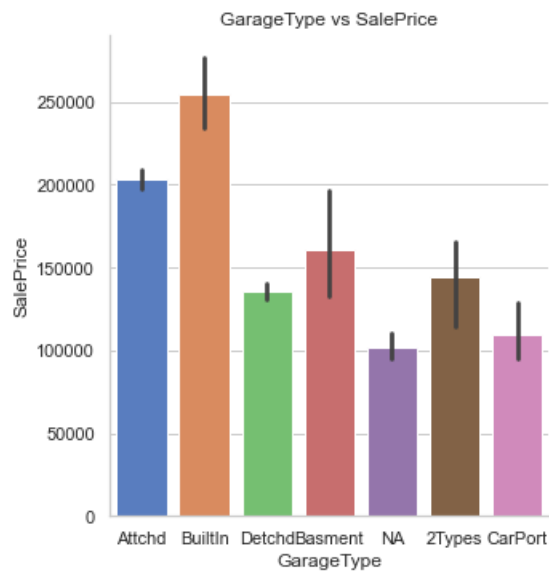
SalePrice is maximum with Ex ExterQual.

```
In [48]: # Let's plot the Factor plot of GarageType vs SalePrice

plt.figure(figsize=(8,6))
sns.factorplot(x='GarageType',y='SalePrice',data=housing_train,kind='bar',size=5)
plt.title('GarageType vs SalePrice')
plt.ylabel('SalePrice')
plt.show()

print(housing_train.groupby('SalePrice')['GarageType'].value_counts())
```

<Figure size 576x432 with 0 Axes>



SalePrice	GarageType	
34900	NA	1
35311	Detchd	1
37900	NA	1
39300	NA	1
40000	Detchd	1

#### Observation:

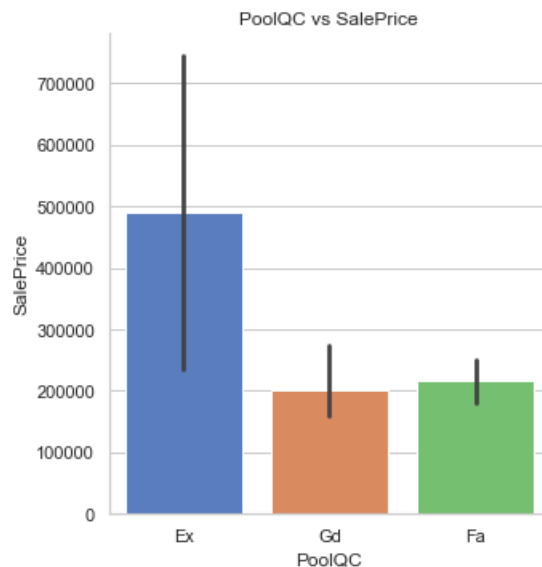
SalePrice is maximum with Builtin GarageType.

```
In [49]: # Let's plot the Factor plot of PoolQC vs SalePrice

plt.figure(figsize=(8,6))
sns.factorplot(x='PoolQC',y='SalePrice',data=housing_train,kind='bar',size=5,pal
plt.title('PoolQC vs SalePrice')
plt.ylabel('SalePrice')
plt.show()

print(housing_train.groupby('SalePrice')['PoolQC'].value_counts())
```

<Figure size 576x432 with 0 Axes>



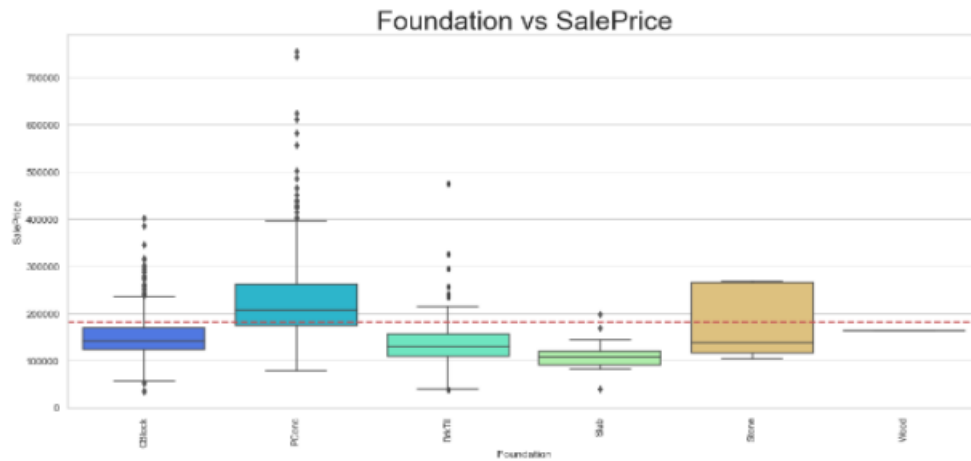
```
SalePrice PoolQC
160000    Gd      1
171000    Gd      1
181000    Fa      1
235000    Ex      1
250000    Fa      1
274970    Gd      1
745000    Ex      1
..      ..      ..
```

#### Observation:

SalePrice is maximum with Ex PoolQC.

