In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
pd.pandas.set_option('display.max_columns',None)
```

In [3]:
```
df=pd.read_csv('train1.csv')
```

In [4]:
```
df.head()
```

Out[4]:

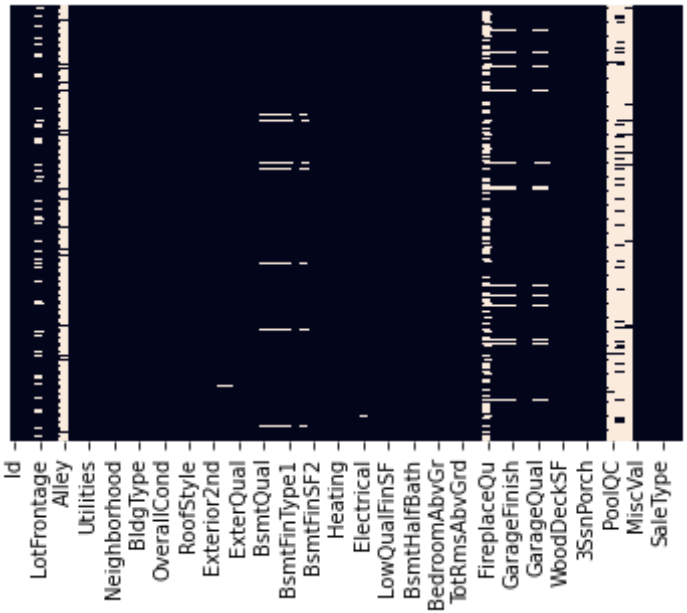
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	All
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	All
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	All
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	All
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	All

In [5]:
```
df['MSZoning'].value_counts()
```

Out[5]:
```
RL         1151
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64
```

In [6]:
```
#Heatr map for null values
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

Out[6]: <AxesSubplot:>



In [7]:
```
df.shape
```

Out[7]: (1460, 81)

In [5]:
```
#Checking percentage of nan values present
#Make the list of features with missing values
features_with_na= [feat for feat in df.columns if df[feat].isnull().sum
()>=1]

#Print feature name and percentage of missung values
for feature in features_with_na:
    print(feature, np.round(df[feature].isnull().mean(), 4),  ' % missi
ng values')
```

```
LotFrontage 0.1774  % missing values
Alley 0.9377  % missing values
MasVnrType 0.0055  % missing values
MasVnrArea 0.0055  % missing values
```

```
BsmtQual 0.0253  % missing values
BsmtCond 0.0253  % missing values
BsmtExposure 0.026  % missing values
BsmtFinType1 0.0253  % missing values
BsmtFinType2 0.026  % missing values
Electrical 0.0007  % missing values
FireplaceQu 0.4726  % missing values
GarageType 0.0555  % missing values
GarageYrBlt 0.0555  % missing values
GarageFinish 0.0555  % missing values
GarageQual 0.0555  % missing values
GarageCond 0.0555  % missing values
PoolQC 0.9952  % missing values
Fence 0.8075  % missing values
MiscFeature 0.963  % missing values
```

In [6]:
```
features_with_na
```

Out[6]:
```
['LotFrontage',
 'Alley',
 'MasVnrType',
 'MasVnrArea',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Electrical',
 'FireplaceQu',
 'GarageType',
 'GarageYrBlt',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PoolQC',
 'Fence',
 'MiscFeature']
```

In [562]:
```
df.isnull().sum()
```

Out[562]:
```
Id                0
MSSubClass        0
MSZoning          0
LotFrontage     259
LotArea           0
                 ...
MoSold            0
YrSold            0
SaleType          0
SaleCondition     0
SalePrice         0
Length: 81, dtype: int64
```

In [563]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
```

```
19  YearBuilt      1460 non-null   int64
20  YearRemodAdd   1460 non-null   int64
21  RoofStyle      1460 non-null   object
22  RoofMatl       1460 non-null   object
23  Exterior1st    1460 non-null   object
24  Exterior2nd    1460 non-null   object
25  MasVnrType     1452 non-null   object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
```

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

Analyzing using Sweetviz Library

In [16]:
```
import sweetviz
my_report = sweetviz.analyze([df,'Train'],target_feat='SalePrice')
```

```
:FEATURES DONE:                    |████████████████████| [100%]   01:
02  -> (00:00 left)
:PAIRWISE DONE:                    |████████████████████| [100%]   00:
09  -> (00:00 left)
```

```
Creating Associations graph... DONE!
```

In [18]:
```
my_report.show_html('Report.html')
```

```
Report Report.html was generated! NOTEBOOK/COLAB USERS: no browser will
pop up, the report is saved in your notebook/colab files.
```

In [19]:
```
##Comparing Train and Test dataframes
df2= pd.read_csv('test1.csv')
my_report1 = sweetviz.compare([df,'Train'],[df2,'test'],"SalePrice")
```

```
:FEATURES DONE:                    |████████████████████| [100%]   01:
13  -> (00:00 left)
:PAIRWISE DONE:                    |████████████████████| [100%]   00:
21  -> (00:00 left)
```

```
Creating Associations graph... DONE!
```

In [21]:
```
my_report1.show_html('Report.html')
```

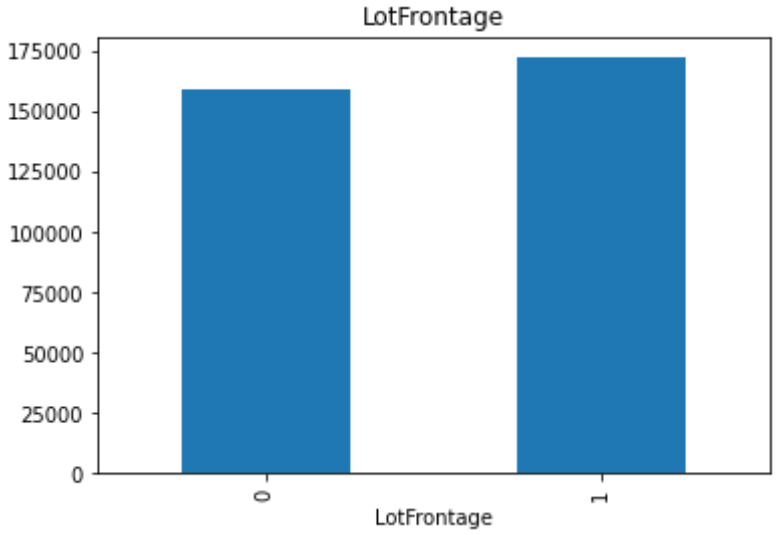
```
Report Report.html was generated! NOTEBOOK/COLAB USERS: no browser will
pop up, the report is saved in your notebook/colab files.
```

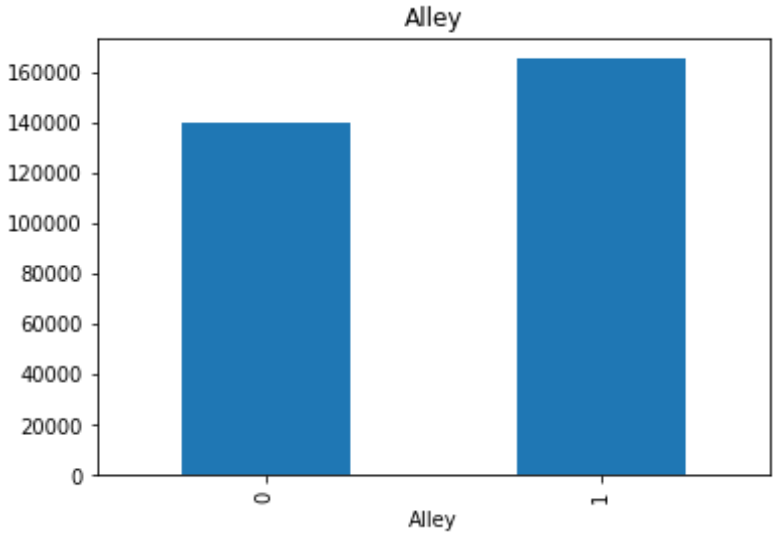
Since there are many missing values, we need to find a relationship b/w missing values and Sales Price

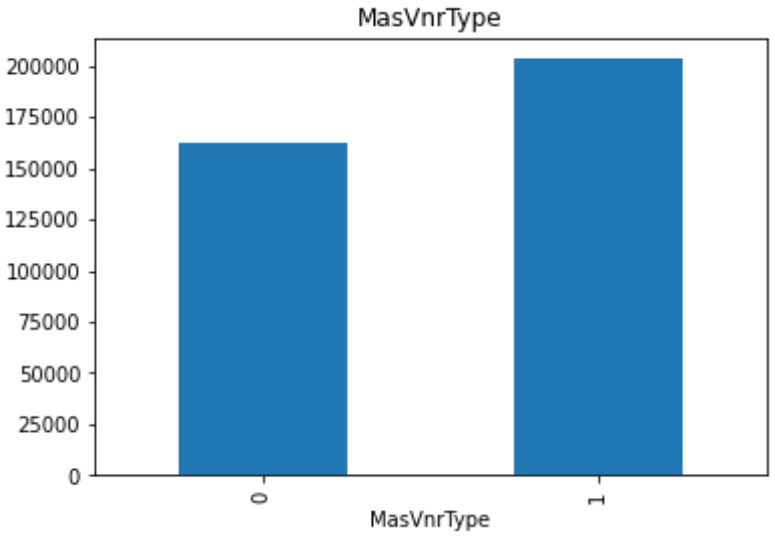
Let's plot some diagram for this relationship

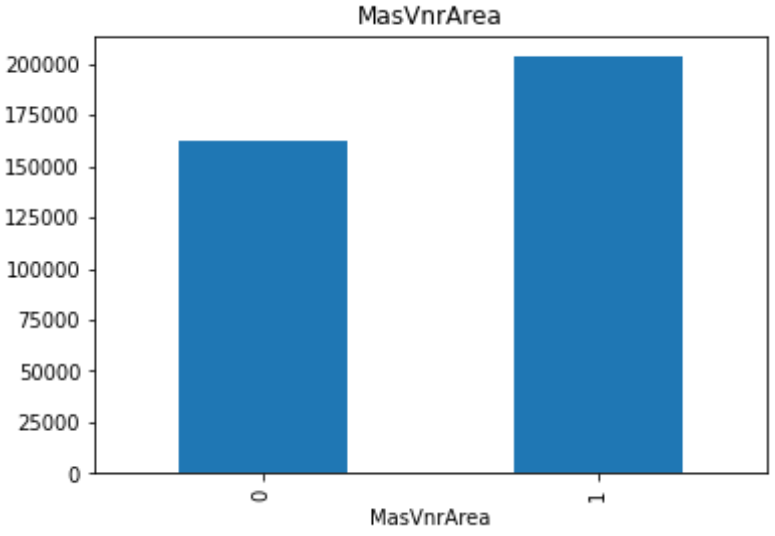
In [7]:
```
for features in features_with_na:
    data = df.copy()

    #Let's make a variable that indicates 1 if the observation was miss
ing or 0 otherwise
    data[features] = np.where(data[features].isnull(),1,0)

    #Let's calculate the median SalePrice where the information was mis
sing or present
    data.groupby(features)['SalePrice'].median().plot.bar()
    plt.title(features)
```

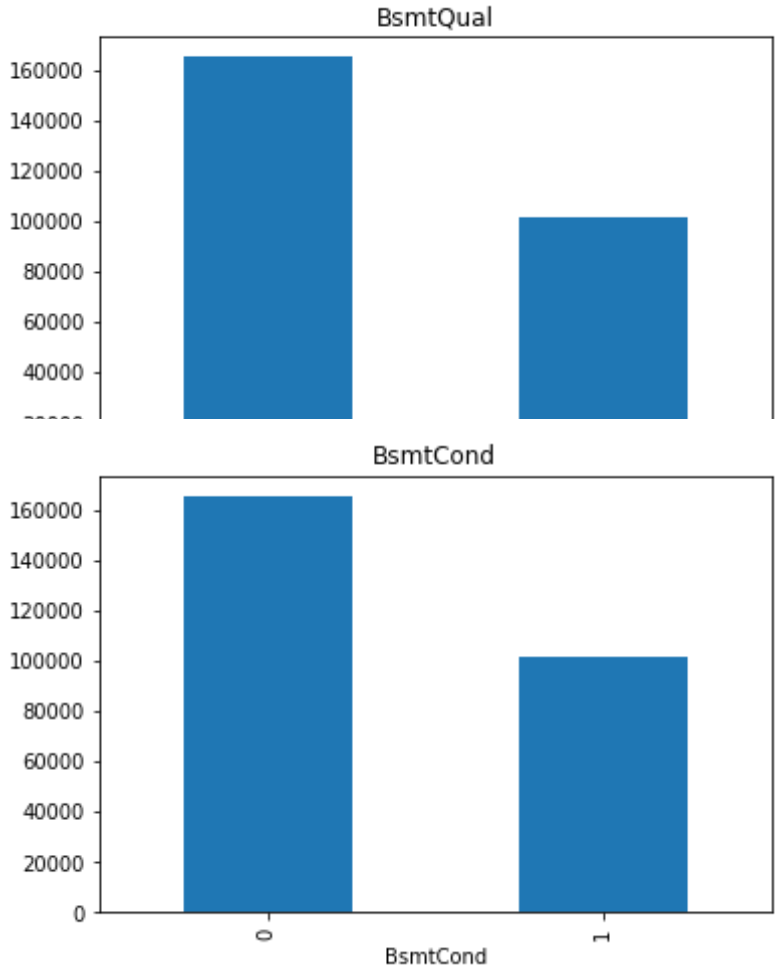
```
    plt.show()
```

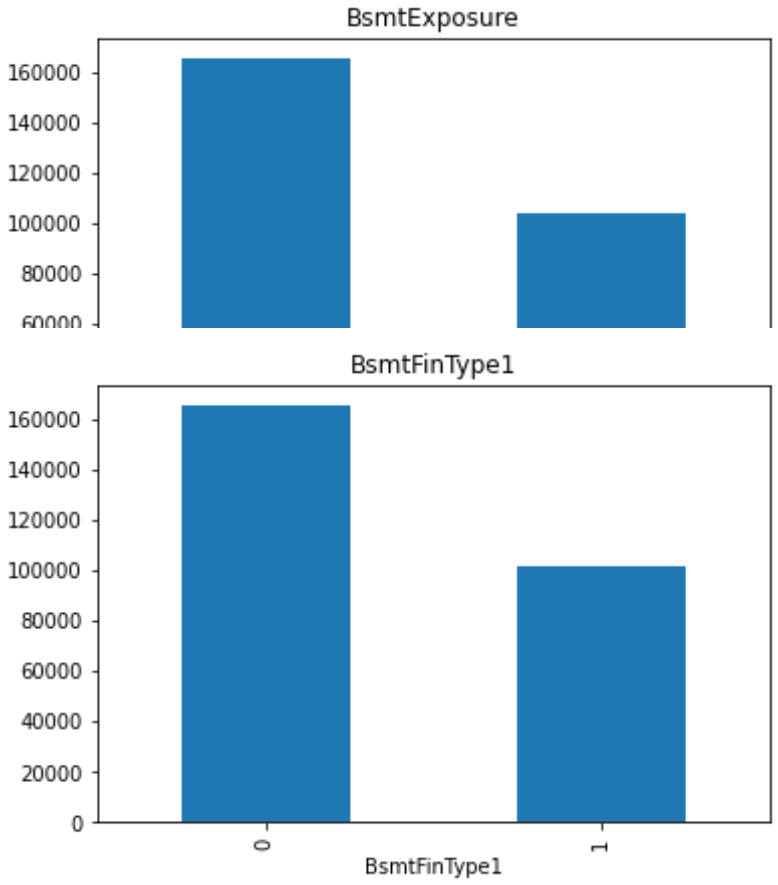


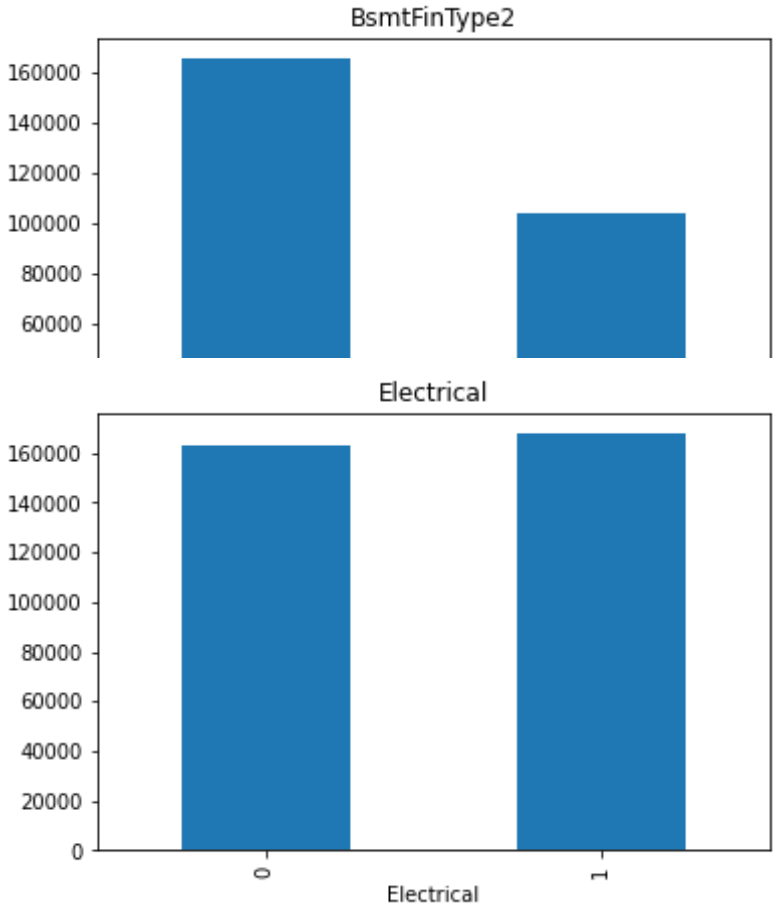


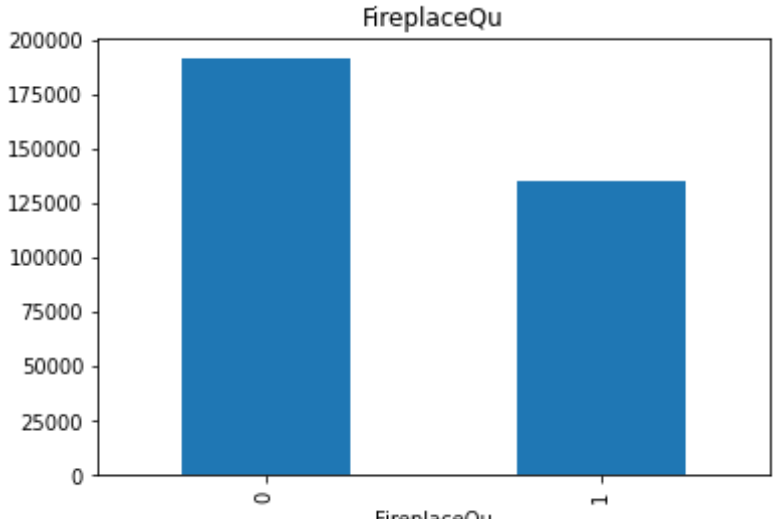


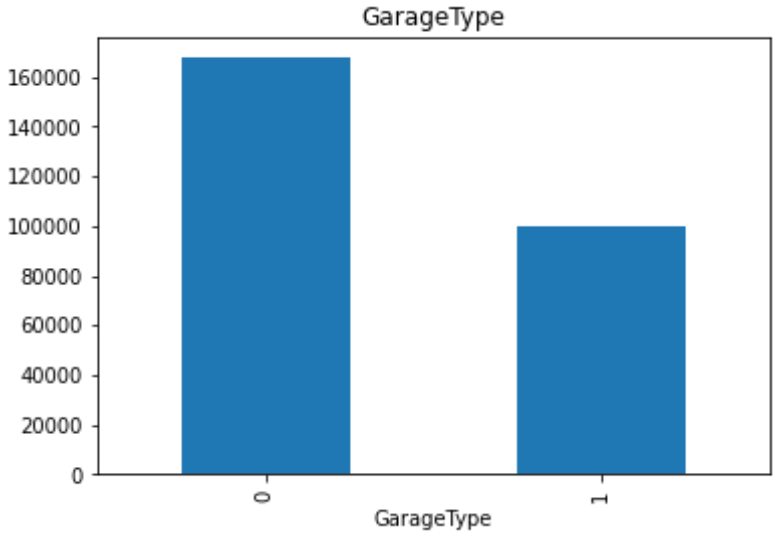


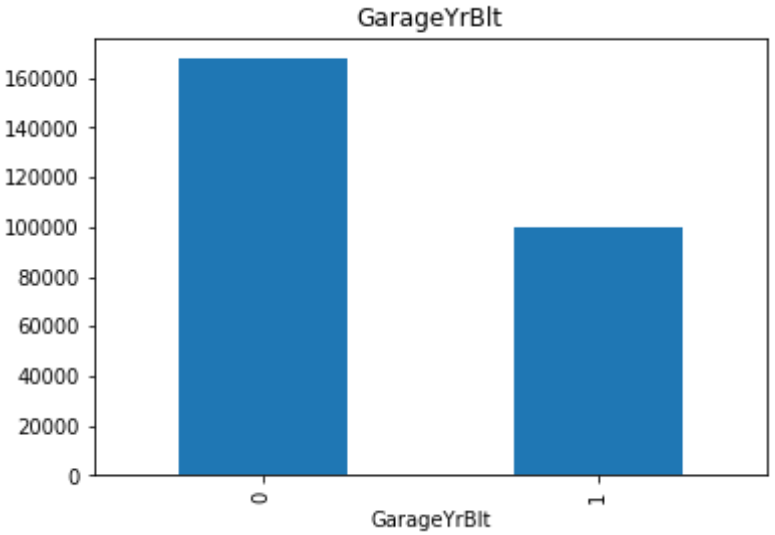


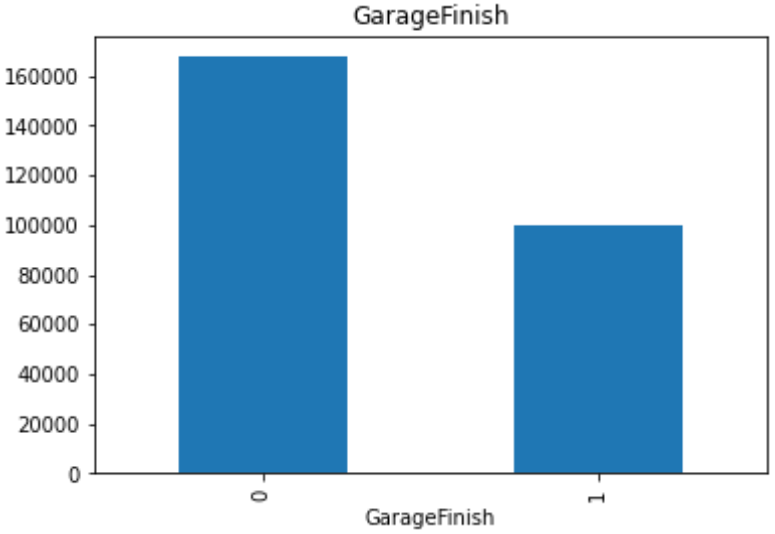


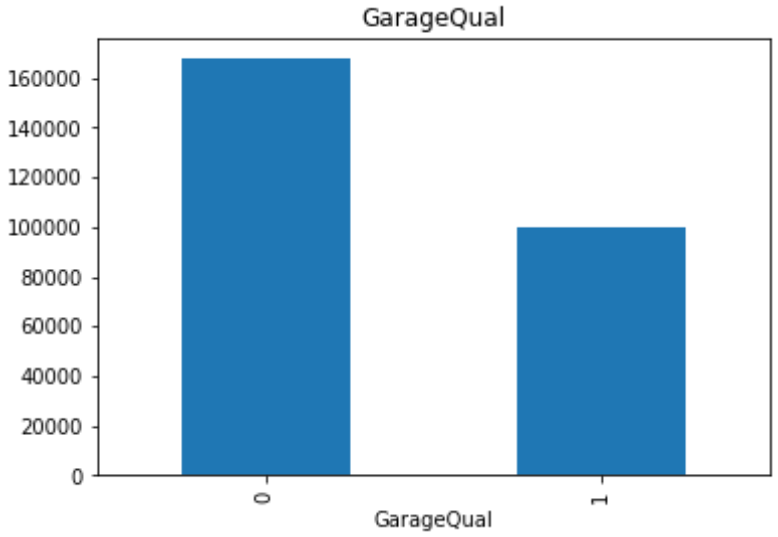


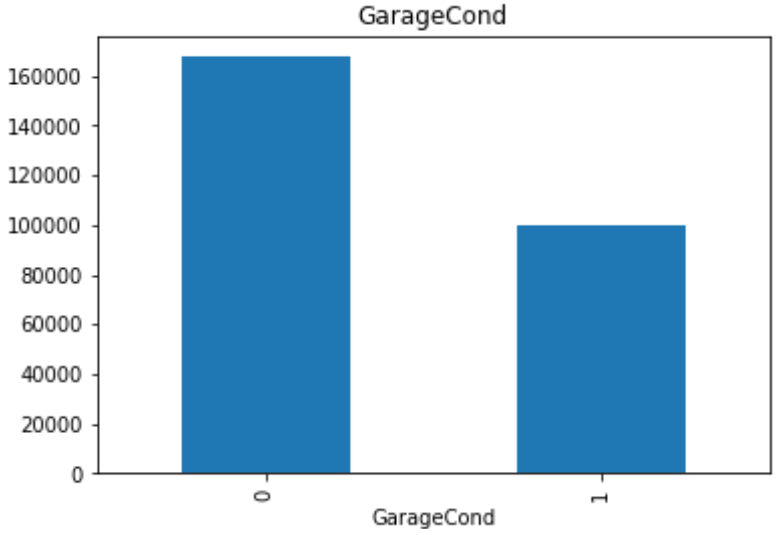


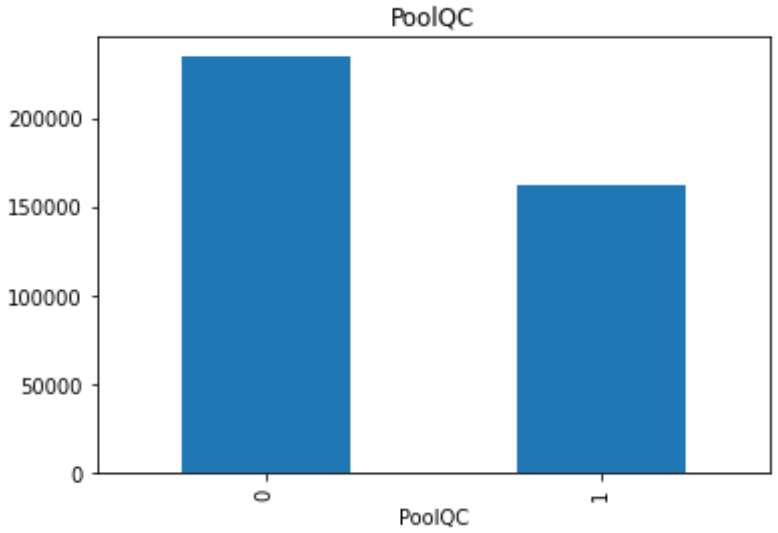


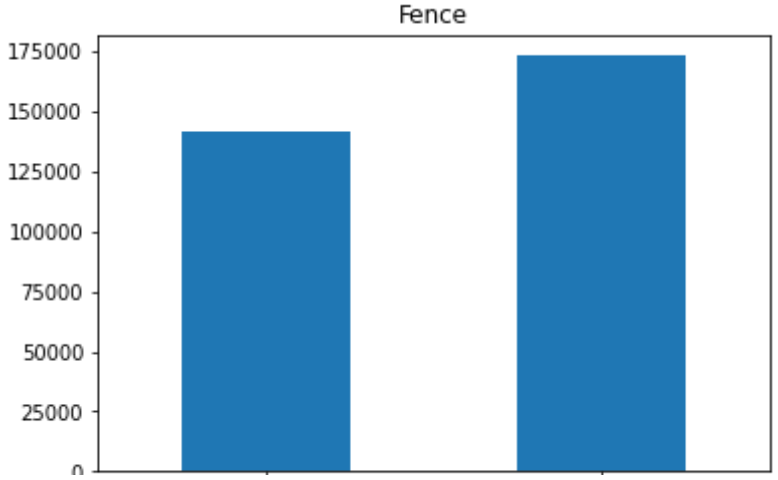


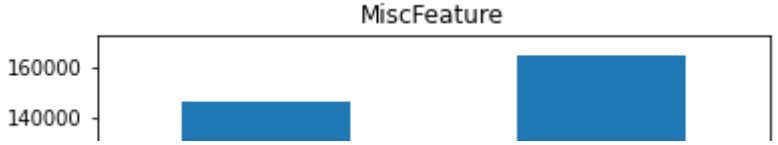


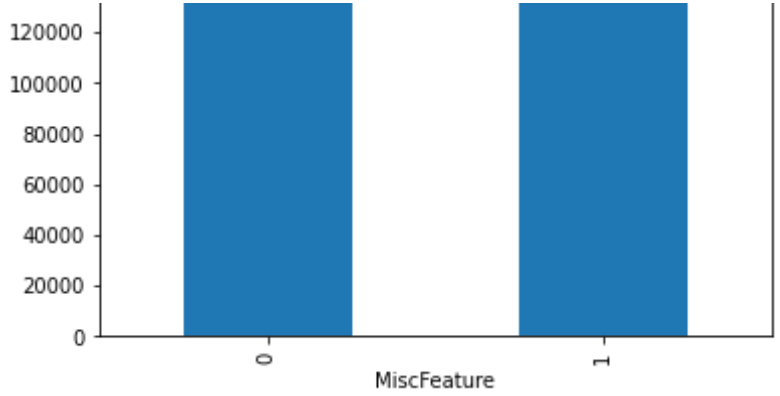












Here With the relation between the missing values and the dependent variable is clearly visible.So We need to replace these nan values with something meaningful which we will do in the Feature Engineering section

From the above dataset some of the features like Id is not required

In [8]:
```
print("Id of houses {}".format(len(df.Id)))
```

```
Id of houses 1460
```

Numerical Variables

In [9]:
```
numerical_features = [feature for feature in df.columns if df[feature].
dtypes!= 'O' ]
print("no. of numerical features {}".format(len(numerical_features)))

#Visualize the numerical features
df[numerical_features].head()
```

```
no. of numerical features 38
```

Out[9]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	M
0	1	60	65.0	8450	7	5	2003	2003	
1	2	20	80.0	9600	6	8	1976	1976	
2	3	60	68.0	11250	7	5	2001	2002	
3	4	70	60.0	9550	7	5	1915	1970	
4	5	60	84.0	14260	8	5	2000	2000	

Temporal Variables(e.g. Datetime variables)

From the Dataset we have 4 year variables. We have to extract information from the datetime variables like no. of years or no. of days. One example in this specific scenario can be difference in years between the year the house was built and the year the house was sold.

In [10]:
```
year_feature = [feature for feature in numerical_features if 'Yr' in fe
ature or 'Year' in feature]
year_feature
```

Out[10]:
```
['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
```

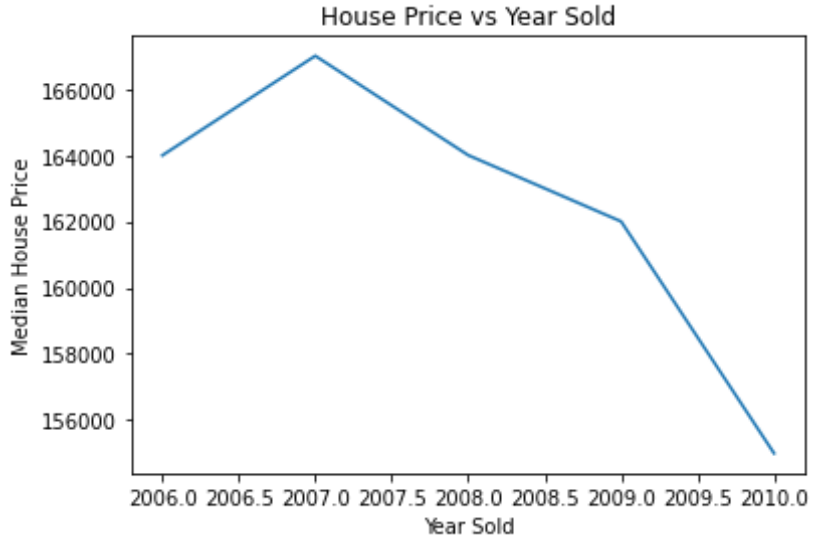
In [11]:
```
# Let's explore the content of these year variables
for feature in year_feature:
    print(feature,df[feature].unique())
```

```
YearBuilt [2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 1965 2005
1962 2006
 1960 1929 1970 1967 1958 1930 2002 1968 2007 1951 1957 1927 1920 1966
 1959 1994 1954 1953 1955 1983 1975 1997 1934 1963 1981 1964 1999 1972
 1921 1945 1982 1998 1956 1948 1910 1995 1991 2009 1950 1961 1977 1985
 1979 1885 1919 1990 1969 1935 1988 1971 1952 1936 1923 1924 1984 1926
 1940 1941 1987 1986 2008 1908 1892 1916 1932 1918 1912 1947 1925 1900
 1980 1989 1992 1949 1880 1928 1978 1922 1996 2010 1946 1913 1937 1942
 1938 1974 1893 1914 1906 1890 1898 1904 1882 1875 1911 1917 1872 1905]
YearRemodAdd [2003 1976 2002 1970 2000 1995 2005 1973 1950 1965 2006 19
62 2007 1960
 2001 1967 2004 2008 1997 1959 1990 1955 1983 1980 1966 1963 1987 1964
 1972 1996 1998 1989 1953 1956 1968 1981 1992 2009 1982 1961 1993 1999
 1985 1979 1977 1969 1958 1991 1971 1952 1975 2010 1984 1986 1994 1988
 1954 1957 1951 1978 1974]
GarageYrBlt [2003. 1976. 2001. 1998. 2000. 1993. 2004. 1973. 1931. 193
9. 1965. 2005.
 1962. 2006. 1960. 1991. 1970. 1967. 1958. 1930. 2002. 1968. 2007. 200
8.
 1957. 1920. 1966. 1959. 1995. 1954. 1953.   nan 1983. 1977. 1997. 198
5.
 1963. 1981. 1964. 1999. 1935. 1990. 1945. 1987. 1989. 1915. 1956. 194
8.
 1974. 2009. 1950. 1961. 1921. 1900. 1979. 1951. 1969. 1936. 1975. 197
1.
 1923. 1984. 1926. 1955. 1986. 1988. 1916. 1932. 1972. 1918. 1980. 192
4.
 1996. 1940. 1949. 1994. 1910. 1978. 1982. 1992. 1925. 1941. 2010. 192
7.
 1947. 1937. 1942. 1938. 1952. 1928. 1922. 1934. 1906. 1914. 1946. 190
8.
 1929. 1933.]
YrSold [2008 2007 2006 2009 2010]
```

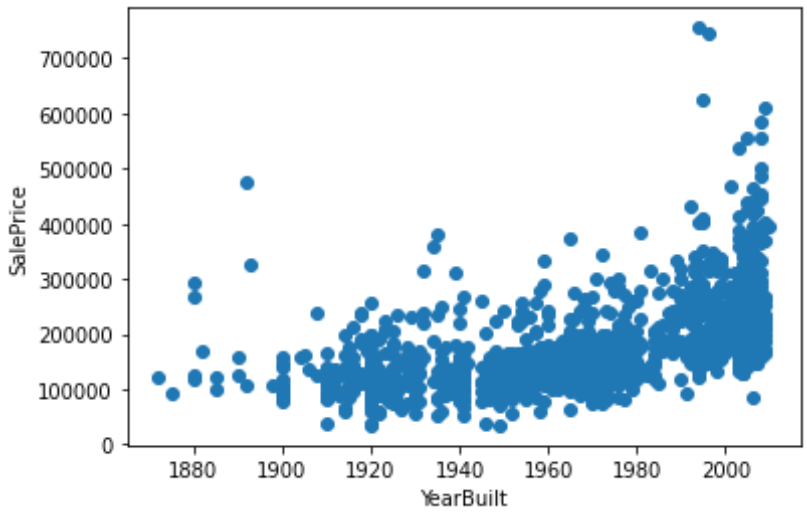
In [12]:
```
# Let's analyze the Temporal Datetime variable
##We will check if there is a relation b/w year the house is sold and t
he SalePrice

df.groupby('YrSold')['SalePrice'].median().plot()
```

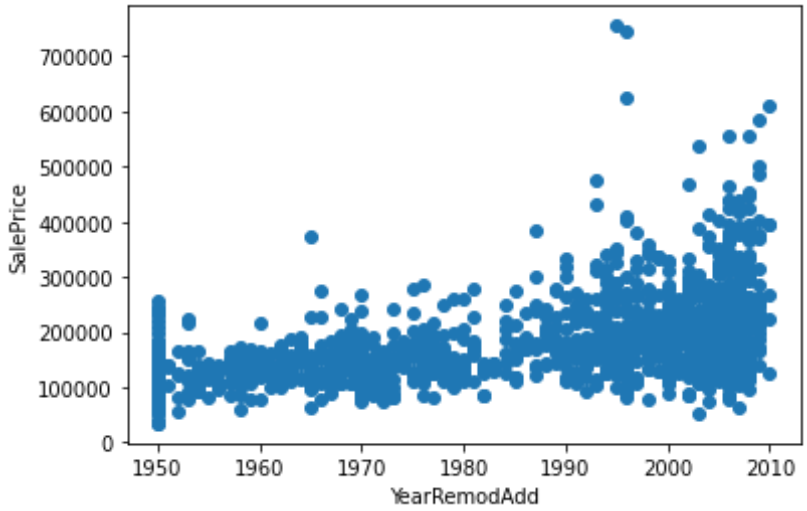
```
plt.xlabel('Year Sold')
plt.ylabel('Median House Price')
plt.title('House Price vs Year Sold')
```

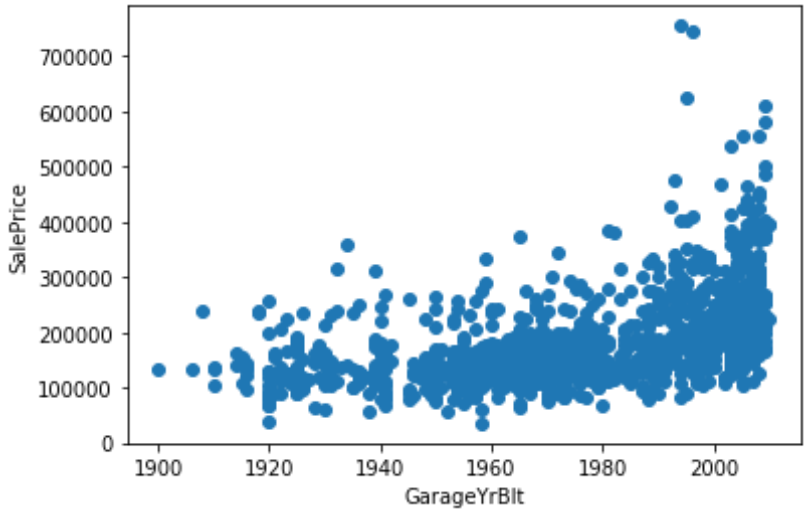
Out[12]: Text(0.5, 1.0, 'House Price vs Year Sold')



In [13]:
```
# Here we will compare the difference b/w all year features with SalePr
ice
for feature in year_feature:
    if feature!= 'YrSold':
        data=df.copy()
        #df[feature]=df['YrSold']-df[feature]

        plt.scatter(df[feature],df['SalePrice'])
        plt.xlabel(feature)
        plt.ylabel('SalePrice')
        plt.show()
```







In [14]:
```
## Numerical variables are usually of 2 type
## 1. Continous variable 2. Discrete Variables

discrete_feature=[feature for feature in numerical_features if len(df[feature].unique())<25 and feature not in year_feature]
print("Discrete Variables Count: {}".format(len(discrete_feature)))
```

```
Discrete Variables Count: 17
```

In [15]:
```
discrete_feature
```

Out[15]:
```
['MSSubClass',
 'OverallQual',
 'OverallCond',
 'LowQualFinSF',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
```

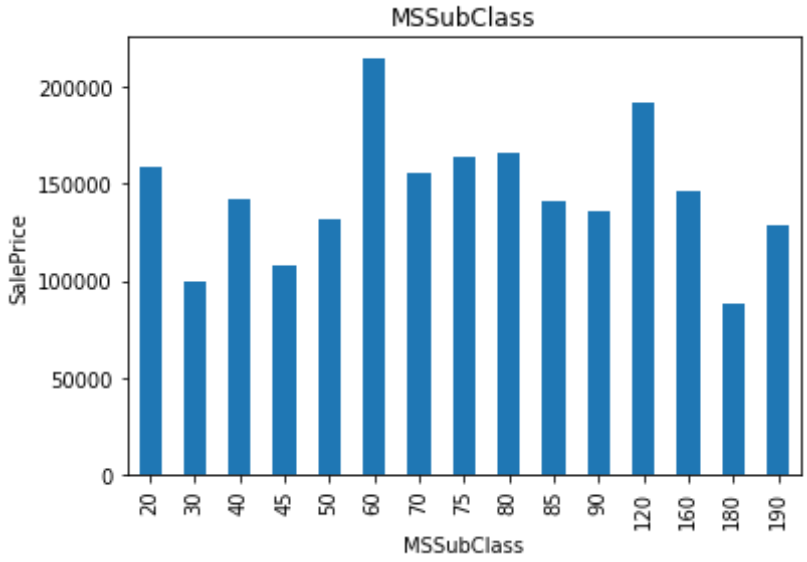
```
 'KitchenAbvGr',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageCars',
 '3SsnPorch',
 'PoolArea',
 'MiscVal',
 'MoSold']
```

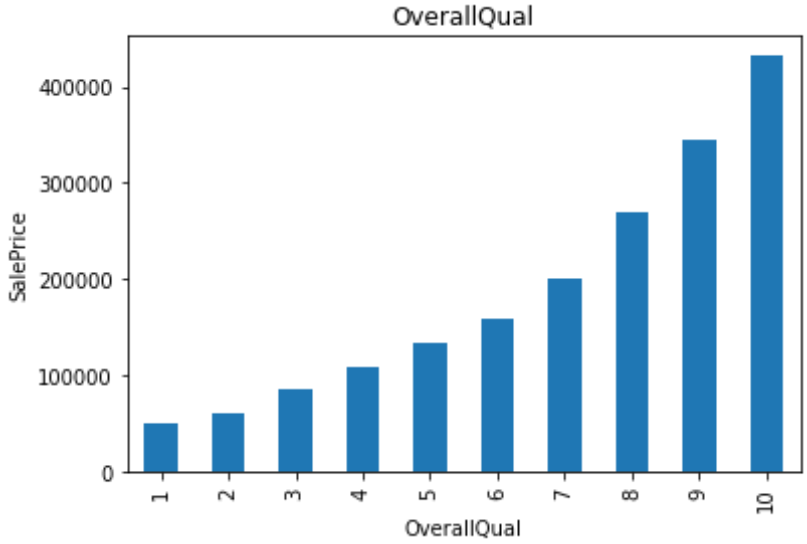
In [16]:
```
df[discrete_feature].head()
```

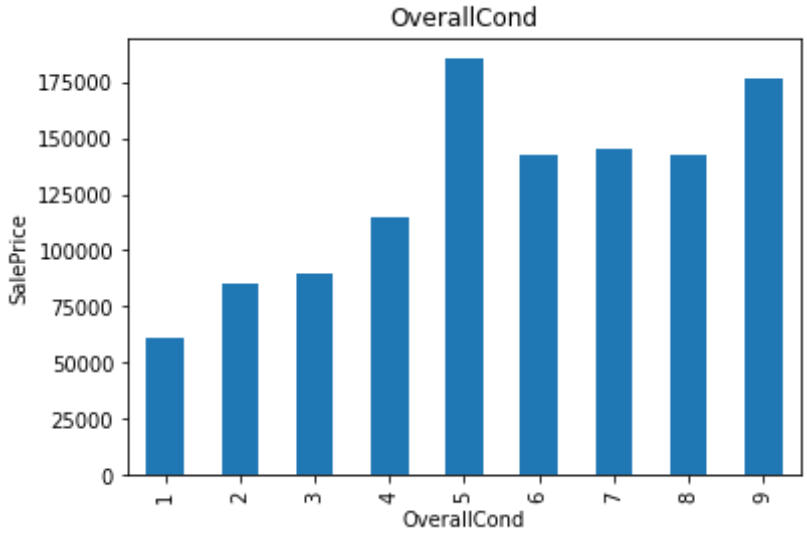
Out[16]:

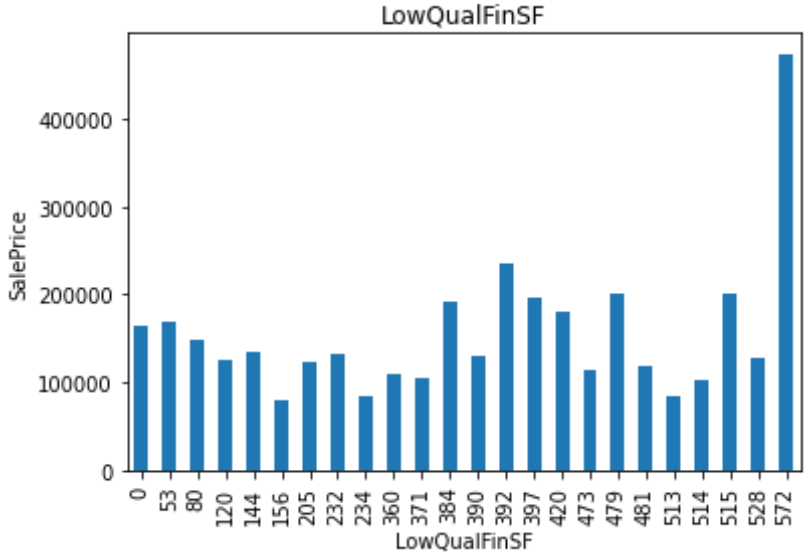
	MSSubClass	OverallQual	OverallCond	LowQualFinSF	BsmtFullBath	BsmtHalfBath	FullBath
0	60	7	5	0	1	0	2
1	20	6	8	0	0	1	2
2	60	7	5	0	1	0	2
3	70	7	5	0	1	0	1
4	60	8	5	0	1	0	2

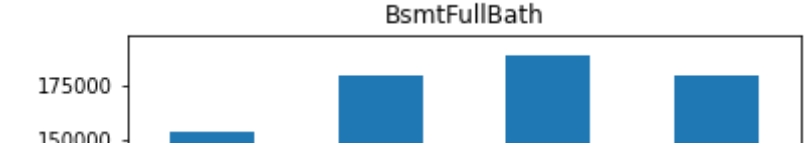
In [17]:
```
## Lets Find the realtionship between them and Sale PRice