In [50]: *# Let's plot the Foundation vs SalePrice plot*

```
plt.figure(figsize=(18,8))
mean_price=np.mean(housing_train['SalePrice'])
sns.boxplot(y='SalePrice',x='Foundation',data=housing_train,palette="rainbow")
plt.axhline(mean_price,color='r',linestyle='dashed',linewidth=2)
plt.title("Foundation vs SalePrice",fontsize=30)
plt.xticks(rotation='vertical')
plt.show()
```

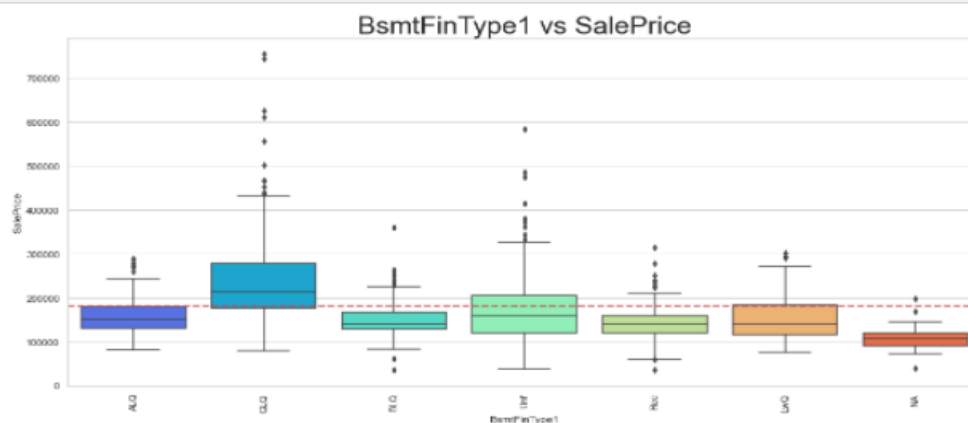


**Observation:**

SalePrice is maximum with PConc.

In [51]: *# Let's plot the BsmtFinType1 vs SalePrice plot*

```
plt.figure(figsize=(18,8))
mean_price=np.mean(housing_train['SalePrice'])
sns.boxplot(y='SalePrice',x='BsmtFinType1',data=housing_train,palette="rainbow")
plt.axhline(mean_price,color='r',linestyle='dashed',linewidth=2)
plt.title("BsmtFinType1 vs SalePrice",fontsize=30)
plt.xticks(rotation='vertical')
plt.show()
```



**Observation:**

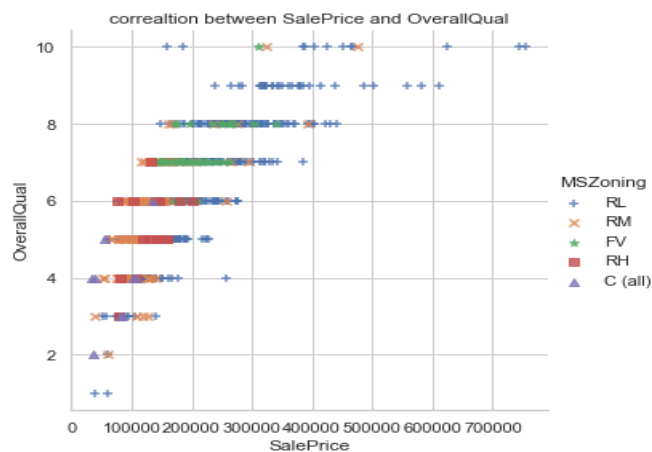
SalePrice is maximum with GLQ BsmtFinType1.