for feature in discrete_feature:
    data=df.copy()
    data.groupby(feature)['SalePrice'].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.title(feature)
    plt.show()
```

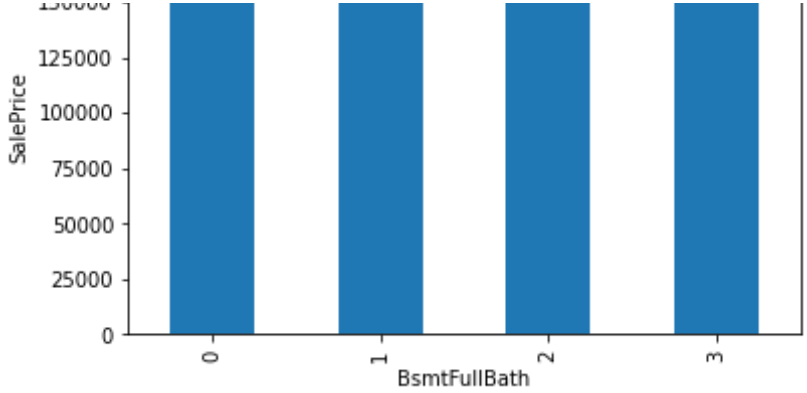


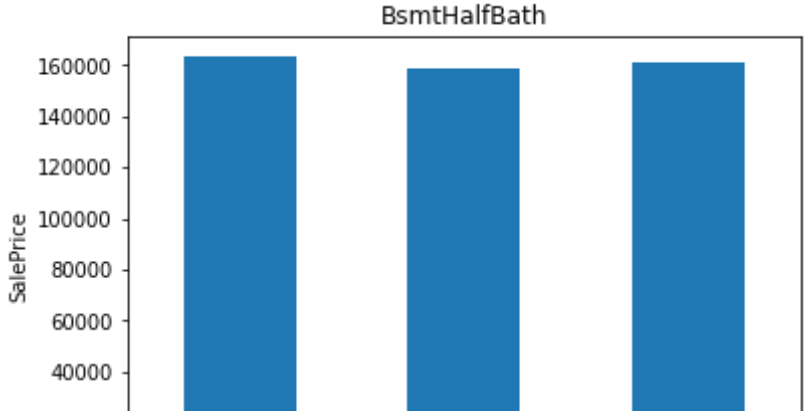


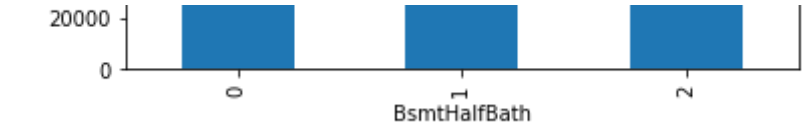


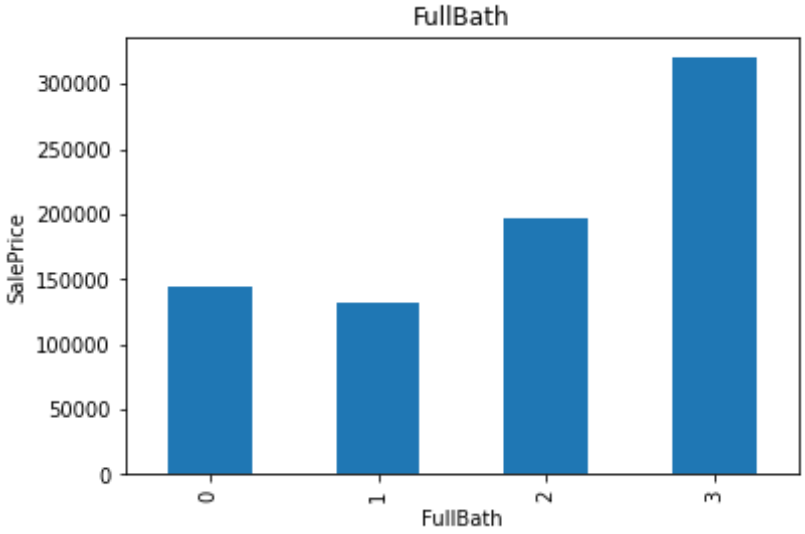


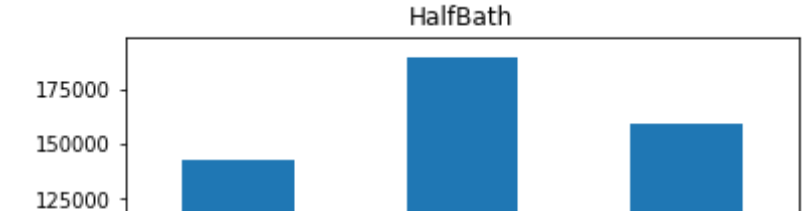


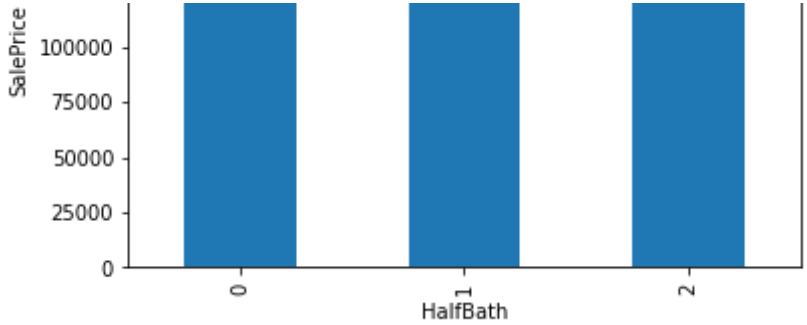


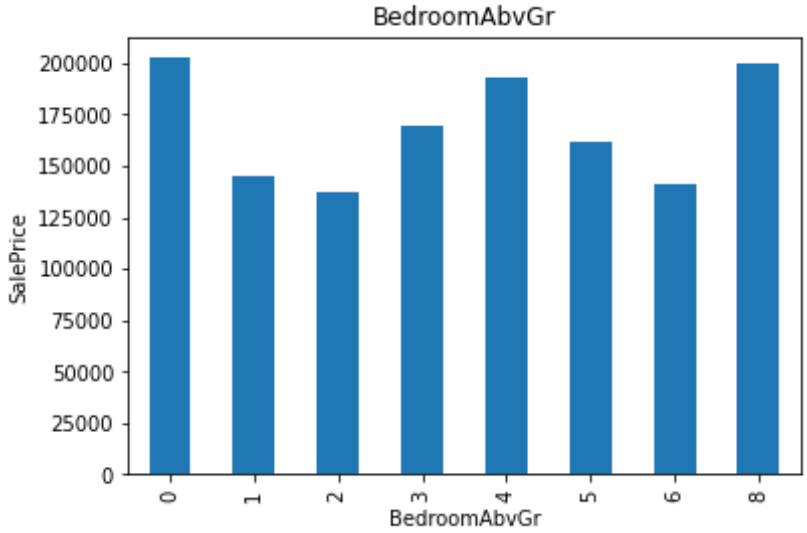


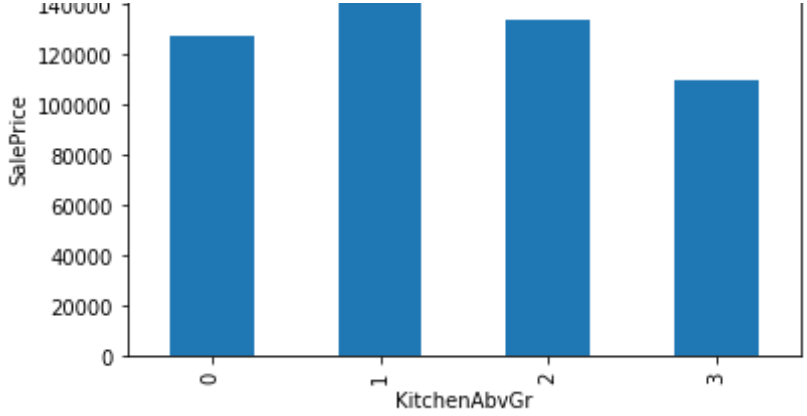


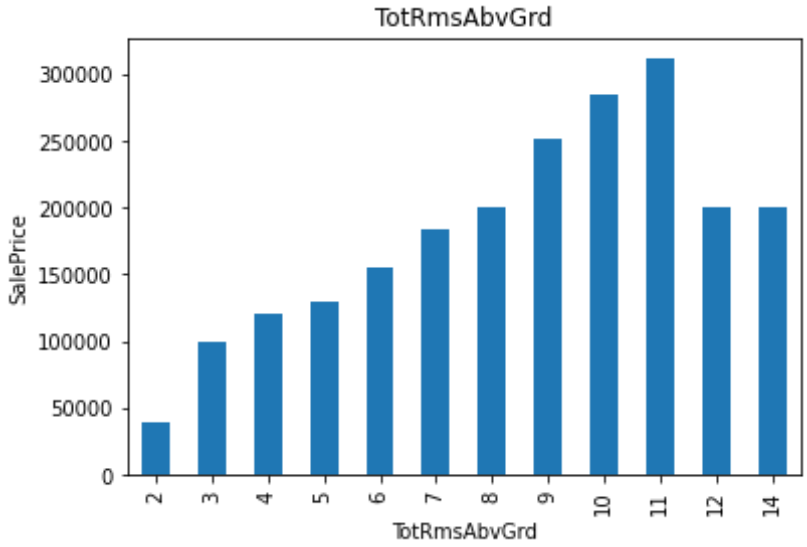


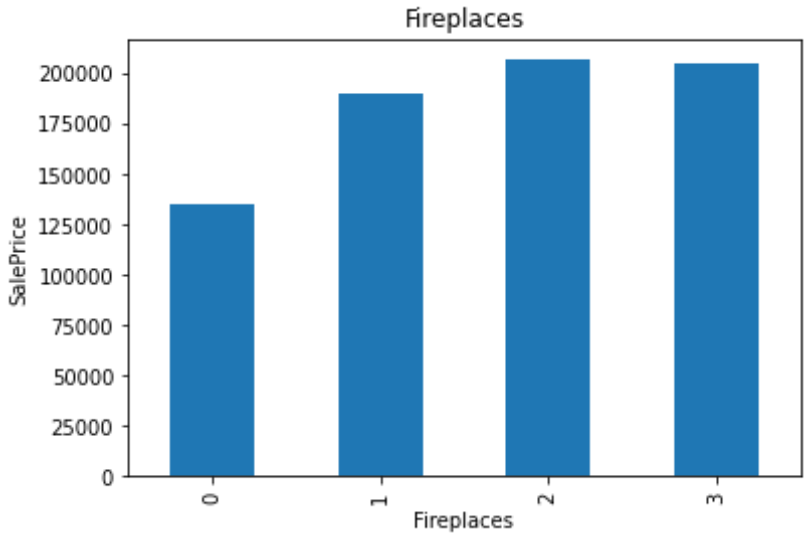


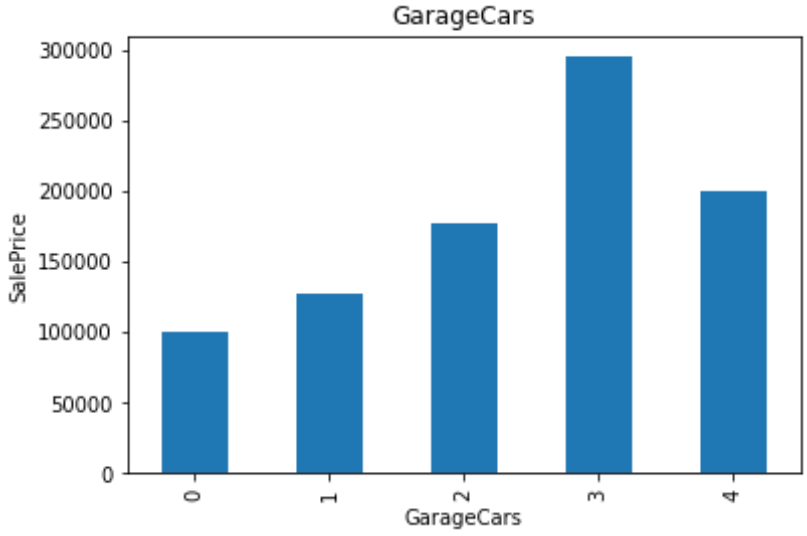


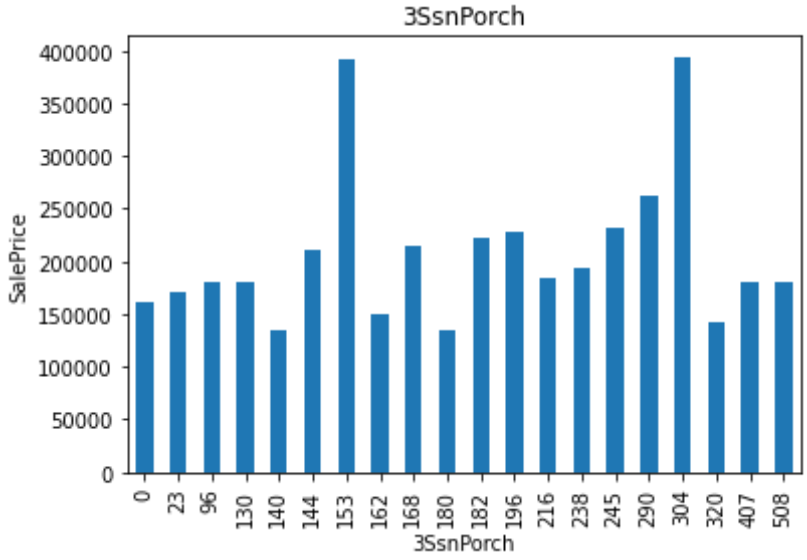


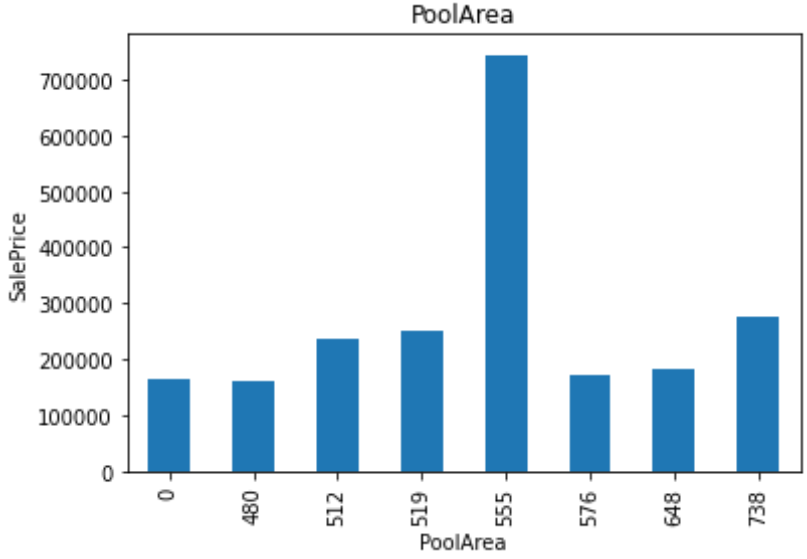


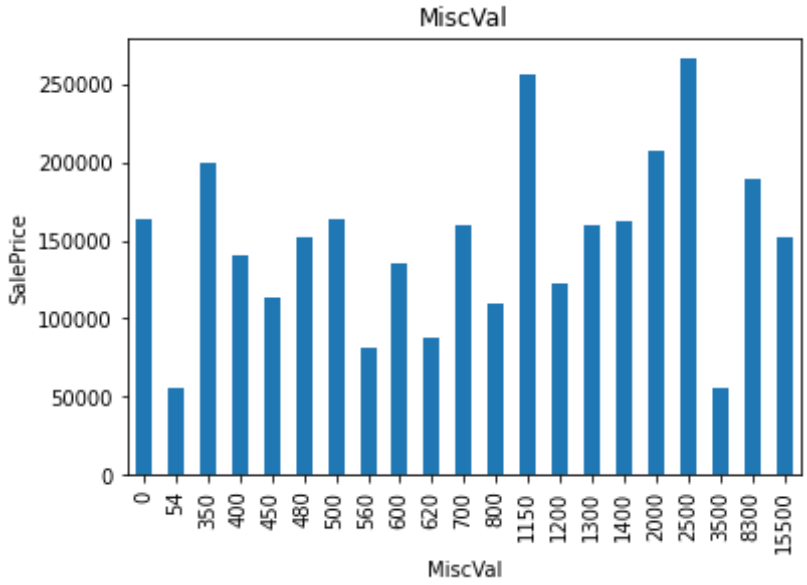


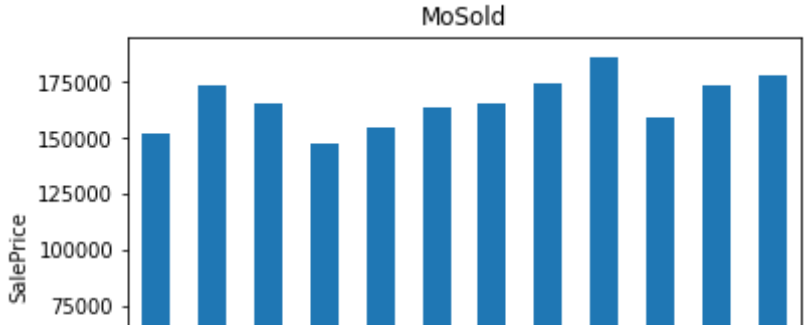


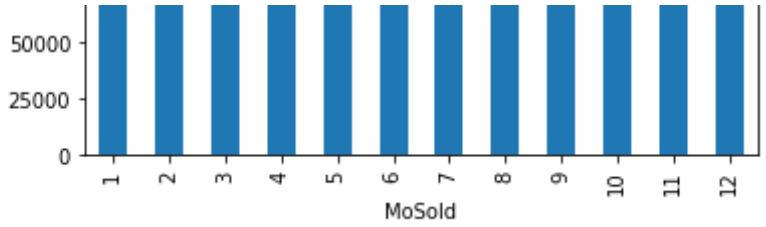












Continuous Variable

In [18]:
```
continuous_feature= [feature for feature in numerical_features if featu
re not in discrete_feature+year_feature+['Id']]
print("Continuous feature Count {}".format(len(continuous_feature)))
```

```
Continuous feature Count 16
```

In [19]:
```
continuous_feature
```

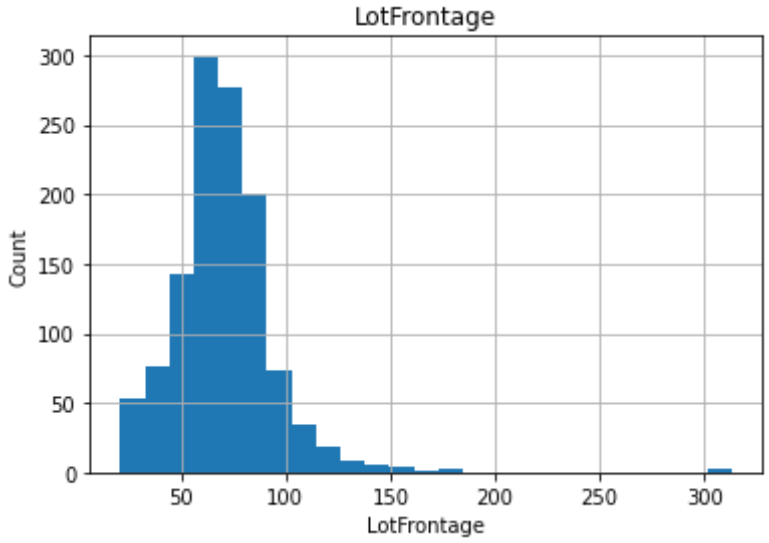
Out[19]:
```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch',
 'SalePrice']
```

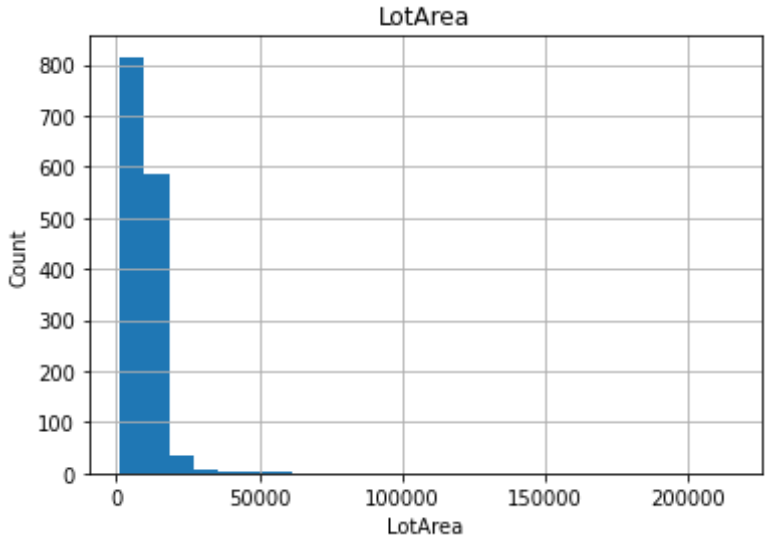
In [20]:
```
df[continuous_feature].head()
```

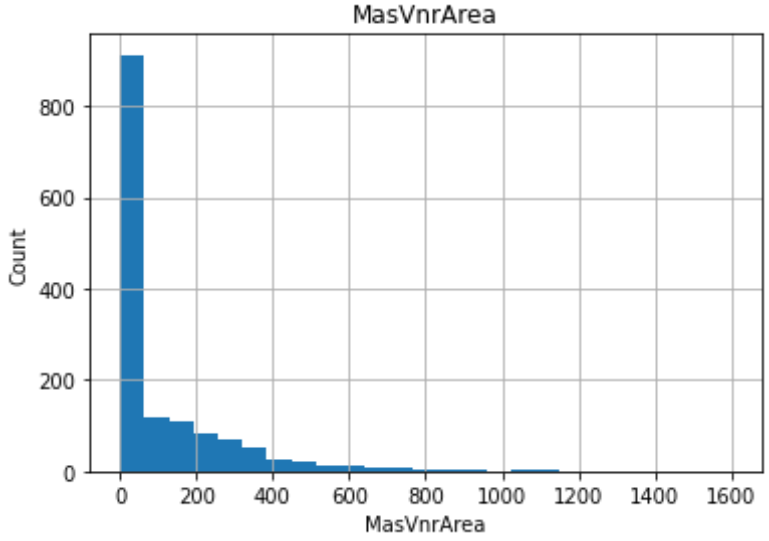
Out[20]:

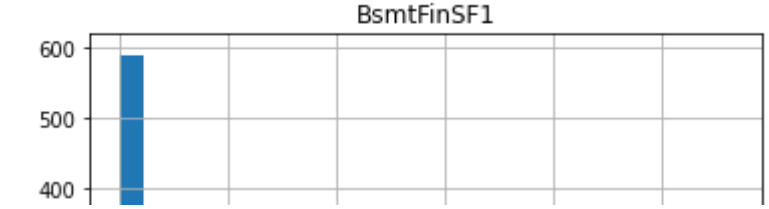
	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	1stFl
0	65.0	8450	196.0	706	0	150	856	
1	80.0	9600	0.0	978	0	284	1262	12
2	68.0	11250	162.0	486	0	434	920	
3	60.0	9550	0.0	216	0	540	756	
4	84.0	14260	350.0	655	0	490	1145	1

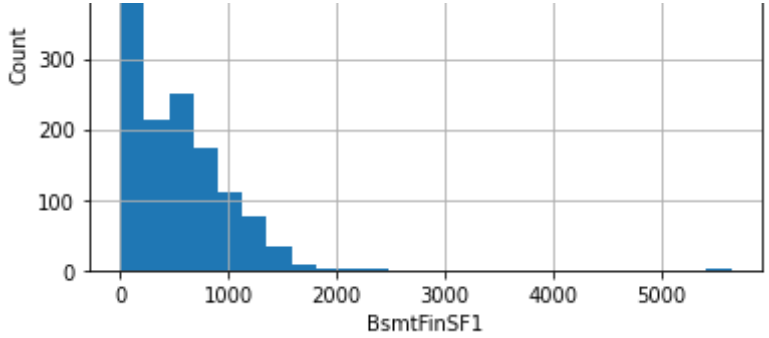
In [21]:
```
for feature in continuous_feature:
    data=df.copy()
    data[feature].hist(bins=25)
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.title(feature)
    plt.show()
```

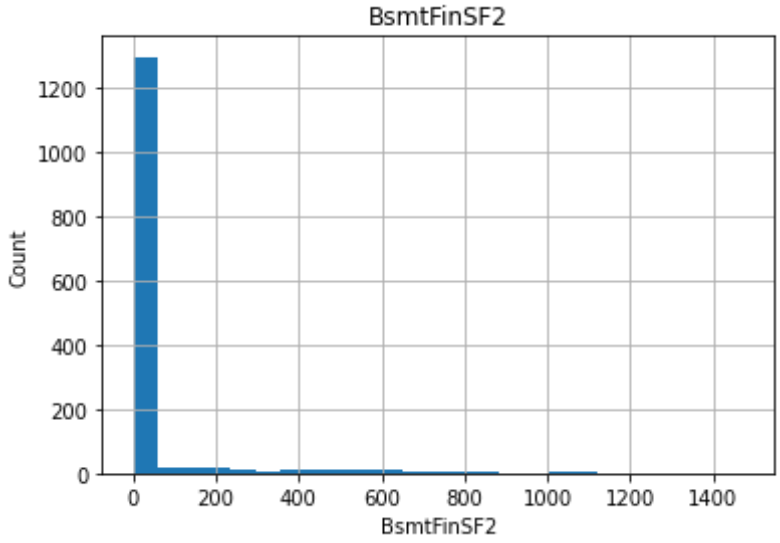


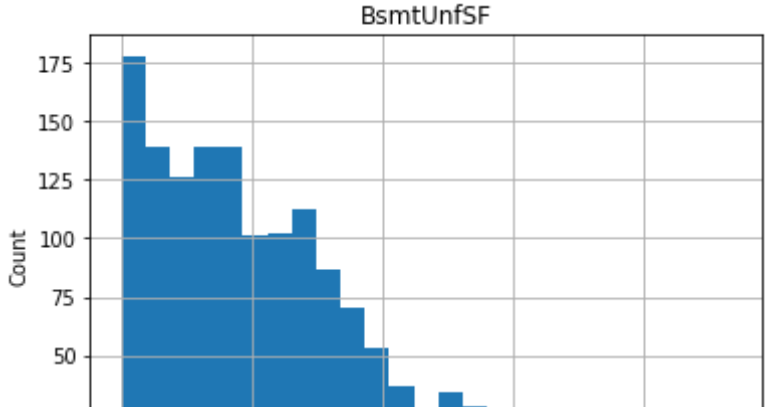


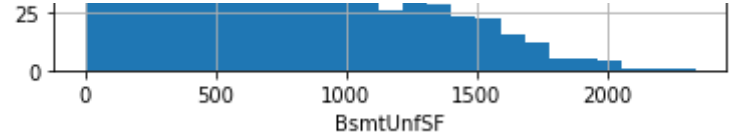


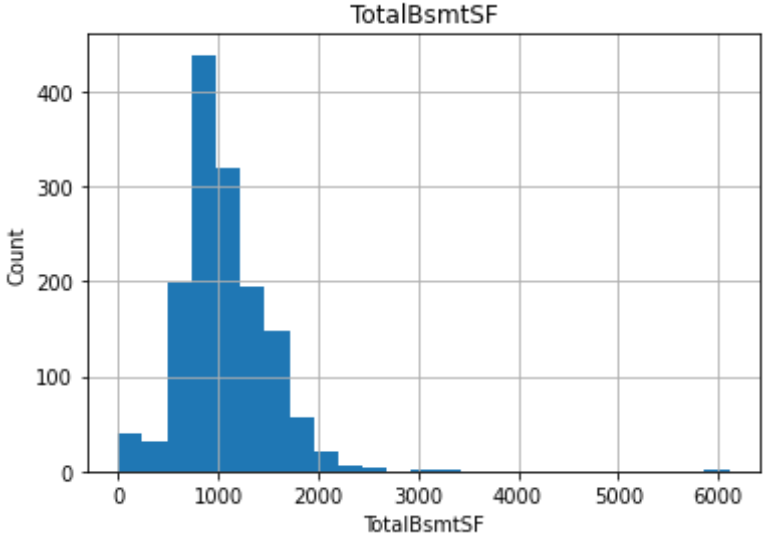


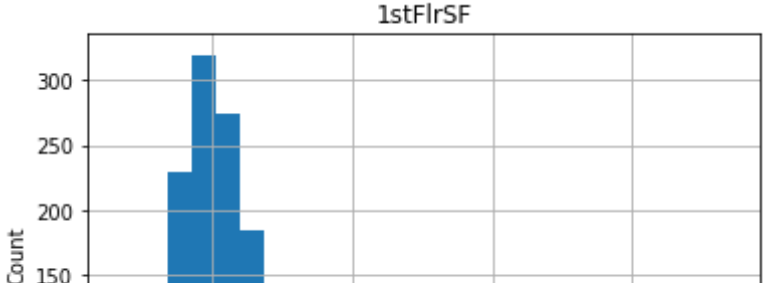


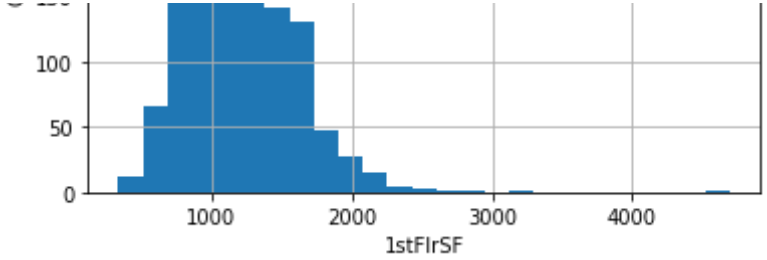


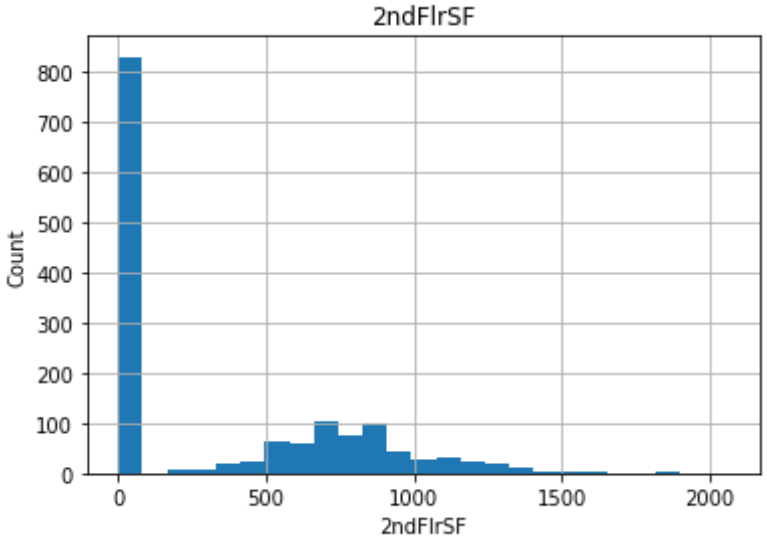


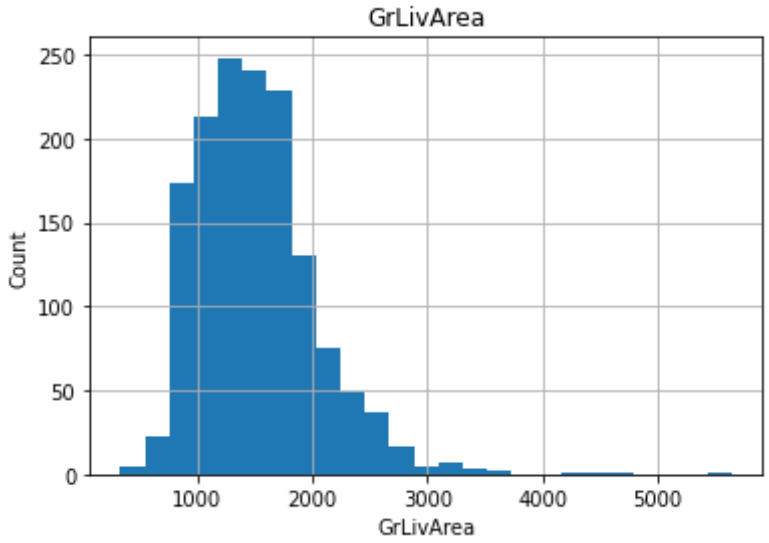


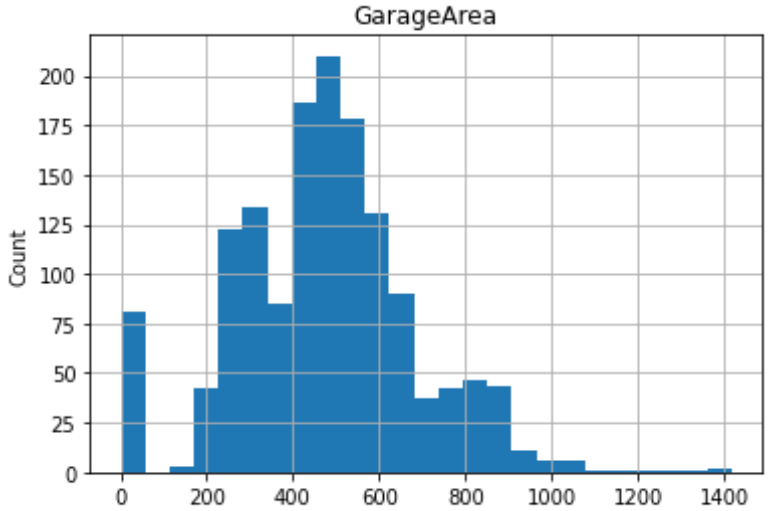




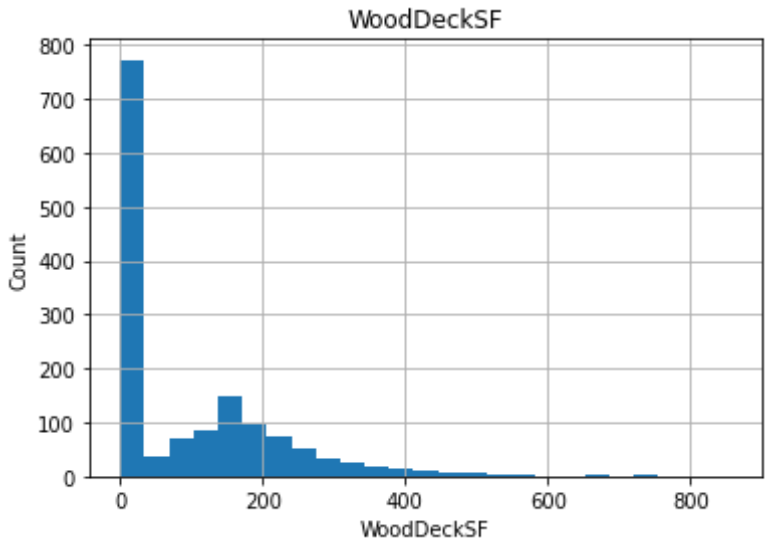


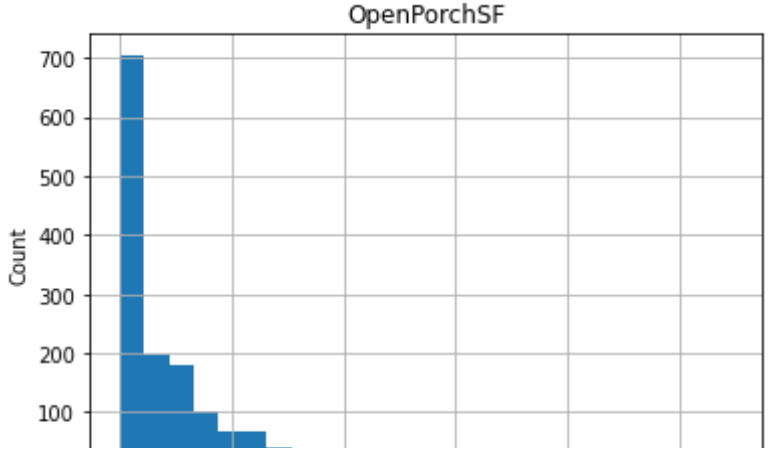


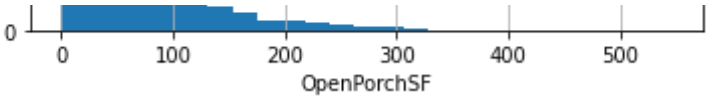


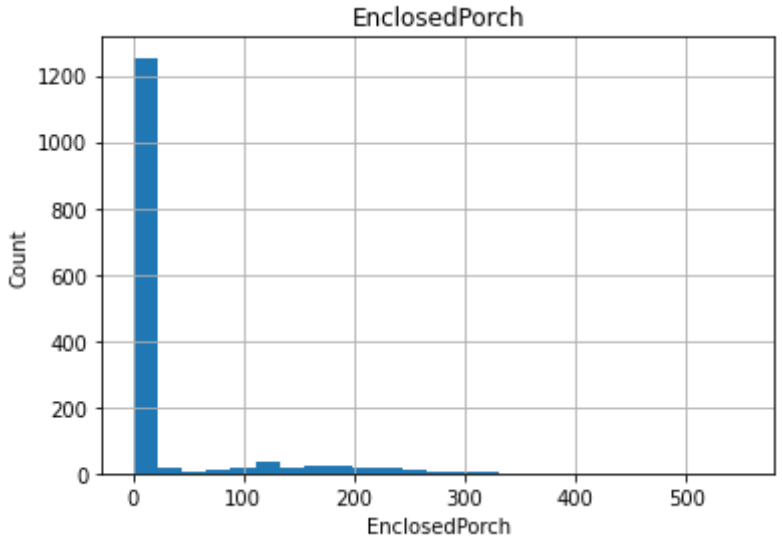


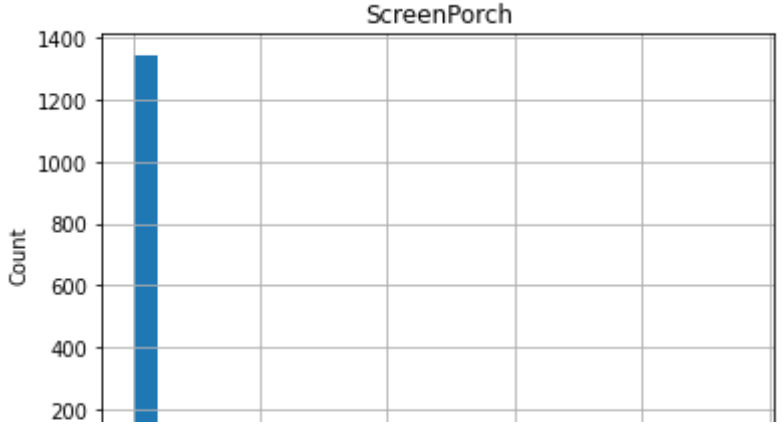
GarageArea

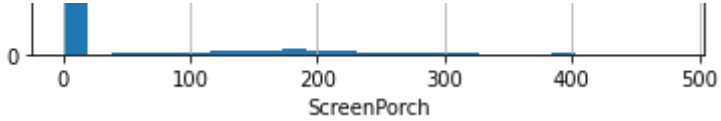


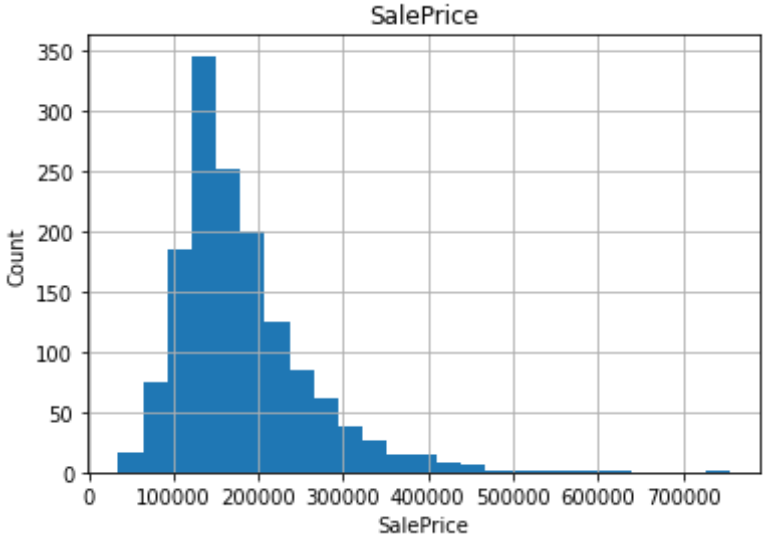








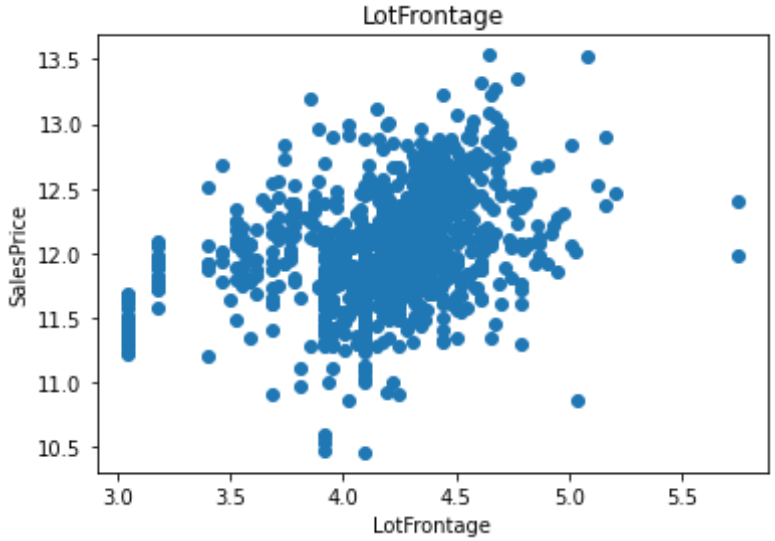


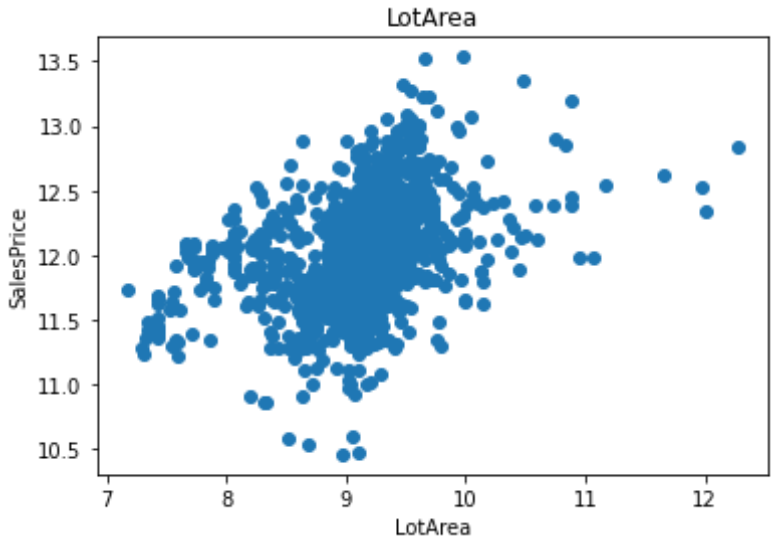


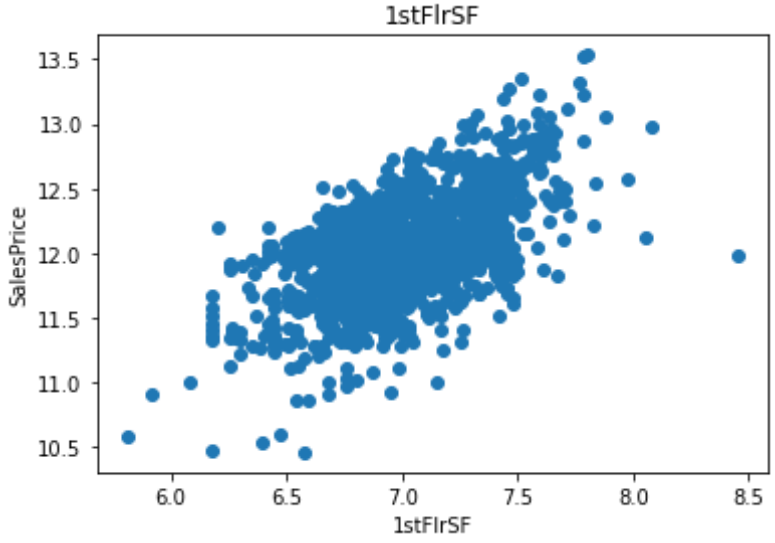
We will be using Logarithmic Transformation

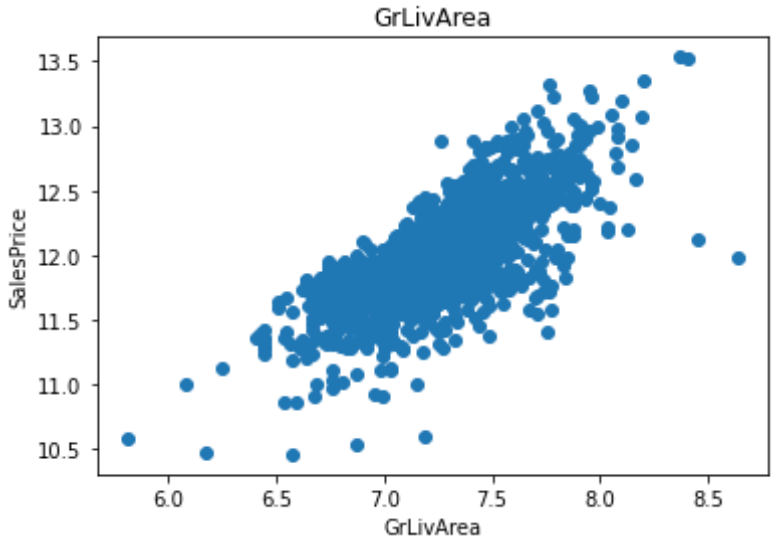
In [166]:
```
for feature in continuous_feature:
    data=df.copy()
    if 0 in data[feature].unique():
        pass
    else:
        data[feature]=np.log(data[feature])
```

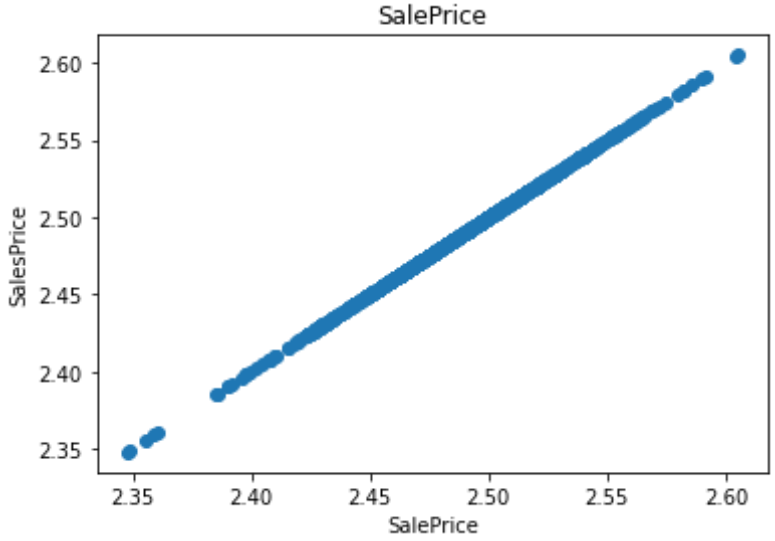
```
        data['SalePrice']=np.log(data['SalePrice'])
        plt.scatter(x=data[feature],y=data['SalePrice'])
        plt.xlabel(feature)
        plt.ylabel('SalesPrice')
        plt.title(feature)
        plt.show()
```











In [22]:
```
df_copy = df.copy()
```

In [23]:
```
df_copy.head()
```

Out[23]:

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

In [24]:
```
df = df_copy
```

We can observe monotonic relationships b/w the continuous variables and the dependent variable

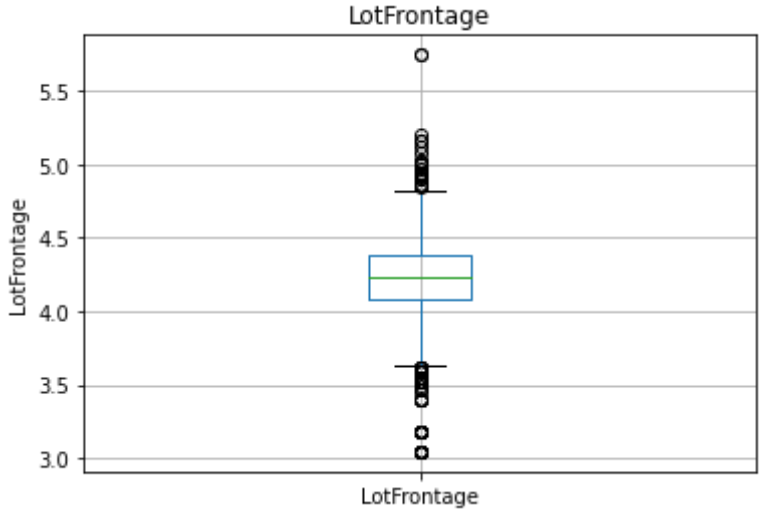
In [25]:
```
df.head()
```

Out[25]:

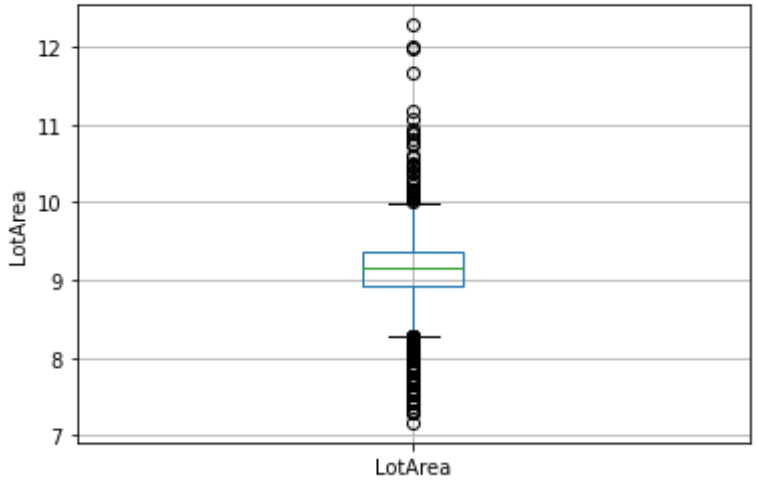
	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utili
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	All
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	All
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	All
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	All
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	All

Outliers

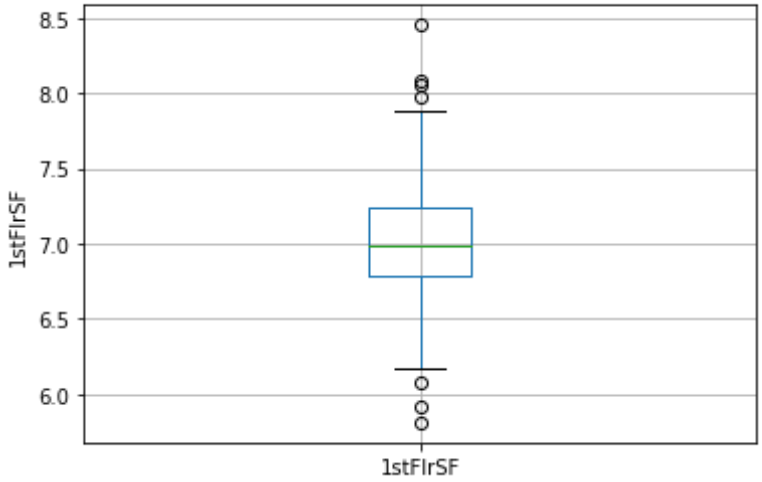
In [26]:
```
for feature in continuous_feature:
    data=df.copy()

    if 0 in data[feature].unique():
        pass
    else:
        data[feature]=np.log(data[feature])
        data.boxplot(column=feature)
        plt.ylabel(feature)
        plt.title(feature)
        plt.show()
```



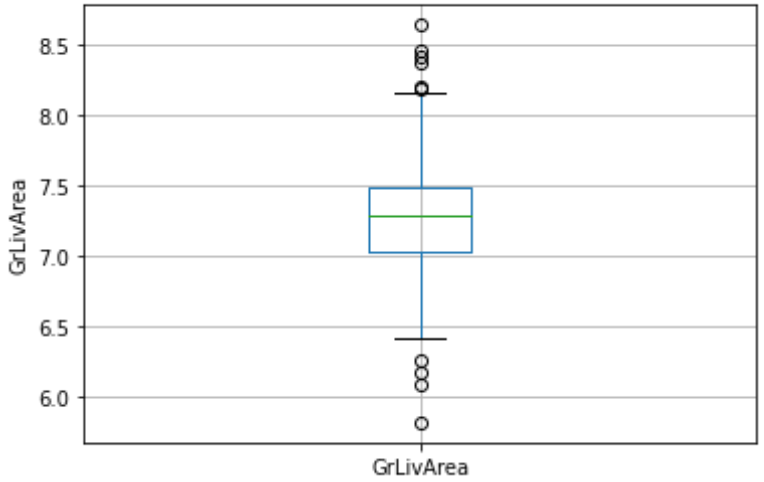
LotArea

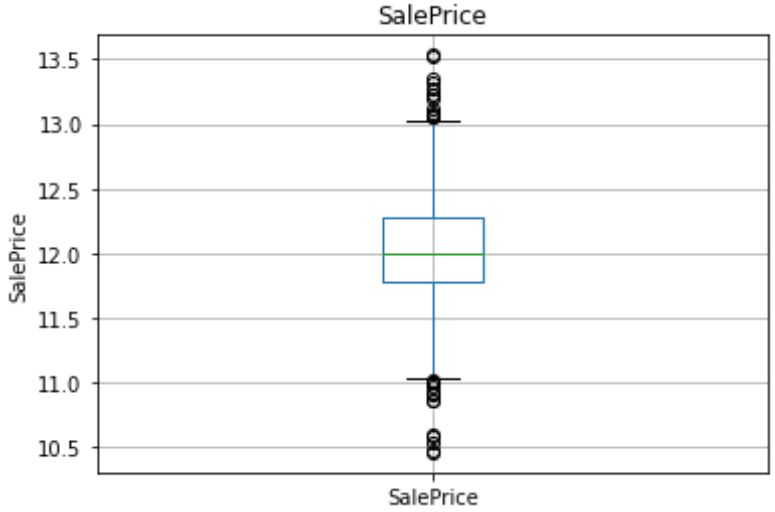


1stFlrSF



GrLivArea





In [27]:
```
data.head()
```

Out[27]:

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

In [28]:
```
data=df.copy()
```

In [29]:
```
continuous_feature
```

Out[29]:
```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch',
 'SalePrice']
```

In [30]:
```
for feature in continuous_feature:
        IQR = np.percentile(df[feature],75) - np.percentile(df[feature
],25)
        lb = np.percentile(df[feature],25)-IQR*1.5
        ub = np.percentile(df[feature],75)+IQR*1.5
```

```
        df[feature] = np.where(df[feature]>ub,ub,df[feature])
        df[feature] = np.where(df[feature]<lb,lb,df[feature])
        df[feature] = np.log1p(df[feature])
```

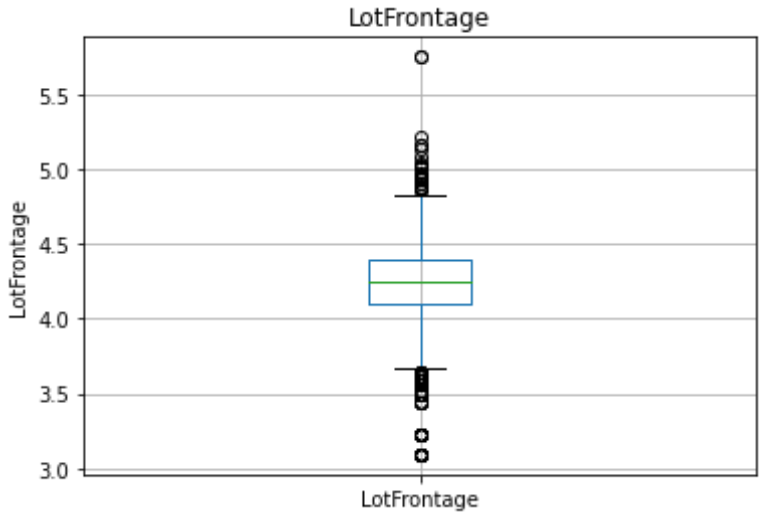
In [ ]:

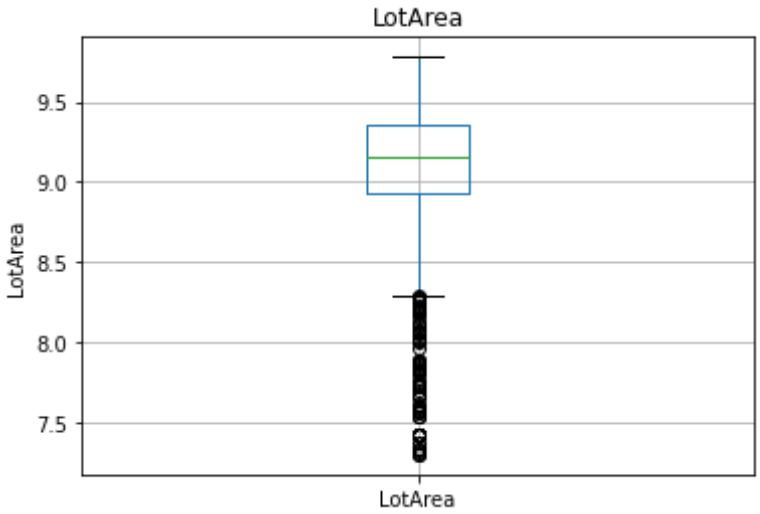
In [588]:
```
#import scipy
#for feature in continuous_feature:
  #  df[feature] = scipy.stats.boxcox(df[feature].values)
```

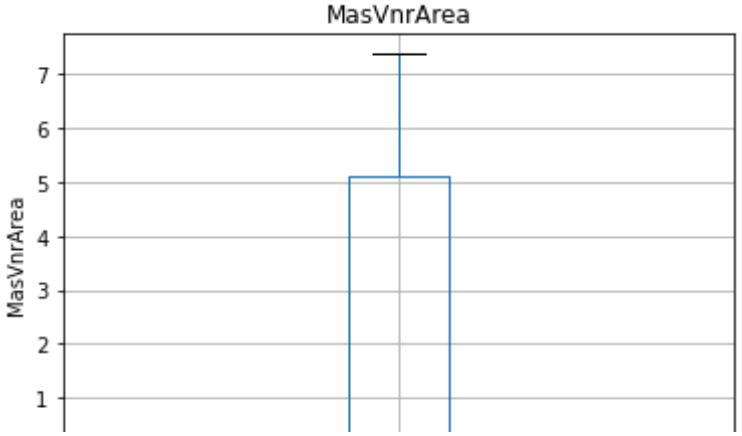
```
---------------------------------------------------------------------
----
ValueError                                Traceback (most recent call l
ast)
<ipython-input-588-b6fa7f7530c5> in <module>
      1 import scipy
      2 for feature in continuous_feature:
----> 3     df[feature] = scipy.stats.boxcox(df[feature].values)

~\Anaconda3\lib\site-packages\pandas\core\frame.py in __setitem__(self,
 key, value)
   3035         else:
   3036             # set column
-> 3037             self._set_item(key, value)
   3038
   3039     def _setitem_slice(self, key: slice, value):

~\Anaconda3\lib\site-packages\pandas\core\frame.py in _set_item(self, k
ey, value)
   3111         """
   3112         self._ensure_valid_index(value)
-> 3113         value = self._sanitize_column(key, value)
   3114         NDFrame._set_item(self, key, value)
   3115

~\Anaconda3\lib\site-packages\pandas\core\frame.py in _sanitize_column
(self, key, value, broadcast)
```

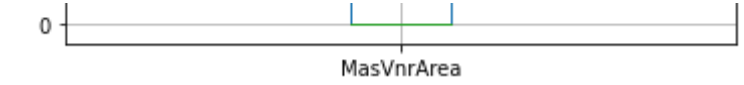
```
   3756
   3757             # turn me into an ndarray
-> 3758             value = sanitize_index(value, self.index)
   3759             if not isinstance(value, (np.ndarray, Index)):
   3760                 if isinstance(value, list) and len(value) > 0:

~\Anaconda3\lib\site-packages\pandas\core\internals\construction.py in
sanitize_index(data, index)
    746     if len(data) != len(index):
    747         raise ValueError(
--> 748             "Length of values "
    749             f"({len(data)}) "
    750             "does not match length of index "