## Multivariate Analysis

In [52]: # Let's plot the scatter plot between SalePrice and OverallQual with respect to MSZoning

```
plt.figure(figsize=(14,14))
sns.lmplot(x='SalePrice',y='OverallQual',fit_reg=False,data=housing_train,hue='MSZoning')
plt.xlabel('SalePrice')
plt.title('correaltion between SalePrice and OverallQual')
plt.ylabel('OverallQual')
plt.show()
```

<Figure size 1008x1008 with 0 Axes>

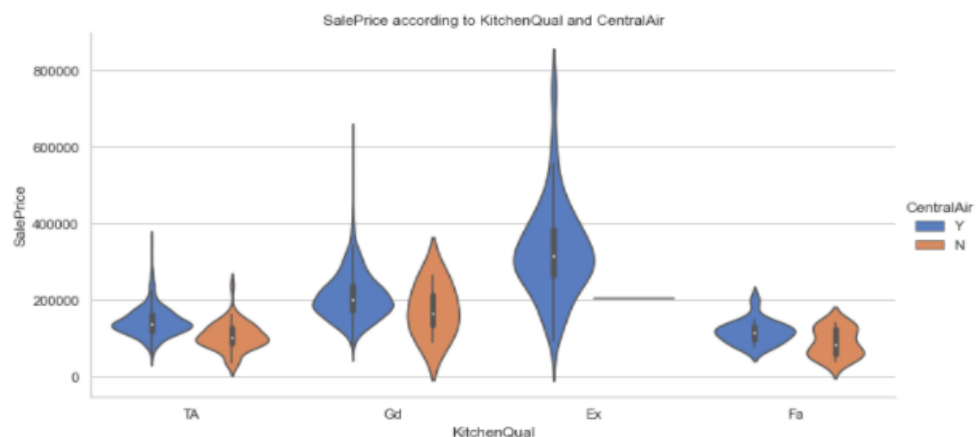


### Observation:

With MSZoning RL and increase in OverallQual the SalePrice of a house increases.

In [53]: # Let's plot the GarageType and GarageCond with respect to SalePrice plot

```
sns.factorplot(x='KitchenQual',y='SalePrice',hue='CentralAir',data=housing_train)
plt.title('SalePrice according to KitchenQual and CentralAir')
plt.xticks()
plt.ylabel('SalePrice')
plt.show()
```



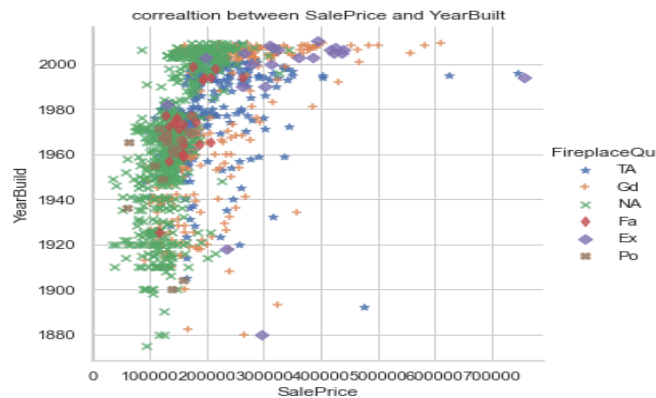
### Observation:

SalePrice is maximum with Ex kitchenQual and CentralAir.

In [54]: # Let's plot the scatter plot between SalePrice and OverallQual with respect to

```
plt.figure(figsize=(14,14))
sns.lmplot(x='SalePrice',y='YearBuilt',fit_reg=False,data=housing_train,hue='FireplaceQu')
plt.xlabel('SalePrice')
plt.title('correlation between SalePrice and YearBuilt')
plt.ylabel('YearBuilt')
plt.show()
```