ValueError: Length of values (2) does not match length of index (1460)
```

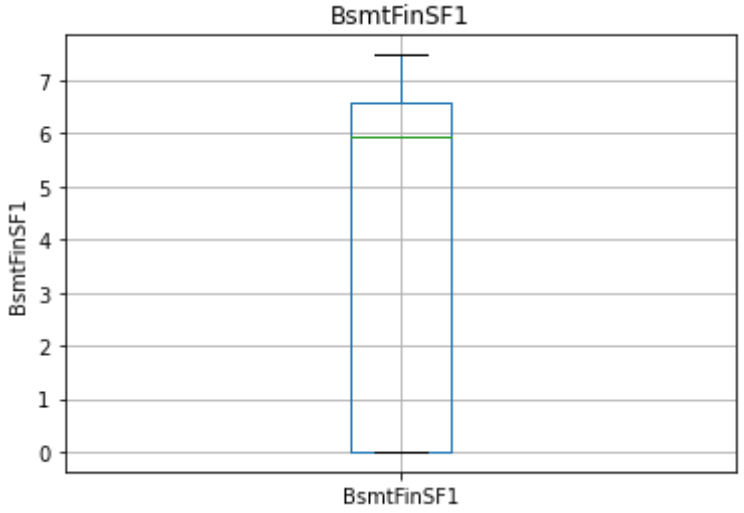
In [31]:
```
for feature in continuous_feature:
        df.boxplot(column=feature)
        plt.ylabel(feature)
        plt.title(feature)
        plt.show()
```

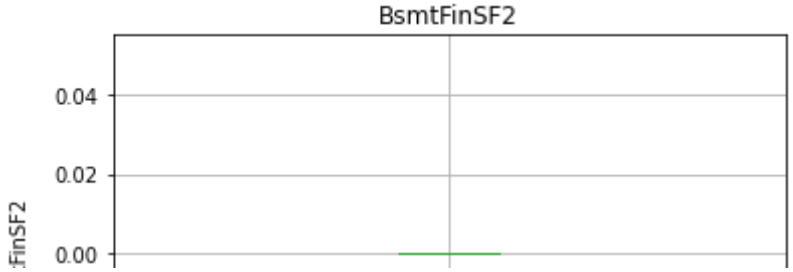


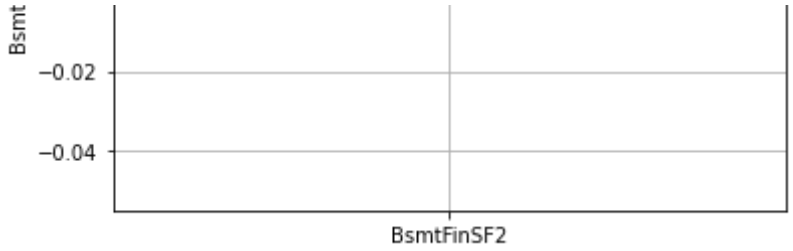


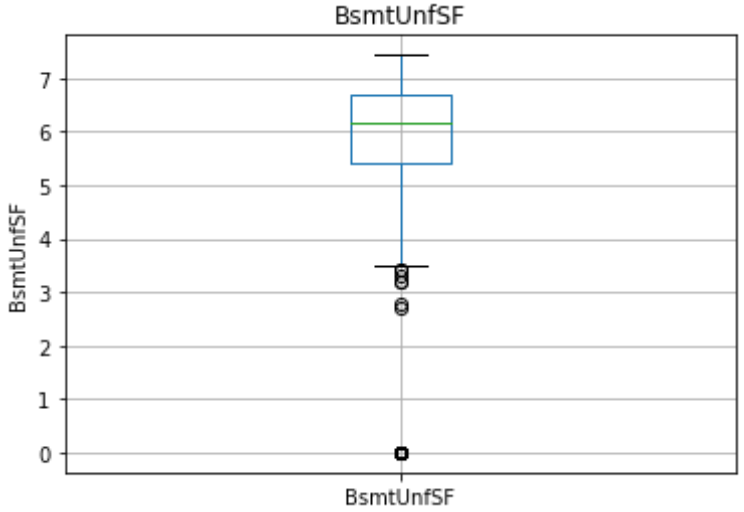


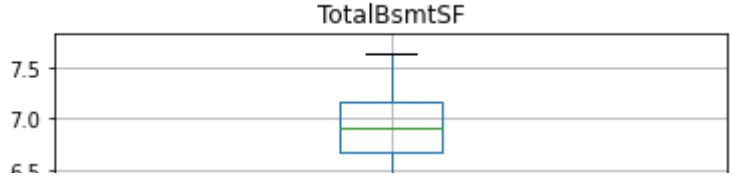


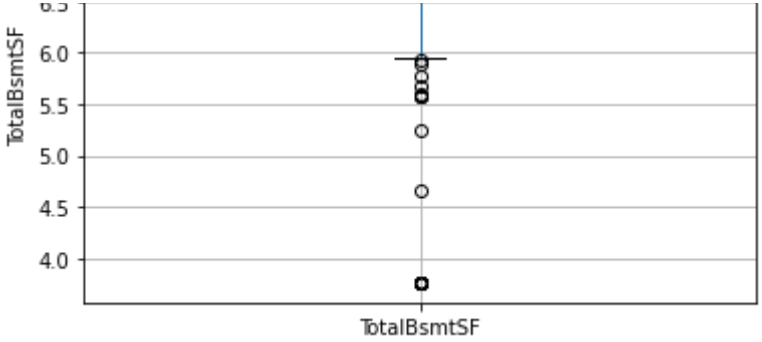


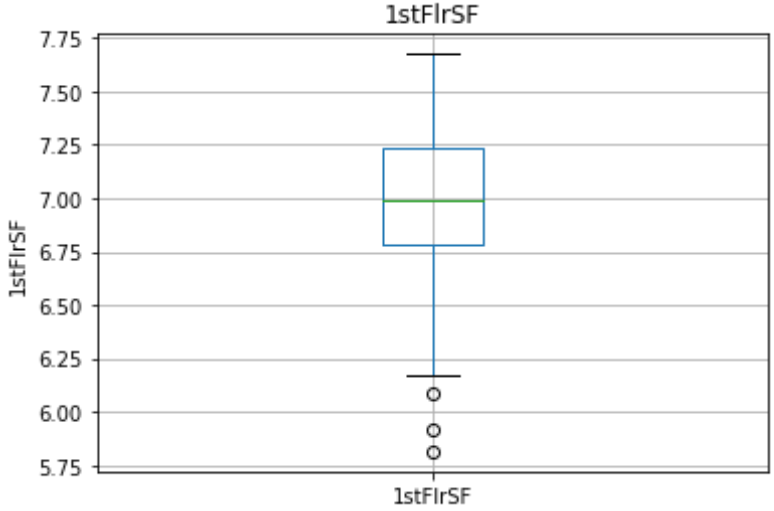


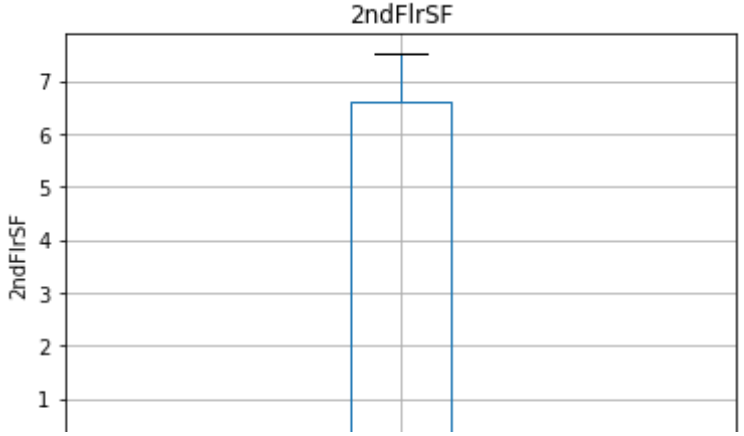


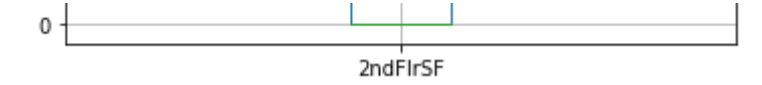


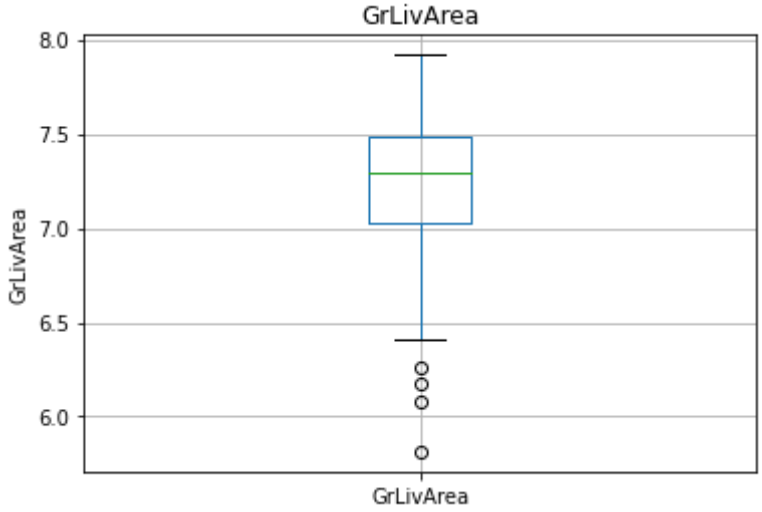


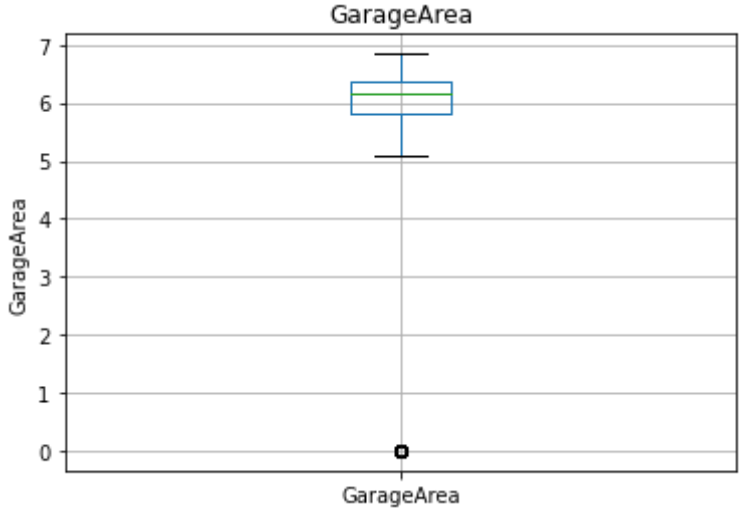


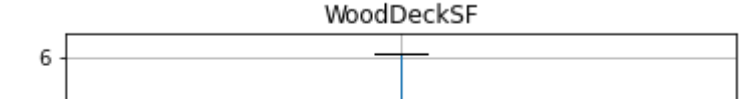


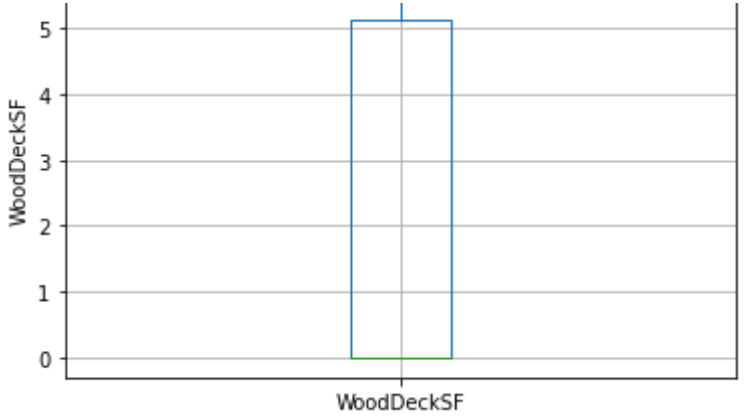


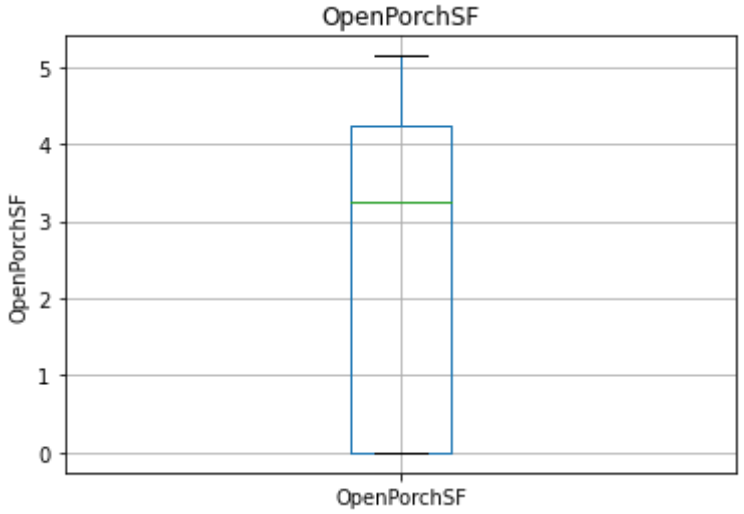


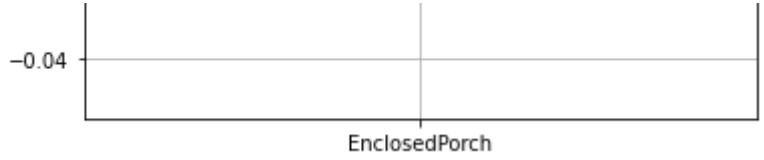


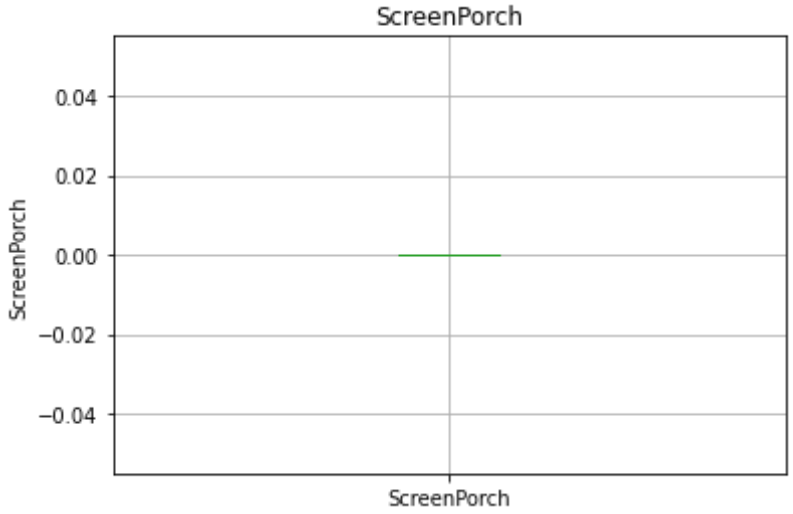


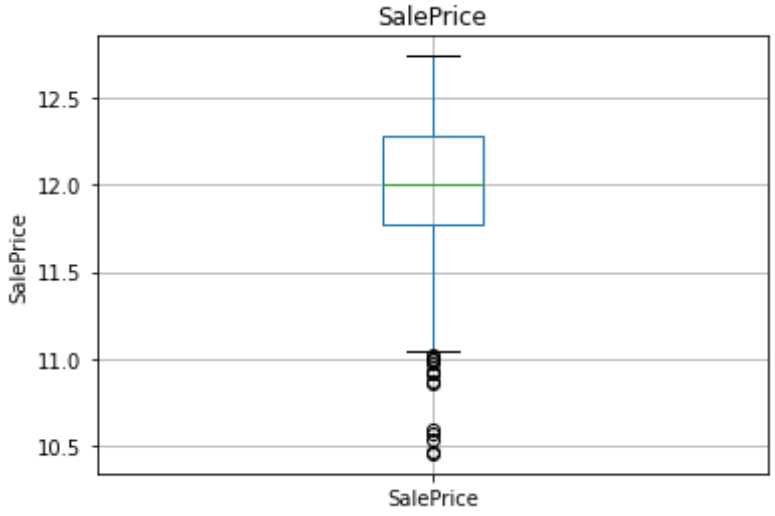






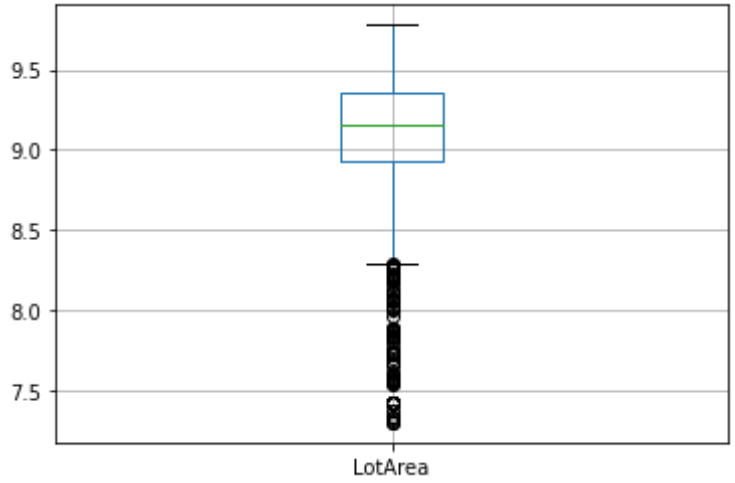






In [437]:
```
data.boxplot(column='LotArea')
#sns.boxplot(df['LotArea'])
```

Out[437]: <AxesSubplot:>



In [32]:
```
continuous_feature
```

Out[32]:
```
['LotFrontage',
 'LotArea',
```

```
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch',
 'SalePrice']
```

Categorical Variables

In [33]:
```
categorical_features= [feature for feature in df.columns if df[feature]
.dtypes=='O']
categorical_features
```

Out[33]:
```
['MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
```

```
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']
```

In [34]:
```
df[categorical_features].head()
```

Out[34]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhoc
0	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	CollgC
1	RL	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Veenk
2	RL	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	CollgC
3	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Crawf
4	RL	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	NoRidg

In [35]:
```
for feature in categorical_features:
    print('The feature is {} and number of categories are {}'.format(fe
ature,len(df[feature].unique())))
```

```
The feature is MSZoning and number of categories are 5
The feature is Street and number of categories are 2
The feature is Alley and number of categories are 3
The feature is LotShape and number of categories are 4
The feature is LandContour and number of categories are 4
The feature is Utilities and number of categories are 2
The feature is LotConfig and number of categories are 5
The feature is LandSlope and number of categories are 3
The feature is Neighborhood and number of categories are 25
The feature is Condition1 and number of categories are 9
The feature is Condition2 and number of categories are 8
The feature is BldgType and number of categories are 5
The feature is HouseStyle and number of categories are 8
The feature is RoofStyle and number of categories are 6
The feature is RoofMatl and number of categories are 8
The feature is Exterior1st and number of categories are 15
The feature is Exterior2nd and number of categories are 16
The feature is MasVnrType and number of categories are 5
The feature is ExterQual and number of categories are 4
The feature is ExterCond and number of categories are 5
The feature is Foundation and number of categories are 6
The feature is BsmtQual and number of categories are 5
The feature is BsmtCond and number of categories are 5
The feature is BsmtExposure and number of categories are 5
The feature is BsmtFinType1 and number of categories are 7
The feature is BsmtFinType2 and number of categories are 7
The feature is Heating and number of categories are 6
The feature is HeatingQC and number of categories are 5
The feature is CentralAir and number of categories are 2
The feature is Electrical and number of categories are 6
The feature is KitchenQual and number of categories are 4
The feature is Functional and number of categories are 7
The feature is FireplaceQu and number of categories are 6
The feature is GarageType and number of categories are 7
```

```
The feature is GarageFinish and number of categories are 4
The feature is GarageQual and number of categories are 6
The feature is GarageCond and number of categories are 6
The feature is PavedDrive and number of categories are 3
The feature is PoolQC and number of categories are 4
The feature is Fence and number of categories are 5
The feature is MiscFeature and number of categories are 5
The feature is SaleType and number of categories are 9
The feature is SaleCondition and number of categories are 6
```

In [108]:
```
l1 = list()
for f in df['Neighborhood'].unique():

    l1.append(df.Neighborhood[df['Neighborhood']==f].value_counts()[0])
    print(df.groupby(f).count())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-108-33e1a564f248> in <module>
      3
      4     l1.append(df.Neighborhood[df['Neighborhood']==f].value_counts()[0])
----> 5     print(df.groupby(f).count())

~\Anaconda3\lib\site-packages\pandas\core\frame.py in groupby(self, by, axis, level, as_index, sort, group_keys, squeeze, observed, dropna)
   6512             squeeze=squeeze,
   6513             observed=observed,
-> 6514             dropna=dropna,
   6515         )
   6516

~\Anaconda3\lib\site-packages\pandas\core\groupby\groupby.py in __init__(self, obj, keys, axis, level, grouper, exclusions, selection, as_index, sort, group_keys, squeeze, observed, mutated, dropna)
    531                 observed=observed,
    532                 mutated=self.mutated,
```

```
--> 533                 dropna=self.dropna,
    534             )
    535

~\Anaconda3\lib\site-packages\pandas\core\groupby\grouper.py in get_gro
uper(obj, key, axis, level, sort, observed, mutated, validate, dropna)
    775                 in_axis, name, level, gpr = False, None, gpr, N
one
    776             else:
--> 777                 raise KeyError(gpr)
    778         elif isinstance(gpr, Grouper) and gpr.key is not None:
    779             # Add key to exclusions

KeyError: 'CollgCr'
```

In [131]:
```
l1 = dict()
l2=list()
for f in df['Neighborhood'].unique():
    #print(df.groupby(f)['SalePrice'].count())
    l1[df.Neighborhood[df['Neighborhood']==f].value_counts().index[0]]
= df.Neighborhood[df['Neighborhood']==f].value_counts()[0]
```

In [129]:
```
df.Neighborhood[df['Neighborhood']==f].value_counts()[0]
```

Out[129]:
```
150
```

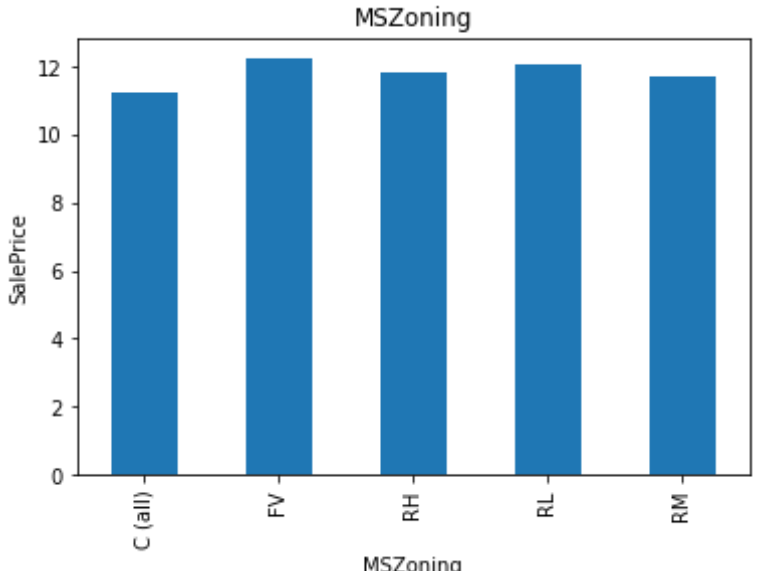
In [132]:
```
l1
```

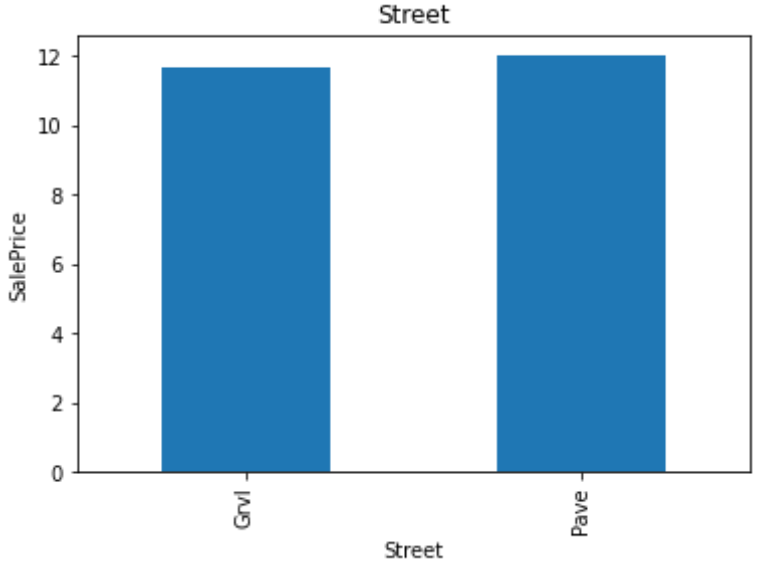
Out[132]:
```
{'CollgCr': 150,
 'Veenker': 11,
 'Crawfor': 51,
 'NoRidge': 41,
 'Mitchel': 49,
 'Somerst': 86,
 'NWAmes': 73,
 'OldTown': 113,
```

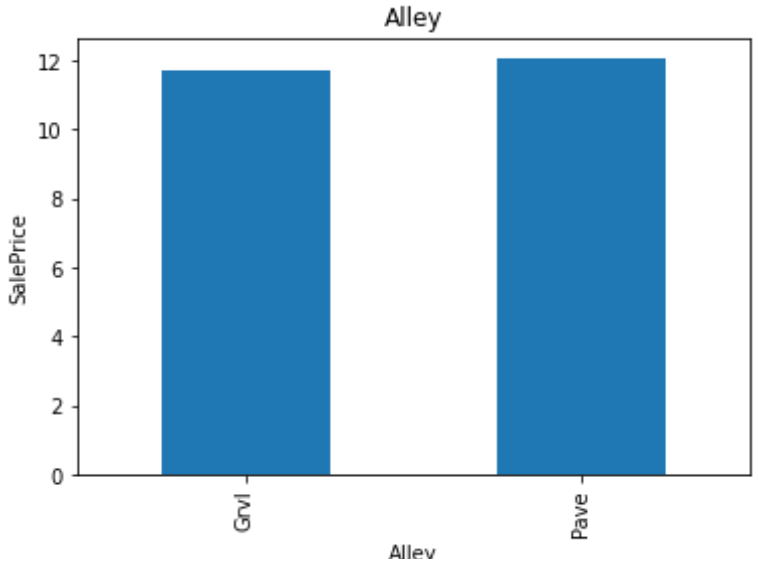
```
 'BrkSide': 58,
 'Sawyer': 74,
 'NridgHt': 77,
 'NAmes': 225,
 'SawyerW': 59,
 'IDOTRR': 37,
 'MeadowV': 17,
 'Edwards': 100,
 'Timber': 38,
 'Gilbert': 79,
 'StoneBr': 25,
 'ClearCr': 28,
 'NPkVill': 9,
 'Blmngtn': 17,
 'BrDale': 16,
 'SWISU': 25,
 'Blueste': 2}
```

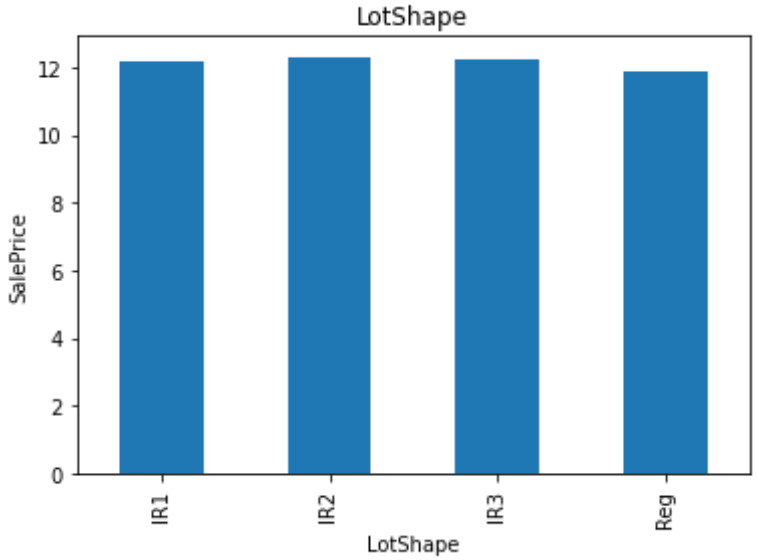
In [595]:
```
## Finding the relationship b/w Categorical features and the SalePrice
```

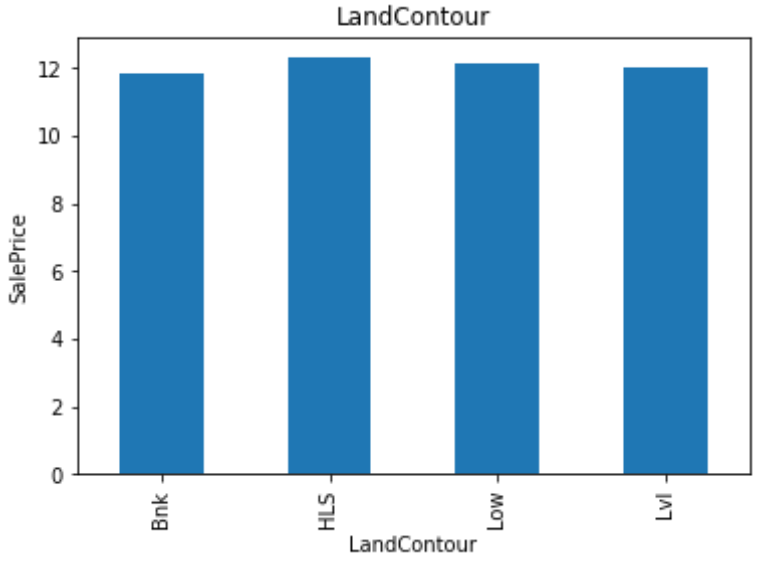
In [36]:
```
for feature in categorical_features:
    data=df.copy()
    data.groupby(feature)['SalePrice'].median().plot.bar()
    plt.xlabel(feature)
    plt.ylabel('SalePrice')
    plt.title(feature)
    plt.show()
```

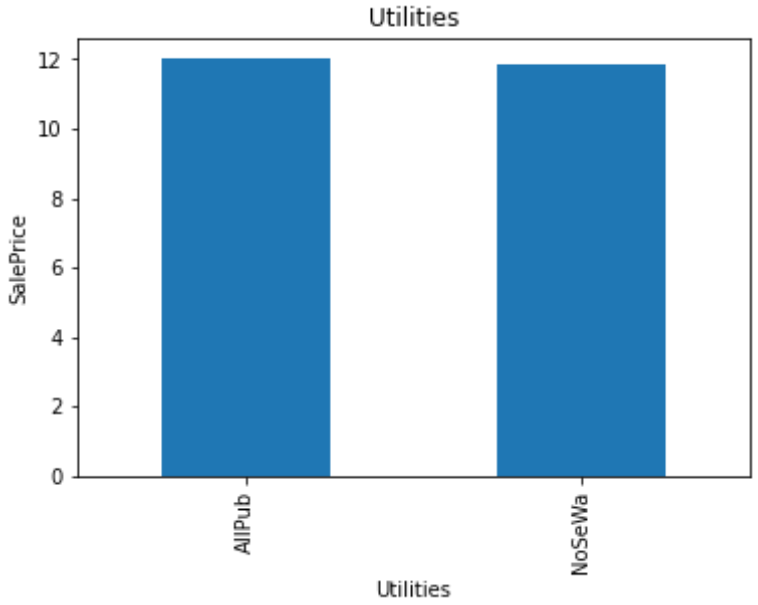


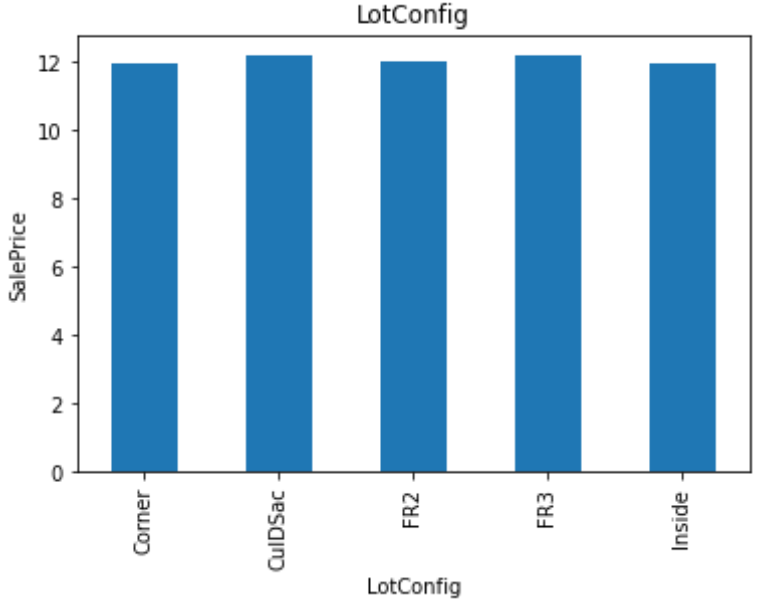


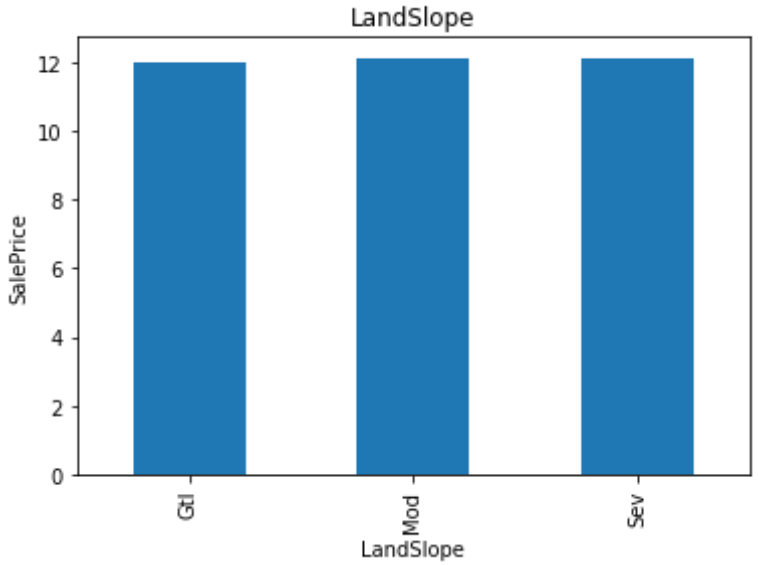


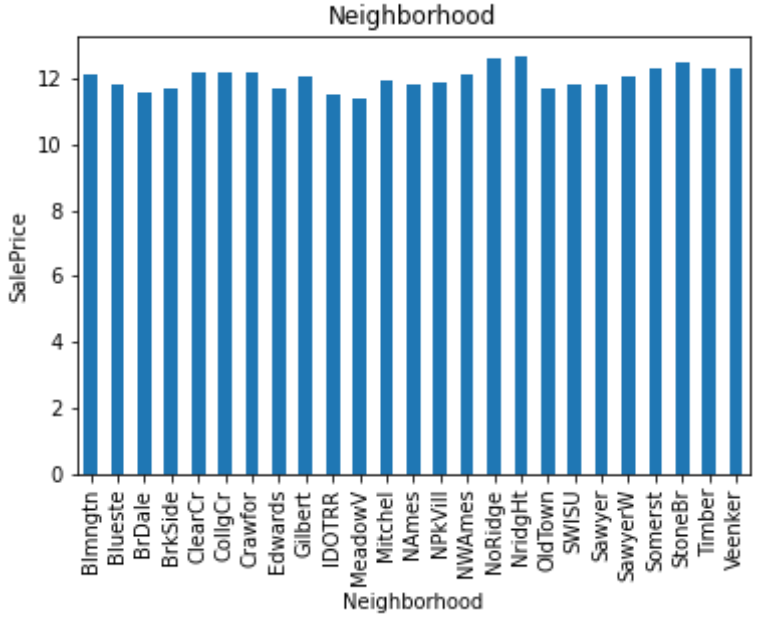


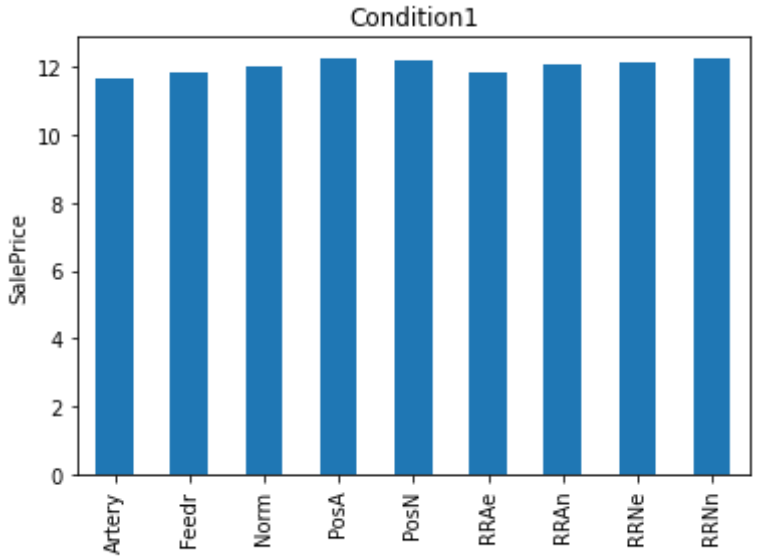


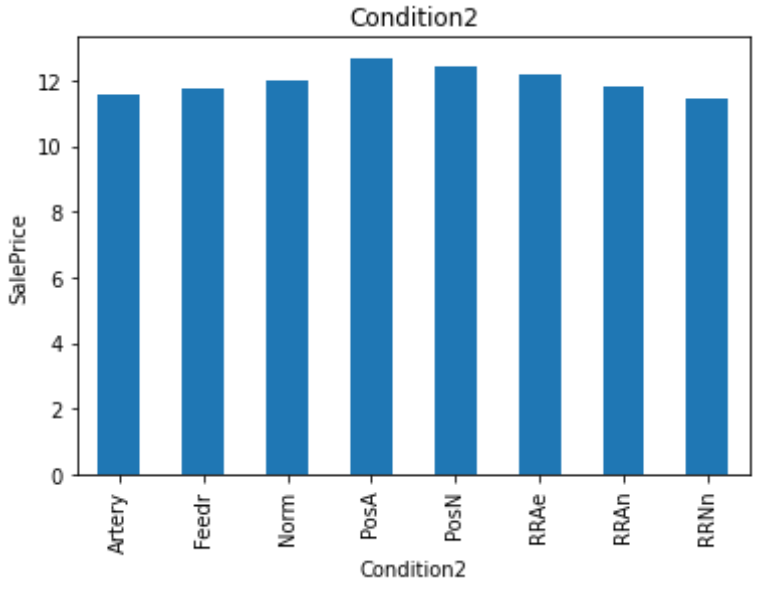


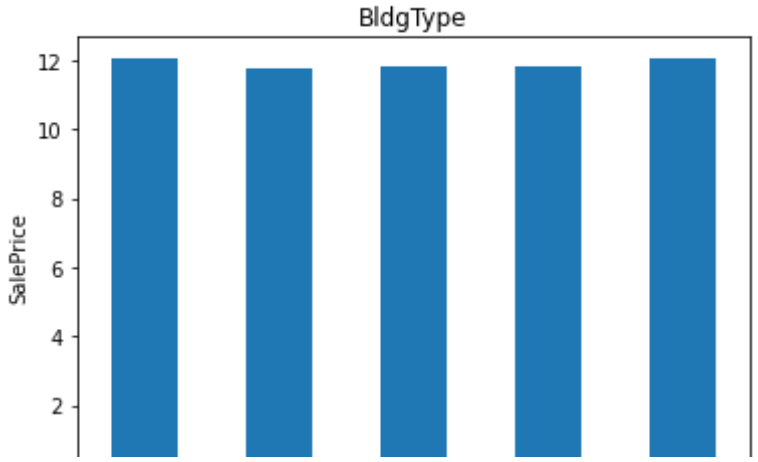


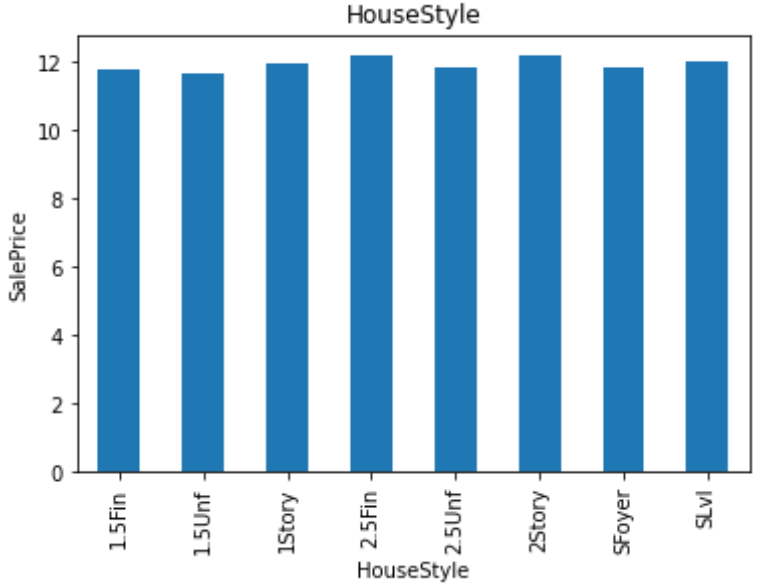


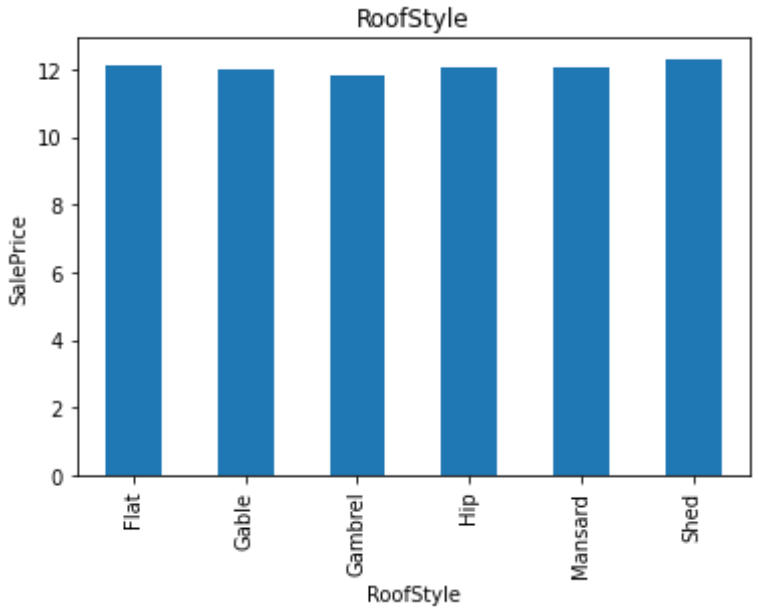


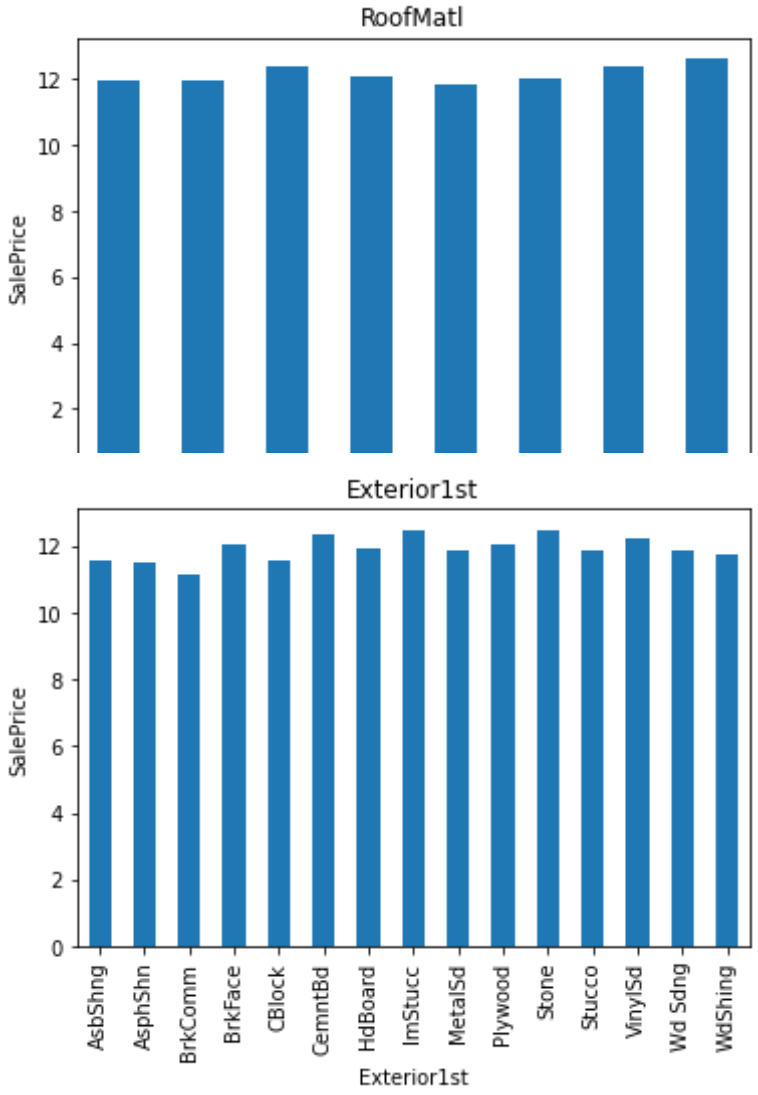


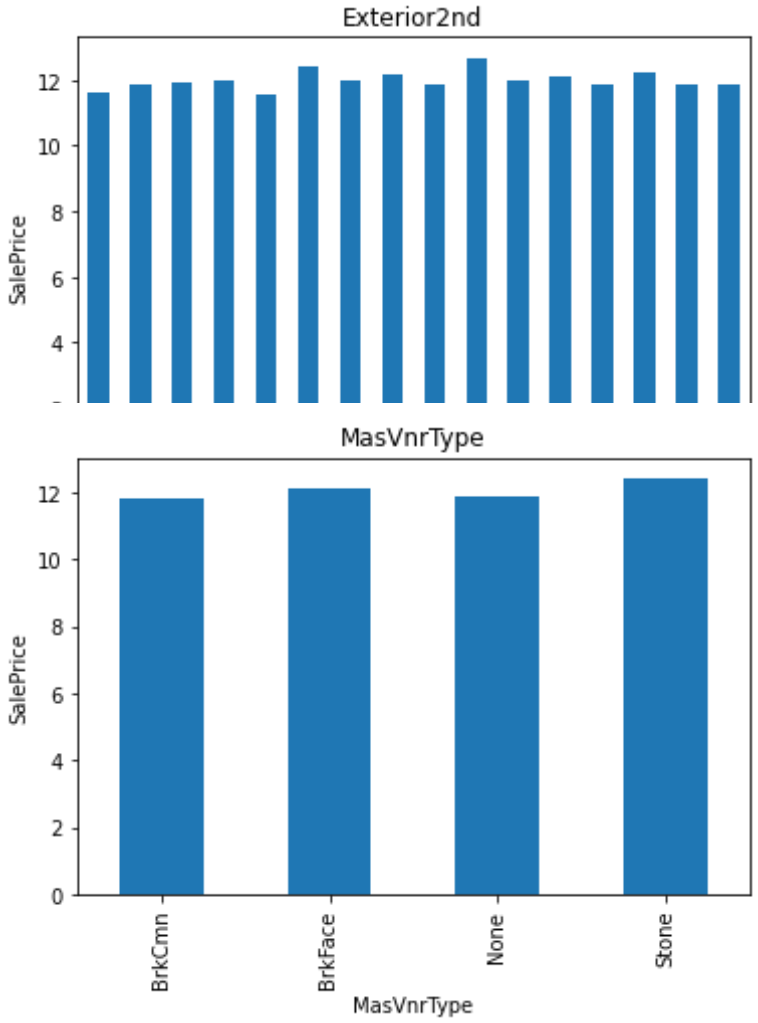


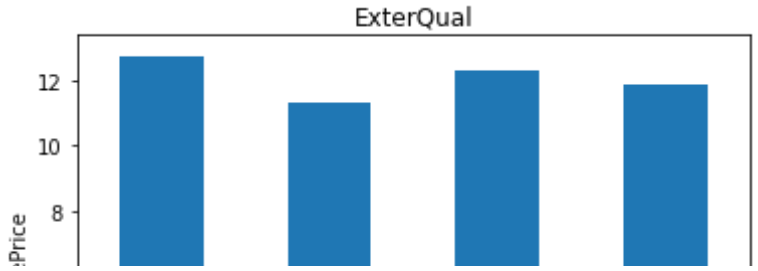


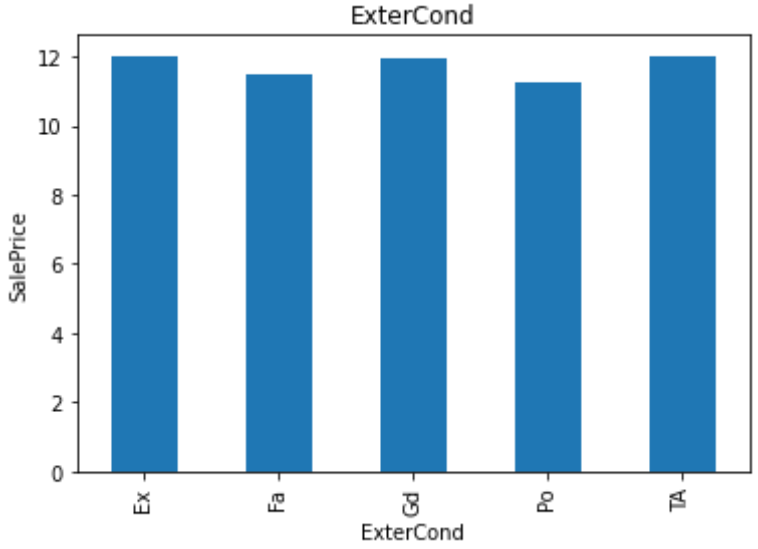


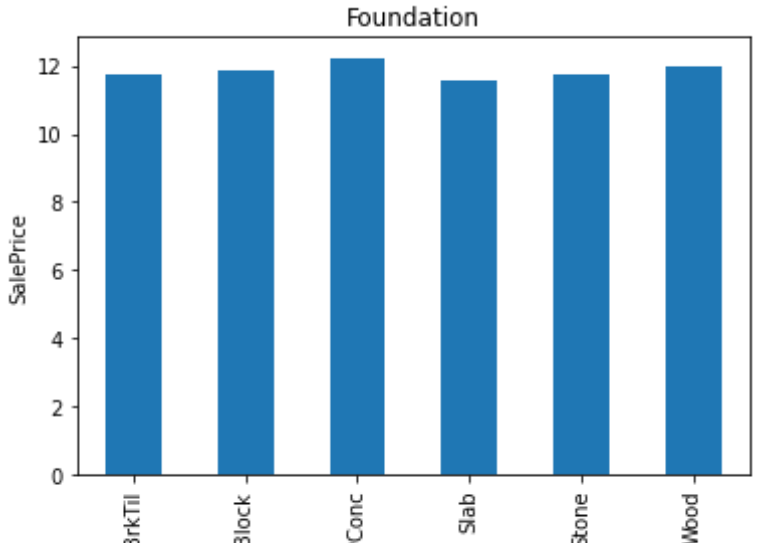


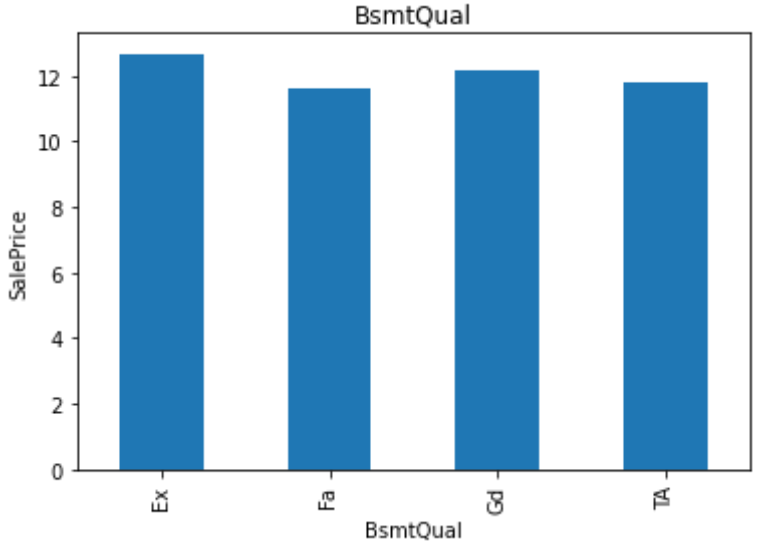


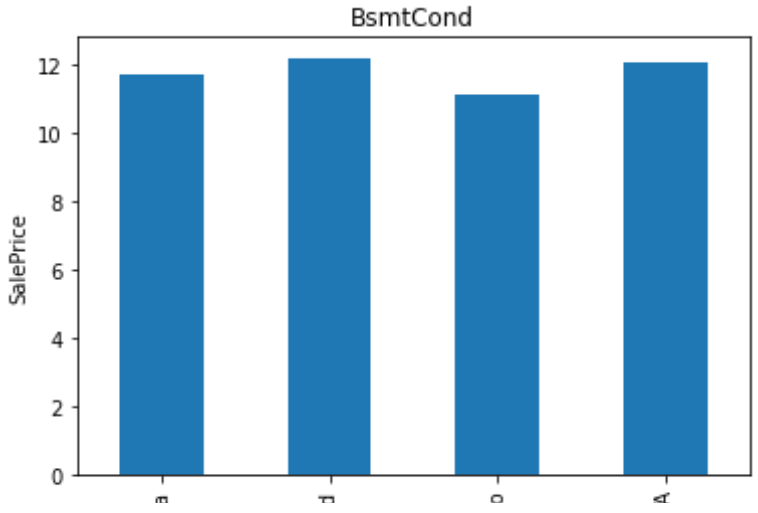


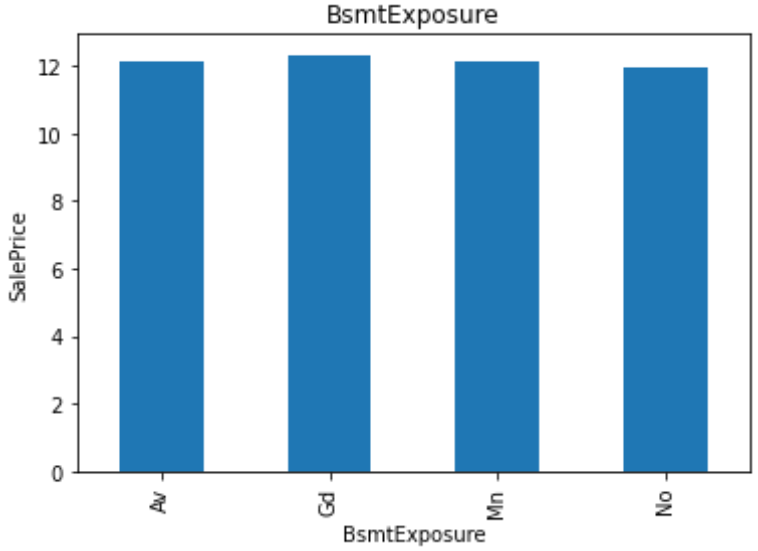


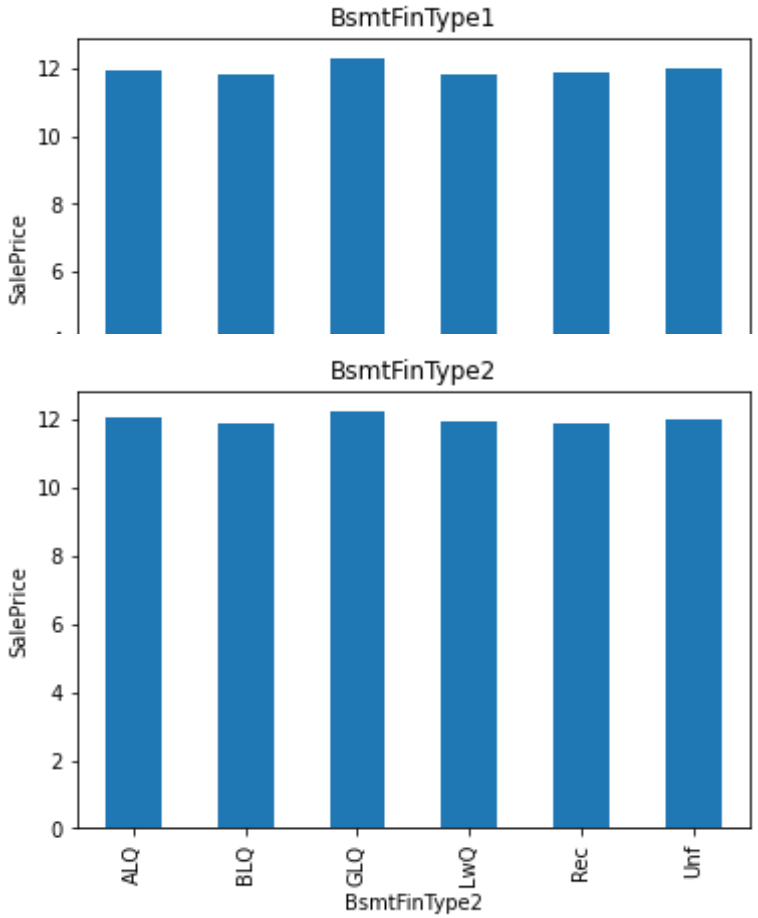


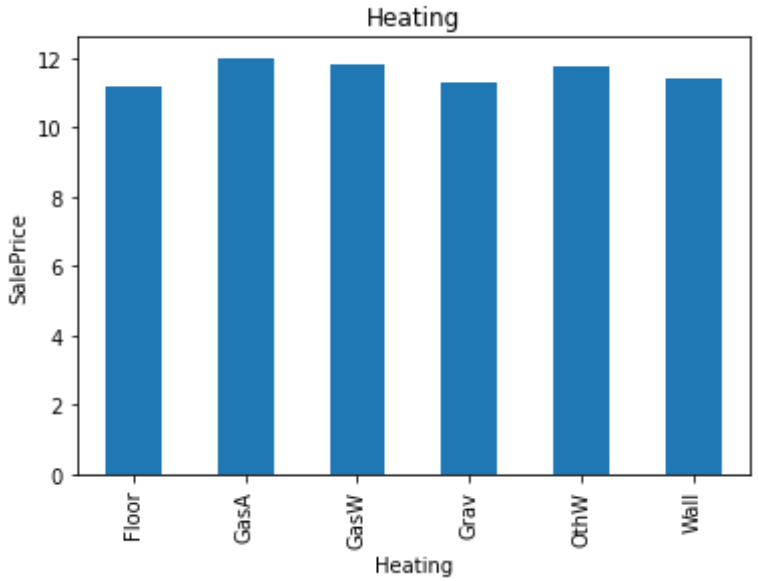


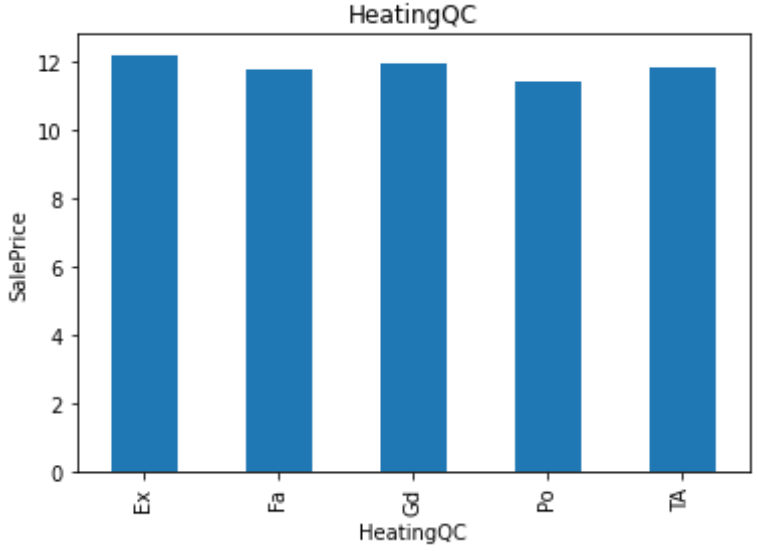


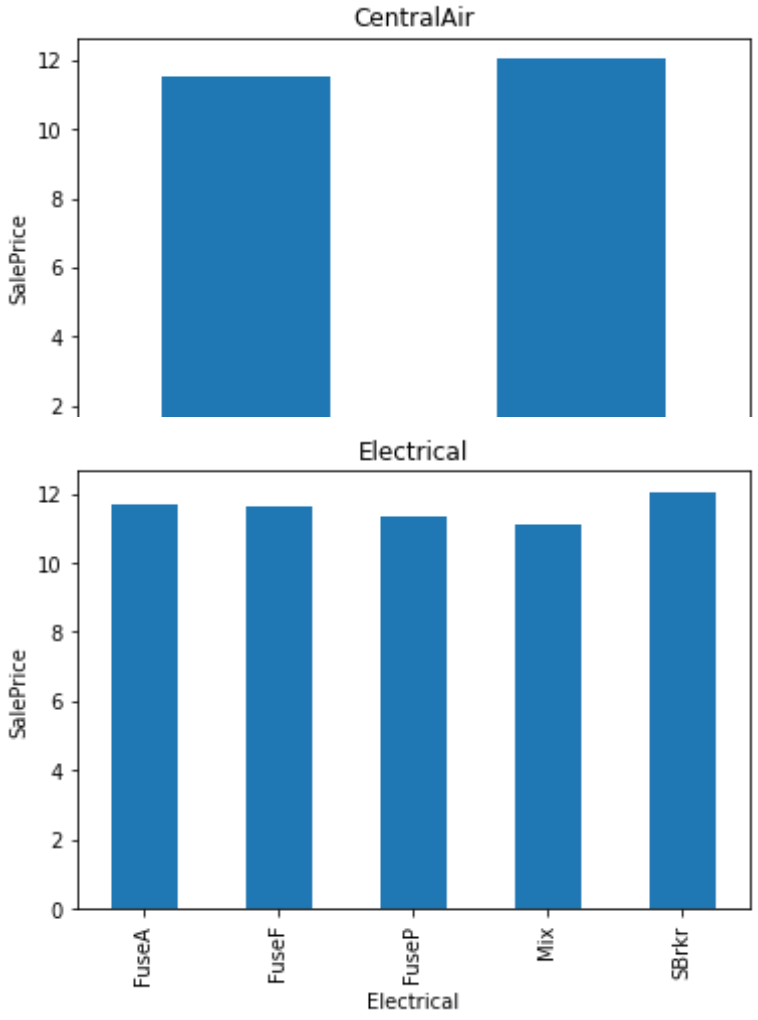


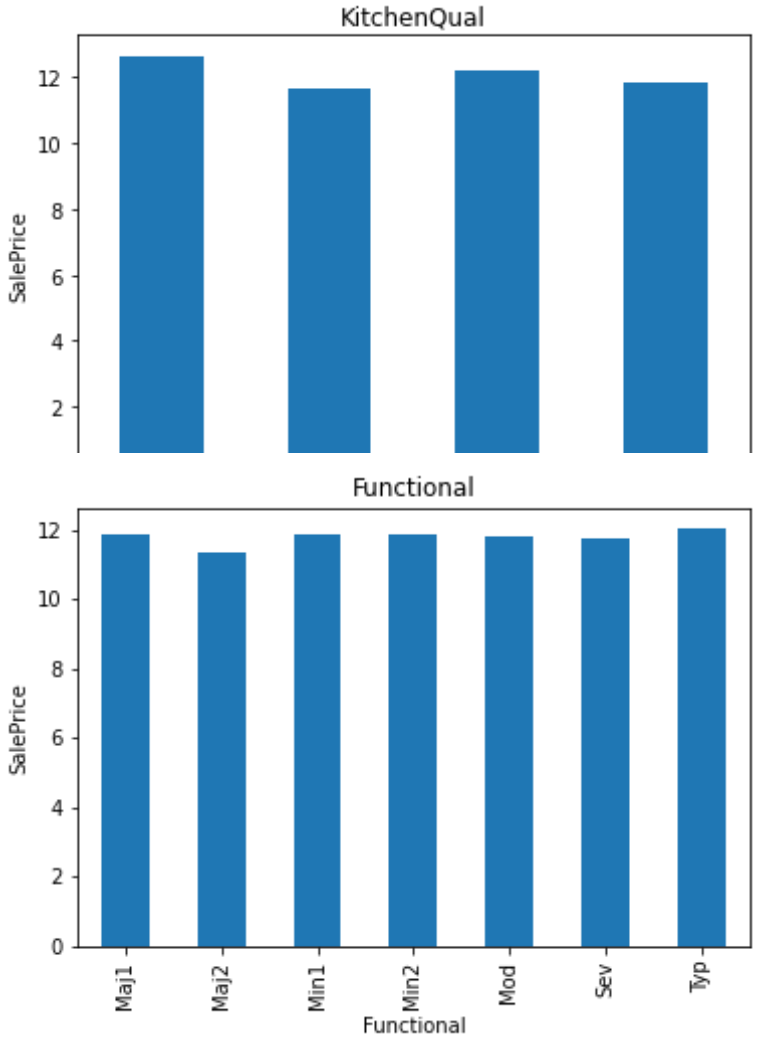


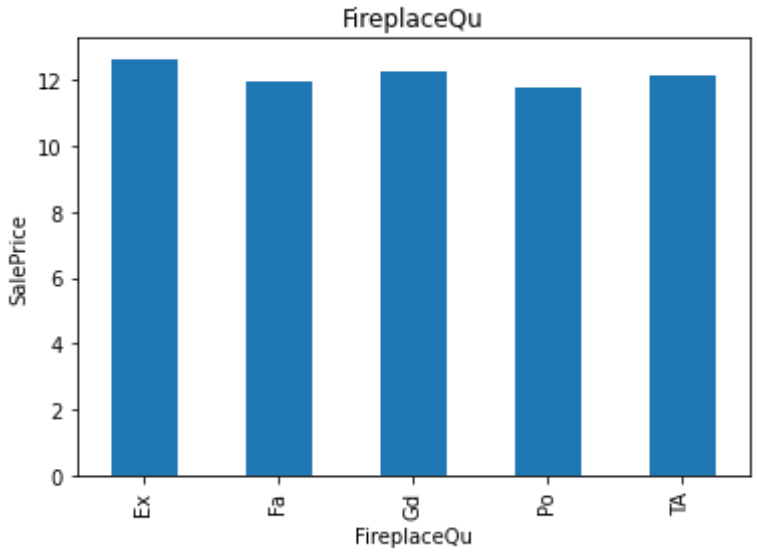


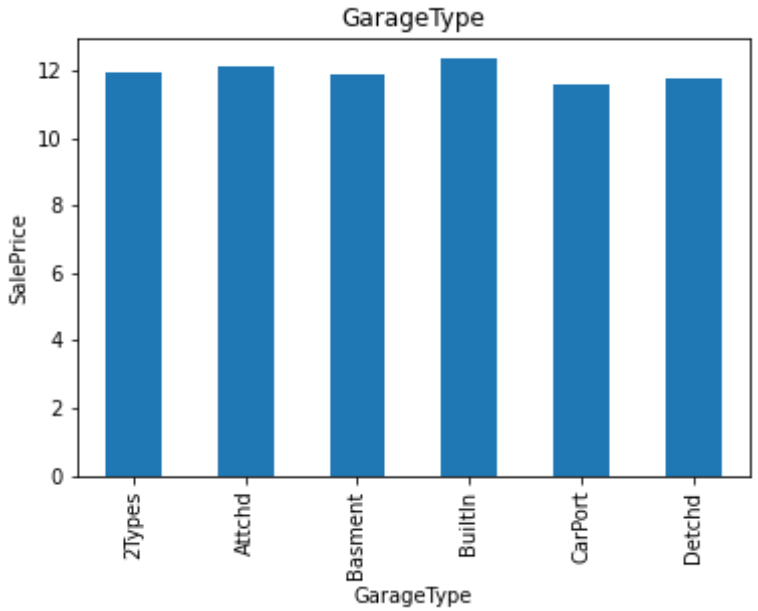


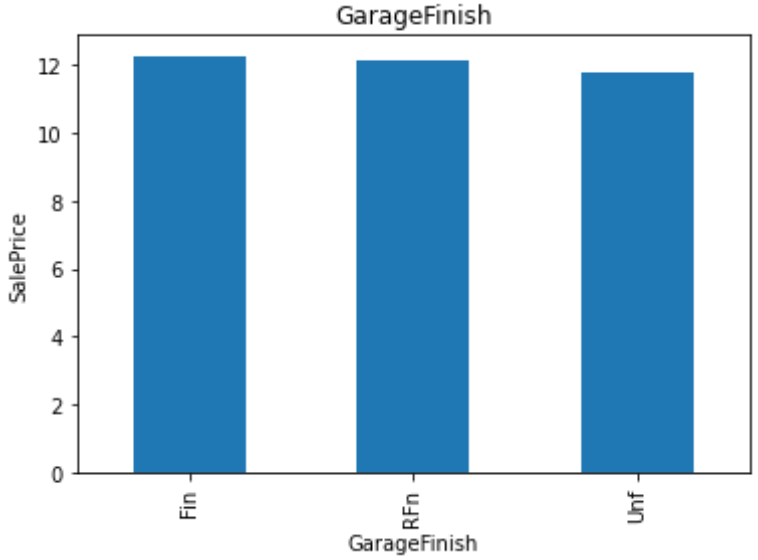


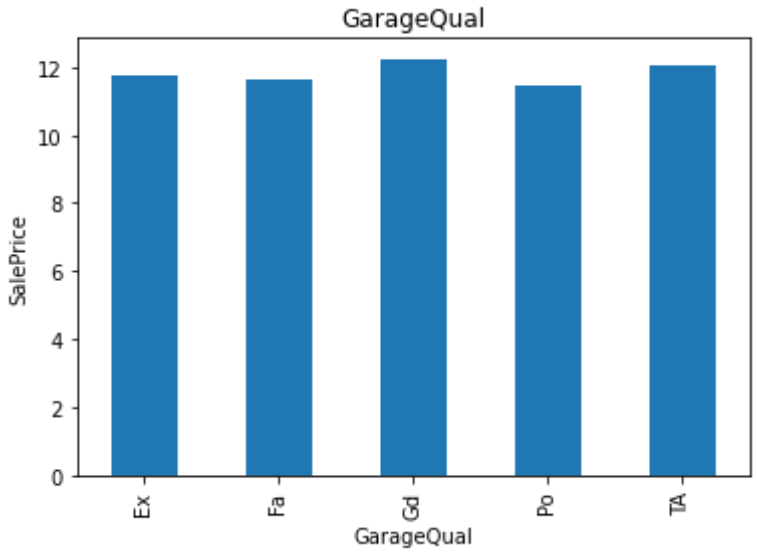




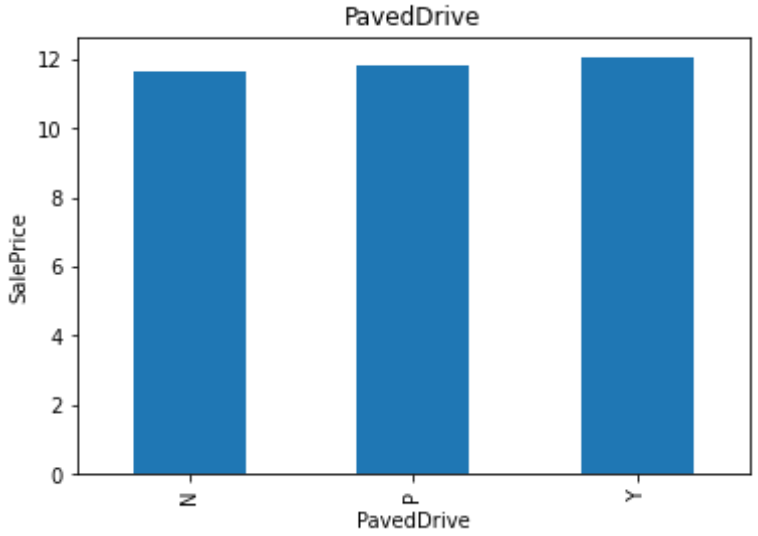


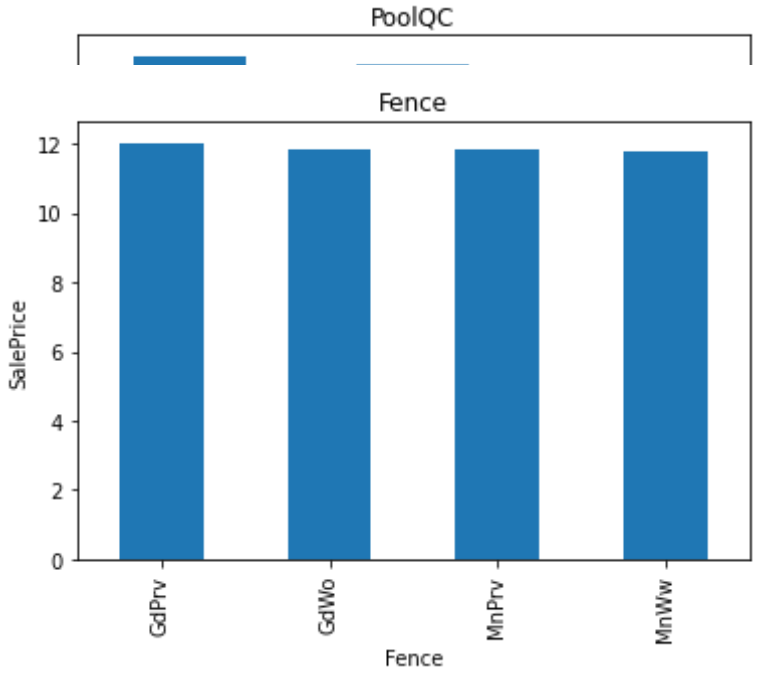


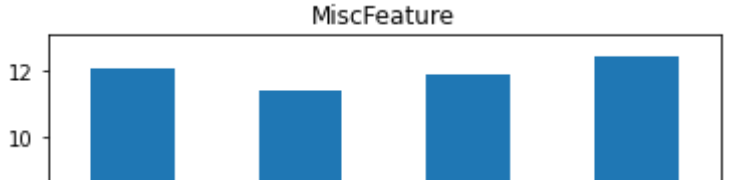


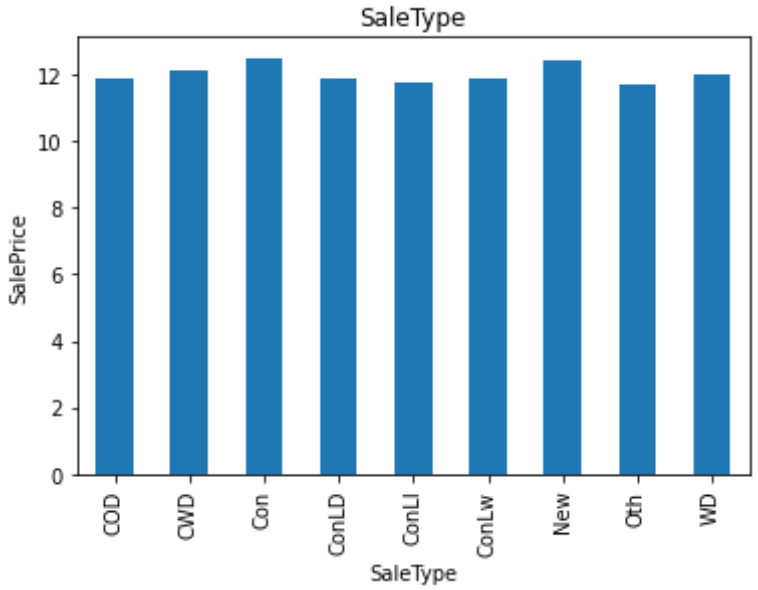


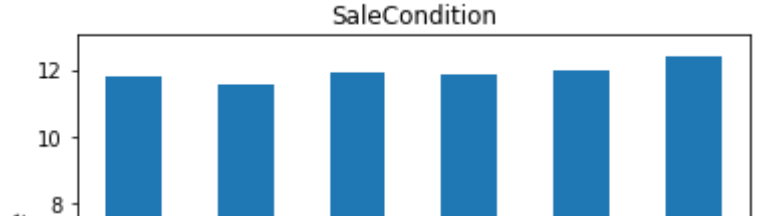
GarageCond

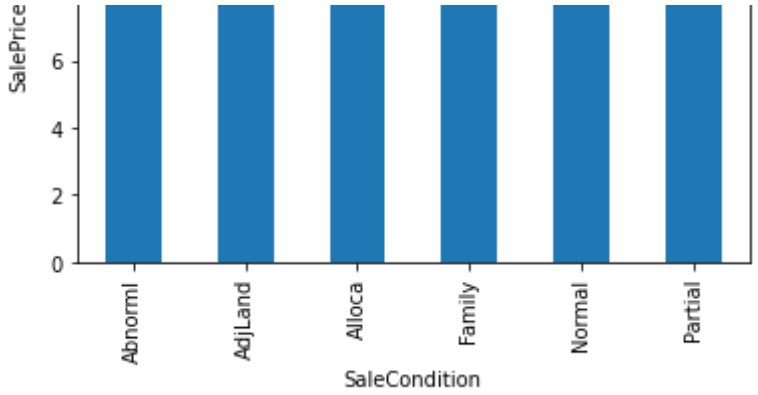












Feature Engineering

We will be performing all the below steps in Feature Engineering: 1.Handling missing values 2.Handling Temporal variables 3.Handling Categorical variables: remove rare labels 4.Standardize the values of the variables to the same range

Missing Values

In [37]:
```
## Let us capture all the nan values
## First lets handle Categorical features which are missing