<Figure size 1008x1008 with 0 Axes>

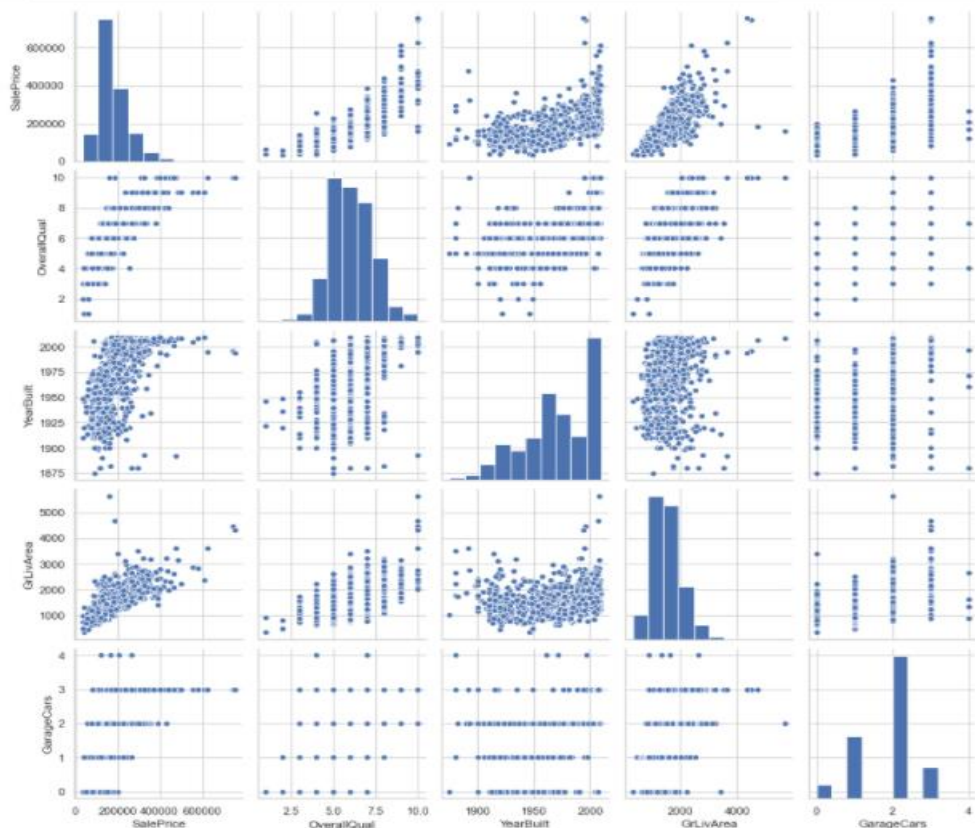


#### Observation:

As the YearBuilt is increasing SalePrice is also increasing.

In [55]: # Let's plot the pairplot

```
sns.pairplot(housing_train, vars=['SalePrice','OverallQual','YearBuilt','GrLivArea'])
```



#### Observation:

SalePrice is highly positively correlated with GrLivArea and OverallQual.

## ***INTERPRETATION OF THE RESULTS***

From the visualization we interpreted that the target variable SalePrice was highly positively correlated with the columns GrLivArea, YearBuilt, OverallQual, GarageCars, GarageArea.

From the preprocessing we interpreted that data was improper scaled.

### **Hyperparameter tuning**

In [74]: *# Let's Use the GridSearchCV to find the best paarameters in Ridge Regressor*

```
parameters={'alpha': [25,10,4,2,1.0,0.8,0.5,0.3,0.2,0.1,0.05,0.02,0.01]}
rg=Ridge()

reg=GridSearchCV(rg,parameters,n_jobs=-1)
reg.fit(x,y)
print(reg.best_params_)

{'alpha': 25}
```

In [75]: *# Let's use the Ridge Regressor with its best parameters*

```
RG=Ridge(alpha=25)
RG.fit(x_train,y_train)
print('Score:',RG.score(x_train,y_train))
y_pred=RG.predict(x_test)
print('\n')
print('Mean absolute error:',mean_absolute_error(y_test,y_pred))
print('Mean squared error:',mean_squared_error(y_test,y_pred))
print('Root Mean Squared error:',np.sqrt(mean_squared_error(y_test,y_pred)))
print('\n')
print("r2_score:",r2_score(y_test,y_pred))
print('\n')
```

Score: 0.8228133117754095

Mean absolute error: 21636.271697150503  
Mean squared error: 1043419637.663922  
Root Mean Squared error: 32302.006712647468

r2\_score: 0.8409922339716864

From the modeling we interpreted that after hyperparameter tuning Ridge Regressor works best with respect to our model with minimum RMSE of 32302



## ***CONCLUSION***

### ***KEY FINDINGS AND CONCLUSIONS OF THE STUDY***

In this project we have tried to show how the house prices vary and what are the factors related to the changing of house prices. The best(minimum) RMSE score was achieved using the best parameters of Ridge Regressor through GridSearchCV though Lasso Regressor model performed well too.

### ***LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE***

This project has demonstrated the importance of sampling effectively, modelling and predicting data.

Through different powerful tools of visualization we were able to analyse and interpret different hidden insights about the data.

Through data cleaning we were able to remove unnecessary columns and outliers from our dataset due to which our model would have suffered from overfitting or underfitting.

The few challenges while working on this project where:-

- Improper scaling
- Too many features
- Missing values
- Skewed data due to outliers

The data was improper scaled so we scaled it to a single scale using sklearn's package StandardScaler.

There were too many(256) features present in the data so we applied Principal Component Analysis(PCA) and found out the Eigenvalues and on the basis of number of nodes we were able able to reduce our features upto 90 columns.

There were lot of missing values present in different columns which we imputed on the basis of our understanding.

The columns were skewed due to presence of outliers which we handled through winsorization technique.

### ***LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK***

While we couldn't reach our goal of minimum RMSE in house price prediction without letting the model to overfit, we did end up creating a system that can with enough time and data get very close to that goal. As with any project there is room for improvement here. The very nature of this project allows for multiple algorithms to be integrated together as modules and their results can be combined to increase the accuracy of the final result. This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others. Once that condition is satisfied, the modules are easy to add as done in the code. This provides a great degree of modularity and versatility to the project.