features_nan= [feature for feature in df.columns if df[feature].dtypes=
='O' and df[feature].isnull().sum()>=1]

for feature in features_nan:
    print('{}: {} missing values'.format(feature,np.round(df[feature].i
snull().mean(),4)))
```

```
Alley: 0.9377 missing values
MasVnrType: 0.0055 missing values
BsmtQual: 0.0253 missing values
BsmtCond: 0.0253 missing values
BsmtExposure: 0.026 missing values
BsmtFinType1: 0.0253 missing values
BsmtFinType2: 0.026 missing values
Electrical: 0.0007 missing values
FireplaceQu: 0.4726 missing values
GarageType: 0.0555 missing values
GarageFinish: 0.0555 missing values
GarageQual: 0.0555 missing values
GarageCond: 0.0555 missing values
PoolQC: 0.9952 missing values
```

```
Fence: 0.8075 missing values
MiscFeature: 0.963 missing values
```

In [38]:
```
## Replace missing value with a new label
def replace_cat_feature(df,features_nan):
    data=df.copy()

    data[features_nan] = np.where(data[features_nan].isnull(),'Missing'
,data[features_nan])
    #data[features_nan]=data[features_nan].fillna('Missing')
    return data
df=replace_cat_feature(df,features_nan)
```

In [39]:
```
df[features_nan].isnull().sum()
```

Out[39]:
```
Alley           0
MasVnrType      0
BsmtQual        0
BsmtCond        0
BsmtExposure    0
BsmtFinType1    0
BsmtFinType2    0
Electrical      0
FireplaceQu     0
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
PoolQC          0
Fence           0
MiscFeature     0
dtype: int64
```

In [40]:
```
df.head(20)
```

Out[40]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	4.189655	9.042040	Pave	Missing	Reg	Lvl
1	2	20	RL	4.394449	9.169623	Pave	Missing	Reg	Lvl
2	3	60	RL	4.234107	9.328212	Pave	Missing	IR1	Lvl
3	4	70	RL	4.110874	9.164401	Pave	Missing	IR1	Lvl
4	5	60	RL	4.442651	9.565284	Pave	Missing	IR1	Lvl
5	6	50	RL	4.454347	9.555064	Pave	Missing	IR1	Lvl
6	7	20	RL	4.330733	9.218804	Pave	Missing	Reg	Lvl
7	8	60	RL	NaN	9.247925	Pave	Missing	IR1	Lvl
8	9	50	RM	3.951244	8.719481	Pave	Missing	Reg	Lvl
9	10	190	RL	3.931826	8.912069	Pave	Missing	Reg	Lvl
10	11	20	RL	4.262680	9.323758	Pave	Missing	Reg	Lvl
11	12	60	RL	4.454347	9.386392	Pave	Missing	IR1	Lvl
12	13	20	RL	NaN	9.470317	Pave	Missing	IR2	Lvl
13	14	20	RL	4.521789	9.273597	Pave	Missing	IR1	Lvl
14	15	20	RL	NaN	9.298443	Pave	Missing	IR1	Lvl
15	16	45	RM	3.951244	8.719481	Pave	Missing	Reg	Lvl
16	17	20	RL	NaN	9.327412	Pave	Missing	IR1	Lvl
17	18	90	RL	4.290459	9.286560	Pave	Missing	Reg	Lvl
18	19	20	RL	4.204693	9.524859	Pave	Missing	Reg	Lvl
19	20	20	RL	4.262680	8.930759	Pave	Missing	Reg	Lvl

In [41]:
```
## Now lets check for numerical variables that contains missing values

numerical_with_nan=[feature for feature in df.columns if df[feature].is
null().sum()>=1 and df[feature].dtypes!='O']
```

```
## We will print the numerical nan variables and percentage of missing
 values

for feature in numerical_with_nan:
    print("{}: {}% missing value".format(feature,np.round(df[feature].i
snull().mean()*100,4)))
```

```
LotFrontage: 17.7397% missing value
MasVnrArea: 0.5479% missing value
GarageYrBlt: 5.5479% missing value
```

In [42]:
```
#Replacing the numerical missing values
for feature in numerical_with_nan:
    ## We will replace by using median since there are outliers

    ## create a new feature to capture nan values
    df[feature+'nan'] = np.where(df[feature].isnull(),1,0)

    ## ## We will replace by using median since there are outliers
    df[feature].fillna(df[feature].median(),inplace=True)

df[numerical_with_nan].isnull().sum()
```

Out[42]:
```
LotFrontage    0
MasVnrArea     0
GarageYrBlt    0
dtype: int64
```

In [43]:
```
df.head(10)
```

Out[43]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	I
0	1	60	RL	4.189655	9.042040	Pave	Missing	Reg	Lvl	
1	2	20	RL	4.394449	9.169623	Pave	Missing	Reg	Lvl	
2	3	60	RL	4.234107	9.328212	Pave	Missing	IR1	Lvl	
3	4	70	RL	4.110874	9.164401	Pave	Missing	IR1	Lvl	

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
4	5	60	RL	4.442651	9.565284	Pave	Missing	IR1	Lvl	
5	6	50	RL	4.454347	9.555064	Pave	Missing	IR1	Lvl	
6	7	20	RL	4.330733	9.218804	Pave	Missing	Reg	Lvl	
7	8	60	RL	4.248495	9.247925	Pave	Missing	IR1	Lvl	
8	9	50	RM	3.951244	8.719481	Pave	Missing	Reg	Lvl	
9	10	190	RL	3.931826	8.912069	Pave	Missing	Reg	Lvl	

In [44]:
```
## Temporal Variables (Date Time Variables)

for feature in ['YearBuilt','YearRemodAdd','GarageYrBlt']:

    df[feature]=df['YrSold']-df[feature]
```

In [45]:
```
df.head()
```

Out[45]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
0	1	60	RL	4.189655	9.042040	Pave	Missing	Reg	Lvl	
1	2	20	RL	4.394449	9.169623	Pave	Missing	Reg	Lvl	
2	3	60	RL	4.234107	9.328212	Pave	Missing	IR1	Lvl	
3	4	70	RL	4.110874	9.164401	Pave	Missing	IR1	Lvl	
4	5	60	RL	4.442651	9.565284	Pave	Missing	IR1	Lvl	

In [46]:
```
df[['YearBuilt','YearRemodAdd','GarageYrBlt']].head()
```

Out[46]:

	YearBuilt	YearRemodAdd	GarageYrBlt
0	5	5	5.0

	YearBuilt	YearRemodAdd	GarageYrBlt
1	31	31	31.0
2	7	6	7.0
3	91	36	8.0
4	8	8	8.0

Numerical Variables

Since the numerical variables are skewed, we will perform log normal distribution

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

Out[47]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
0	1	60	RL	4.189655	9.042040	Pave	Missing	Reg	Lvl	
1	2	20	RL	4.394449	9.169623	Pave	Missing	Reg	Lvl	
2	3	60	RL	4.234107	9.328212	Pave	Missing	IR1	Lvl	
3	4	70	RL	4.110874	9.164401	Pave	Missing	IR1	Lvl	
4	5	60	RL	4.442651	9.565284	Pave	Missing	IR1	Lvl	

In [48]:
```
#Considering only non zero value features, as we're taking log
num_features=['LotFrontage', 'LotArea', '1stFlrSF', 'GrLivArea', 'SaleP
rice']

#for feature in num_features:
#    df[feature]=np.log(df[feature])
```

Categorical Features

In [49]:
```
categorical_features=[feature for feature in df.columns if df[feature].
dtype=='O']
```

In [50]:
```
df.head()
```

Out[50]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	L
0	1	60	RL	4.189655	9.042040	Pave	Missing	Reg	Lvl	
1	2	20	RL	4.394449	9.169623	Pave	Missing	Reg	Lvl	
2	3	60	RL	4.234107	9.328212	Pave	Missing	IR1	Lvl	
3	4	70	RL	4.110874	9.164401	Pave	Missing	IR1	Lvl	
4	5	60	RL	4.442651	9.565284	Pave	Missing	IR1	Lvl	

In [51]:
```
df[categorical_features]
```

Out[51]:

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

1460 rows × 43 columns

In [52]:
```
for feature in categorical_features:
    labels_ordered=df.groupby([feature])['SalePrice'].mean().sort_value
s().index
    labels_ordered={k:i for i,k in enumerate(labels_ordered,0)}
    df[feature]=df[feature].map(labels_ordered)
    #df2[feature]=df2[feature].map(labels_ordered)  # For test data as
 SalePrice column is not present
```

In [53]:
```
df.head()
```

Out[53]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
0	1	60	3	4.189655	9.042040	1	2	0	1	
1	2	20	3	4.394449	9.169623	1	2	0	1	
2	3	60	3	4.234107	9.328212	1	2	1	1	
3	4	70	3	4.110874	9.164401	1	2	1	1	
4	5	60	3	4.442651	9.565284	1	2	1	1	

In [54]:
```
len(df.columns)
```

Out[54]: 84

Feature Scaling

In [55]:
```
feature_scale=[feature for feature in df.columns if feature not in ['Sa
lePrice','Id']]
len(feature_scale)
```

Out[55]: 82

In [56]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df[feature_scale])
```

Out[56]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [57]:
```
scaler.transform(df[feature_scale])
```

Out[57]:
```
array([[0.23529412, 0.75      , 0.41326841, ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.75      , 0.49030656, ..., 0.        , 0.
,
        0.        ],
       [0.23529412, 0.75      , 0.42998996, ..., 0.        , 0.
,
        0.        ],
       ...,
       [0.29411765, 0.75      , 0.41892525, ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.75      , 0.42998996, ..., 0.        , 0.
,
        0.        ],
       [0.        , 0.75      , 0.46633838, ..., 0.        , 0.
,
        0.        ]])
```

In [58]:
```
df.head()
```

Out[58]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
0	1	60	3	4.189655	9.042040	1	2	0	1	
1	2	20	3	4.394449	9.169623	1	2	0	1	
2	3	60	3	4.234107	9.328212	1	2	1	1	
3	4	70	3	4.110874	9.164401	1	2	1	1	

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Util
4	5	60	3	4.442651	9.565284	1	2	1	1	

In [59]:
```
#Transform the train and test set, and add on the Id and SalePrice variables
data = pd.concat([df[['Id','SalePrice']].reset_index(drop=True),
                  pd.DataFrame(scaler.transform(df[feature_scale]),
                               columns=feature_scale)],axis=1)
```

In [60]:
```
data.head(20)
```

Out[60]:

	Id	SalePrice	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Land
0	1	12.247699	0.235294	0.75	0.413268	0.702292	1.0	1.0	0.000000	
1	2	12.109016	0.000000	0.75	0.490307	0.753770	1.0	1.0	0.000000	
2	3	12.317171	0.235294	0.75	0.429990	0.817759	1.0	1.0	0.333333	
3	4	11.849405	0.294118	0.75	0.383633	0.751663	1.0	1.0	0.333333	
4	5	12.429220	0.235294	0.75	0.508439	0.913414	1.0	1.0	0.333333	
5	6	11.870607	0.176471	0.75	0.512839	0.909290	1.0	1.0	0.333333	
6	7	12.634606	0.000000	0.75	0.466338	0.773614	1.0	1.0	0.000000	
7	8	12.206078	0.235294	0.75	0.435403	0.785364	1.0	1.0	0.333333	
8	9	11.774528	0.176471	0.25	0.323585	0.572143	1.0	1.0	0.000000	
9	10	11.678448	1.000000	0.75	0.316280	0.649850	1.0	1.0	0.000000	
10	11	11.771444	0.000000	0.75	0.440738	0.815961	1.0	1.0	0.000000	
11	12	12.736814	0.235294	0.75	0.512839	0.841233	1.0	1.0	0.333333	
12	13	11.877576	0.000000	0.75	0.435403	0.875096	1.0	1.0	1.000000	
13	14	12.540761	0.000000	0.75	0.538208	0.795722	1.0	1.0	0.333333	
14	15	11.964007	0.000000	0.75	0.435403	0.805747	1.0	1.0	0.333333	

	Id	SalePrice	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Land
15	16	11.790565	0.147059	0.25	0.323585	0.572143	1.0	1.0	0.000000	
16	17	11.911708	0.000000	0.75	0.435403	0.817436	1.0	1.0	0.333333	
17	18	11.407576	0.411765	0.75	0.451188	0.800953	1.0	1.0	0.000000	
18	19	11.976666	0.000000	0.75	0.418925	0.897103	1.0	1.0	0.000000	
19	20	11.842236	0.000000	0.75	0.440738	0.657391	1.0	1.0	0.000000	

In [61]:
```
data.to_csv('X_train_outlier_removed_3.csv',index=False)
```

Feature Selection

In [62]:
```
dataset=pd.read_csv('X_train_outlier_removed_3.csv')
```

In [63]:
```
dataset.head()
```

Out[63]:

	Id	SalePrice	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandC
0	1	12.247699	0.235294	0.75	0.413268	0.702292	1.0	1.0	0.000000	0.
1	2	12.109016	0.000000	0.75	0.490307	0.753770	1.0	1.0	0.000000	0.
2	3	12.317171	0.235294	0.75	0.429990	0.817759	1.0	1.0	0.333333	0.
3	4	11.849405	0.294118	0.75	0.383633	0.751663	1.0	1.0	0.333333	0.
4	5	12.429220	0.235294	0.75	0.508439	0.913414	1.0	1.0	0.333333	0.

In [64]:
```
len(dataset.columns)
```

Out[64]: 84

In [65]:
```
##Capture the dependent feature
```

```
y_train=dataset[['SalePrice']]
```

In [66]:
```
y_train
```

Out[66]:

	SalePrice
0	12.247699
1	12.109016
2	12.317171
3	11.849405
4	12.429220
...	...
1455	12.072547
1456	12.254868
1457	12.493133
1458	11.864469
1459	11.901590

1460 rows × 1 columns

In [67]:
```
##Drop dependent feature from dataset
X_train=dataset.drop(['Id','SalePrice'],axis=1)
```

In [68]:
```
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel

pd.pandas.set_option('display.max_columns',None)
```

In [69]:
```
### Apply Feature Selection
# first, I specify the Lasso Regression model, and I
# select a suitable alpha (equivalent of penalty).
# The bigger the alpha the less features that will be selected.
```

```
# Then I use the selectFromModel object from sklearn, which
# will select the features which coefficients are non-zero

feature_sel_model = SelectFromModel(Lasso(alpha=0.005,random_state=0))
```

In [70]:
```
feature_sel_model.fit(X_train,y_train)
```

Out[70]:
```
SelectFromModel(estimator=Lasso(alpha=0.005, copy_X=True, fit_intercept
=True,
                                max_iter=1000, normalize=False, positiv
e=False,
                                precompute=False, random_state=0,
                                selection='cyclic', tol=0.0001,
                                warm_start=False),
                max_features=None, norm_order=1, prefit=False, threshol
d=None)
```

In [71]:
```
feature_sel_model.get_support()
```

Out[71]:
```
array([False, False, False,  True, False, False, False, False, False,
       False, False,  True, False, False, False, False,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False,  True,  True, False,  True, False,  True, False, False,
       False,  True, False,  True,  True, False,  True, False, False,
        True, False, False, False, False, False, False,  True, False,
       False, False,  True,  True, False,  True,  True, False, False,
        True, False,  True,  True, False, False, False, False, False,
       False, False, False, False, False, False,  True, False, False,
       False])
```

In [72]:
```
# let's print the number of total and selected features

selected_feat = X_train.columns[(feature_sel_model.get_support())]

#Let's print some stats
print('total features: {}'.format((X_train.shape[1])))
print('selected features: {}'.format(len(selected_feat)))
```

```
print('features with coefficients shrank to zero: {}'.format(X_train.shape[1]-len(selected_feat)))
```

```
total features: 82
selected features: 22
features with coefficients shrank to zero: 60
```

In [73]:
```
selected_feat
```

Out[73]:
```
Index(['LotArea', 'Neighborhood', 'OverallQual', 'YearRemodAdd', 'Foundation',
       'BsmtQual', 'BsmtExposure', 'BsmtFinSF1', 'TotalBsmtSF', 'HeatingQC',
       'CentralAir', '1stFlrSF', 'GrLivArea', 'KitchenQual', 'FireplaceQu',
       'GarageType', 'GarageFinish', 'GarageCars', 'GarageCond', 'WoodDeckSF',
       'OpenPorchSF', 'SaleCondition'],
      dtype='object')
```

In [74]:
```
X_train=X_train[selected_feat]
```

In [75]:
```
X_train.head()
```

Out[75]:

	LotArea	Neighborhood	OverallQual	YearRemodAdd	Foundation	BsmtQual	BsmtExposure
0	0.702292	0.625000	0.666667	0.098361	1.0	0.75	0.25
1	0.753770	0.833333	0.555556	0.524590	0.4	0.75	1.00
2	0.817759	0.625000	0.666667	0.114754	1.0	0.75	0.50
3	0.751663	0.708333	0.666667	0.606557	0.2	0.50	0.25
4	0.913414	1.000000	0.777778	0.147541	1.0	0.75	0.75

In [76]:
```
X_train.to_csv("X_train_.csv")
```

Test Data

In [77]:
```
df2=pd.read_csv('test1.csv')
```

In [78]:
```
df2
```

Out[78]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS
...	...	...	...	...	...	...	...	...	...
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl

1459 rows × 80 columns

In [79]:
```
#Checking percentage of nan values present
#Make the list of features with missing values
features_with_na= [feat for feat in df2.columns if df2[feat].isnull().sum()>=1]

#Print feature name and percentage of missung values
for feature in features_with_na:
    print(feature, np.round(df2[feature].isnull().mean(), 4),  ' % missing values')
```

```
MSZoning 0.0027  % missing values
LotFrontage 0.1556  % missing values
Alley 0.9267  % missing values
Utilities 0.0014  % missing values
Exterior1st 0.0007  % missing values
Exterior2nd 0.0007  % missing values
MasVnrType 0.011  % missing values
MasVnrArea 0.0103  % missing values
BsmtQual 0.0302  % missing values
BsmtCond 0.0308  % missing values
BsmtExposure 0.0302  % missing values
BsmtFinType1 0.0288  % missing values
BsmtFinSF1 0.0007  % missing values
BsmtFinType2 0.0288  % missing values
BsmtFinSF2 0.0007  % missing values
BsmtUnfSF 0.0007  % missing values
TotalBsmtSF 0.0007  % missing values
BsmtFullBath 0.0014  % missing values
BsmtHalfBath 0.0014  % missing values
KitchenQual 0.0007  % missing values
Functional 0.0014  % missing values
FireplaceQu 0.5003  % missing values
GarageType 0.0521  % missing values
GarageYrBlt 0.0535  % missing values
GarageFinish 0.0535  % missing values
GarageCars 0.0007  % missing values
GarageArea 0.0007  % missing values
GarageQual 0.0535  % missing values
GarageCond 0.0535  % missing values
PoolQC 0.9979  % missing values
Fence 0.8012  % missing values
MiscFeature 0.965  % missing values
SaleType 0.0007  % missing values
```

Feature Engineering

In [80]:
```
## Let us capture all the nan values
## First lets handle Categorical features which are missing

features_nan= [feature for feature in df2.columns if df2[feature].dtype
s=='O' and df2[feature].isnull().sum()>=1]

for feature in features_nan:
    print('{}: {} missing values'.format(feature,np.round(df2[feature].
isnull().mean(),4)))
```

```
MSZoning: 0.0027 missing values
Alley: 0.9267 missing values
Utilities: 0.0014 missing values
Exterior1st: 0.0007 missing values
Exterior2nd: 0.0007 missing values
MasVnrType: 0.011 missing values
BsmtQual: 0.0302 missing values
BsmtCond: 0.0308 missing values
BsmtExposure: 0.0302 missing values
BsmtFinType1: 0.0288 missing values
BsmtFinType2: 0.0288 missing values
KitchenQual: 0.0007 missing values
Functional: 0.0014 missing values
FireplaceQu: 0.5003 missing values
GarageType: 0.0521 missing values
GarageFinish: 0.0535 missing values
GarageQual: 0.0535 missing values
GarageCond: 0.0535 missing values
PoolQC: 0.9979 missing values
Fence: 0.8012 missing values
MiscFeature: 0.965 missing values
SaleType: 0.0007 missing values
```

In [81]:
```
features_nan2= [feature for feature in df2.columns if df2[feature].dtyp
es==('float64'or'int64'or'int32'or'O') and df2[feature].isnull().sum()>
=1]
```

In [82]:
```
features_nan2
```

Out[82]:
```
['LotFrontage',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 'BsmtFullBath',
 'BsmtHalfBath',
 'GarageYrBlt',
 'GarageCars',
 'GarageArea']
```

In [83]:
```
## Replace missing value with a new label
def replace_cat_feature(df2,features_nan):
    data=df2.copy()

    data[features_nan] = np.where(data[features_nan].isnull(),'Missing'
,data[features_nan])
    #data[features_nan]=data[features_nan].fillna('Missing')
    return data
df2=replace_cat_feature(df2,features_nan)
```

In [84]:
```
df2[features_nan].isnull().sum()
```

Out[84]:
```
MSZoning        0
Alley           0
Utilities       0
Exterior1st     0
Exterior2nd     0
MasVnrType      0
BsmtQual        0
BsmtCond        0
BsmtExposure    0
BsmtFinType1    0
BsmtFinType2    0
KitchenQual     0
Functional      0
FireplaceQu     0
GarageType      0
```

```
GarageFinish    0
GarageQual      0
GarageCond      0
PoolQC          0
Fence           0
MiscFeature     0
SaleType        0
dtype: int64
```

In [85]:
```
df2.head()
```

Out[85]:

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

In [86]:
```
## Now lets check for numerical variables that contains missing values

numerical_with_nan=[feature for feature in df2.columns if df2[feature].
isnull().sum()>=1 and df2[feature].dtypes!='O']

## We will print the numerical nan variables and percentage of missing
 values

for feature in numerical_with_nan:
    print("{}: {}% missing value".format(feature,np.round(df2[feature].
isnull().mean()*100,4)))
```

```
LotFrontage: 15.5586% missing value
MasVnrArea: 1.0281% missing value
BsmtFinSF1: 0.0685% missing value
BsmtFinSF2: 0.0685% missing value
BsmtUnfSF: 0.0685% missing value
```

```
TotalBsmtSF: 0.0685% missing value
BsmtFullBath: 0.1371% missing value
BsmtHalfBath: 0.1371% missing value
GarageYrBlt: 5.3461% missing value
GarageCars: 0.0685% missing value
GarageArea: 0.0685% missing value
```

In [87]:
```
#Replacing the numerical missing values
for feature in numerical_with_nan:
    ## We will replace by using median since there are outliers

    ## create a new feature to capture nan values
    df2[feature+'nan'] = np.where(df2[feature].isnull(),1,0)

    ## ## We will replace by using median since there are outliers
    df2[feature].fillna(df2[feature].median(),inplace=True)

df2[numerical_with_nan].isnull().sum()
```

Out[87]:
```
LotFrontage     0
MasVnrArea      0
BsmtFinSF1      0
BsmtFinSF2      0
BsmtUnfSF       0
TotalBsmtSF     0
BsmtFullBath    0
BsmtHalfBath    0
GarageYrBlt     0
GarageCars      0
GarageArea      0
dtype: int64
```

In [88]:
```
#For temporal_feature
temporal_feat= [feature for feature in df2.columns if 'Year' in feature
 or 'Yr'in feature]
temporal_feat
```

Out[88]: ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold', 'GarageYrBltna

Out[88]: ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold', 'GarageYrBltha
n']

In [89]:
```
## Temporal Variables (Date Time Variables)

for feature in ['YearBuilt','YearRemodAdd','GarageYrBlt']:

    df2[feature]=df2['YrSold']-df2[feature]
```

In [90]:
```
df2.head()
```

Out[90]:

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

In [91]:
```
df2[['YearBuilt','YearRemodAdd','GarageYrBlt']].head()
```

Out[91]:

	YearBuilt	YearRemodAdd	GarageYrBlt
0	49	49	49.0
1	52	52	52.0
2	13	12	13.0
3	12	12	12.0
4	18	18	18.0

In [92]:
```
for feature in categorical_features:
    df3=pd.read_csv('train1.csv')
```

```
    df3[feature]=np.where(df3[feature].isnull(),'Missing',df3[feature])
    labels_ordered=df3.groupby([feature])['SalePrice'].mean().sort_valu
es().index
    labels_ordered={k:i for i,k in enumerate(labels_ordered,0)}
    df2[feature]=df2[feature].map(labels_ordered)  # For test data as S
alePrice column is not present
```

In [93]:
```
df2
```

Out[93]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1461	20	2.0	80.0	11622	1	2	0	1
1	1462	20	3.0	81.0	14267	1	2	1	1
2	1463	60	3.0	74.0	13830	1	2	1	1
3	1464	60	3.0	78.0	9978	1	2	1	1
4	1465	120	3.0	43.0	5005	1	2	1	3
...	...	...	...	...	...	...	...	...	...
1454	2915	160	1.0	21.0	1936	1	2	0	1
1455	2916	160	1.0	21.0	1894	1	2	0	1
1456	2917	20	3.0	160.0	20000	1	2	0	1
1457	2918	85	3.0	62.0	10441	1	2	0	1
1458	2919	60	3.0	74.0	9627	1	2	0	1

1459 rows × 91 columns

Missing Values

In [94]:
```
## Let us capture all the nan values
## First lets handle Categorical features which are missing
```

```
features_nan= [feature for feature in df2.columns if df2[feature].dtype
s=='O' and df2[feature].isnull().sum()>=1]
```

In [95]:
```
continuous_feature
```

Out[95]:
```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch',
 'SalePrice']
```

In [96]:
```
continuous_feature.append('SalePrice')
```

In [97]:
```
continuous_feature
```

Out[97]:
```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
```

```
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch',
 'SalePrice',
 'SalePrice']
```

In [103]:
```
continuous_feature.remove('SalePrice')
```

In [104]:
```
continuous_feature
```

Out[104]:
```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
 '2ndFlrSF',
 'GrLivArea',
 'GarageArea',
 'WoodDeckSF',
 'OpenPorchSF',
 'EnclosedPorch',
 'ScreenPorch']
```

In [105]:
```
continuous_feature_test = continuous_feature
```

In [266]:
```
continuous_feature_test
```

Out[266]:
```
['LotFrontage',
 'LotArea',
 'MasVnrArea',
 'BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 '1stFlrSF',
```

```
'2ndFlrSF',
'GrLivArea',
'GarageArea',
'WoodDeckSF',
'OpenPorchSF',
'EnclosedPorch',
'ScreenPorch']
```

In [269]:
```
df3
```

Out[269]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	20	2.0	1.685370	2.338024	1	2	0	
1	1462	20	3.0	1.687642	2.357620	1	2	1	
2	1463	60	3.0	1.671001	2.354672	1	2	1	
3	1464	60	3.0	1.680725	2.323195	1	2	1	
4	1465	120	3.0	1.565317	2.253226	1	2	1	
...	...	...	...	...	...	...	...	...	...
1454	2915	160	1.0	1.552447	2.223847	1	2	0	
1455	2916	160	1.0	1.552447	2.223847	1	2	0	
1456	2917	20	3.0	1.734031	2.377858	1	2	0	
1457	2918	85	3.0	1.637663	2.327628	1	2	0	
1458	2919	60	3.0	1.671001	2.319681	1	2	0	

1459 rows × 91 columns

In [107]:
```
for feature in continuous_feature_test:
    IQR = np.percentile(df2[feature],75) - np.percentile(df2[feature],25)
    lb = np.percentile(df2[feature],25)-IQR*1.5
    ub = np.percentile(df2[feature],75)+IQR*1.5
```

```
    df2[feature] = np.where(df2[feature]>ub,ub,df2[feature])
    df2[feature] = np.where(df2[feature]<lb,lb,df2[feature])
    df2[feature] = np.log1p(df2[feature])
```

In [270]:
```
outlier_dict = dict()
for feature in continuous_feature_test:
    IQR = np.percentile(df4[feature],75) - np.percentile(df4[feature],2
5)
    lb = np.percentile(df4[feature],25)-IQR*1.5
    ub = np.percentile(df4[feature],75)+IQR*1.5

    df4[feature] = np.where(df4[feature]>ub,ub,df4[feature])
    df4[feature] = np.where(df4[feature]<lb,lb,df4[feature])
    df4[feature] = np.log1p(df4[feature])

    dict_feature = dict()

    dict_feature['IQR'] = IQR
    dict_feature['Lower_bound'] = lb
    dict_feature['Upper_bound'] = ub

    outlier_dict[feature] = dict_feature
```

In [276]:
```
outlier_dict_percentile = dict()
for feature in continuous_feature_test:
    percentile_99th = np.percentile(df4[feature],99)
    percentile_1st = np.percentile(df4[feature],1)
    median_value = df[feature].median()

    #df4[feature] = np.where(df4[feature]>ub,ub,df4[feature])
    #df4[feature] = np.where(df4[feature]<lb,lb,df4[feature])
    #df4[feature] = np.log1p(df4[feature])

    dict_feature_percentile = dict()

    dict_feature_percentile['1st'] = percentile_1st
    dict_feature_percentile['99th'] = percentile_99th
```

```
    dict_feature_percentile['Median'] = median_value

    outlier_dict_percentile[feature] = dict_feature_percentile
```

In [313]:
```
outlier_dict
```

Out[313]:
```
{'LotFrontage': {'IQR': 0.04935468226831574,
  'Lower_bound': 1.5573383767939242,
  'Upper_bound': 1.7547571058671871},
 'LotArea': {'IQR': 0.04379367920570498,
  'Lower_bound': 2.2276674956502895,
  'Upper_bound': 2.4028422124731095},
 'MasVnrArea': {'IQR': 1.807263688716924,
  'Lower_bound': -2.710895533075386,
  'Upper_bound': 4.51815922179231},
 'BsmtFinSF1': {'IQR': 2.0313085443910763,
  'Lower_bound': -3.0469628165866145,
  'Upper_bound': 5.078271360977691},
 'BsmtFinSF2': {'IQR': 0.0, 'Lower_bound': 0.0, 'Upper_bound': 0.0},
 'BsmtUnfSF': {'IQR': 0.18331923565035124,
  'Lower_bound': 1.5806774813803603,
  'Upper_bound': 2.313954423981765},
 'TotalBsmtSF': {'IQR': 0.06419941643668414,
  'Lower_bound': 1.9404545836622826,
  'Upper_bound': 2.1972522494090194},
 '1stFlrSF': {'IQR': 0.057334072754528176,
  'Lower_bound': 1.9647389718441362,
  'Upper_bound': 2.194075262862249},
 '2ndFlrSF': {'IQR': 2.0172564188484525,
  'Lower_bound': -3.025884628272679,
  'Upper_bound': 5.043141047121131},
 'GrLivArea': {'IQR': 0.05240691570189382,
  'Lower_bound': 2.00329608628795,
  'Upper_bound': 2.212923749095525},
 'GarageArea': {'IQR': 0.0839762094741221,
  'Lower_bound': 1.7858261957612633,
  'Upper_bound': 2.1217310336577517},
 'WoodDeckSF': {'IQR': 1.813178226960568,
  'Lower_bound': -2.719767340440852,
```

```
 'Upper_bound': 4.53294556740142},
'OpenPorchSF': {'IQR': 1.6659050929776016,
 'Lower_bound': -2.4988576394664026,
 'Upper_bound': 4.164762732444004},
'EnclosedPorch': {'IQR': 0.0, 'Lower_bound': 0.0, 'Upper_bound': 0.0},
'ScreenPorch': {'IQR': 0.0, 'Lower_bound': 0.0, 'Upper_bound': 0.0}}
```

In [277]:
```
outlier_dict_percentile
```

Out[277]:
```
{'LotFrontage': {'1st': 0.9389670210672053,
  '99th': 1.005777194524828,
  'Median': 4.248495242049359},
 'LotArea': {'1st': 1.1717597387980652,
  '99th': 1.2172418461444838,
  'Median': 9.156886838722746},
 'MasVnrArea': {'1st': 0.0, '99th': 1.080725717331393, 'Median': 0.0},
 'BsmtFinSF1': {'1st': 0.0,
  '99th': 1.1415626429333063,
  'Median': 5.951942943437755},
 'BsmtFinSF2': {'1st': 0.0, '99th': 0.0, 'Median': 0.0},
 'BsmtUnfSF': {'1st': 0.9480519541451999,
  '99th': 1.1411474047821348,
  'Median': 6.170651297395139},
 'TotalBsmtSF': {'1st': 1.0785641896913982,
  '99th': 1.1495254945144728,
  'Median': 6.900226885665022},
 '1stFlrSF': {'1st': 1.0916336887769167,
  '99th': 1.1505971766270453,
  'Median': 6.992096005027085},
 '2ndFlrSF': {'1st': 0.0, '99th': 1.1339307567244956, 'Median': 0.0},
 'GrLivArea': {'1st': 1.1040865573224683,
  '99th': 1.1578527534077312,
  'Median': 7.289610521451167},
 'GarageArea': {'1st': 1.0245444885448576,
  '99th': 1.1194590497357892,
  'Median': 6.175867270105761},
 'WoodDeckSF': {'1st': 0.0, '99th': 1.0824768370427216, 'Median': 0.0},
 'OpenPorchSF': {'1st': 0.0,
  '99th': 1.0382629710766498,
```

```
 'Median': 3.258096538021482},
'EnclosedPorch': {'1st': 0.0, '99th': 0.0, 'Median': 0.0},
'ScreenPorch': {'1st': 0.0, '99th': 0.0, 'Median': 0.0}}
```

In [272]:
```
np.save('outlier_dict.npy', outlier_dict)
```

In [273]:
```
outlier_dict = np.load('outlier_dict.npy',allow_pickle='TRUE').item()
```

In [274]:
```
outlier_dict
```

Out[274]:
```
{'LotFrontage': {'IQR': 0.04935468226831574,
  'Lower_bound': 1.5573383767939242,
  'Upper_bound': 1.7547571058671871},
 'LotArea': {'IQR': 0.04379367920570498,
  'Lower_bound': 2.2276674956502895,
  'Upper_bound': 2.4028422124731095},
 'MasVnrArea': {'IQR': 1.807263688716924,
  'Lower_bound': -2.710895533075386,
  'Upper_bound': 4.51815922179231},
 'BsmtFinSF1': {'IQR': 2.0313085443910763,
  'Lower_bound': -3.0469628165866145,
  'Upper_bound': 5.078271360977691},
 'BsmtFinSF2': {'IQR': 0.0, 'Lower_bound': 0.0, 'Upper_bound': 0.0},
 'BsmtUnfSF': {'IQR': 0.18331923565035124,
  'Lower_bound': 1.5806774813803603,
  'Upper_bound': 2.313954423981765},
 'TotalBsmtSF': {'IQR': 0.06419941643668414,
  'Lower_bound': 1.9404545836622826,
  'Upper_bound': 2.1972522494090194},
 '1stFlrSF': {'IQR': 0.057334072754528176,
  'Lower_bound': 1.9647389718441362,
  'Upper_bound': 2.194075262862249},
 '2ndFlrSF': {'IQR': 2.0172564188484525,
  'Lower_bound': -3.025884628272679,
  'Upper_bound': 5.043141047121131},
 'GrLivArea': {'IQR': 0.05240691570189382,
  'Lower_bound': 2.00329608628795,
  'Upper_bound': 2.212923749095525},
```

```
 'GarageArea': {'IQR': 0.0839762094741221,
  'Lower_bound': 1.7858261957612633,
  'Upper_bound': 2.1217310336577517},
 'WoodDeckSF': {'IQR': 1.813178226960568,
  'Lower_bound': -2.719767340440852,
  'Upper_bound': 4.53294556740142},
 'OpenPorchSF': {'IQR': 1.6659050929776016,
  'Lower_bound': -2.4988576394664026,
  'Upper_bound': 4.164762732444004},
 'EnclosedPorch': {'IQR': 0.0, 'Lower_bound': 0.0, 'Upper_bound': 0.0},
 'ScreenPorch': {'IQR': 0.0, 'Lower_bound': 0.0, 'Upper_bound': 0.0}}
```

In [311]:
```
df5 = scaler.transform(entry.values)
```

In [312]:
```
df5
```

Out[312]:
```
array([[ 0.23529412,  0.75      , -0.79284434, -2.46147135,  1.
,
         1.        ,  0.33333333,  0.33333333,  1.        ,  0.5
,
         0.        ,  0.58333333,  0.5       ,  0.57142857,  1.
,
         0.85714286,  0.55555556,  0.5       ,  0.125     ,  0.2786885
2,
         0.2       ,  0.28571429,  0.57142857,  0.6       ,  0.25
,
         0.        ,  0.33333333,  0.75      ,  1.        ,  0.75
,
         0.75      ,  0.25      ,  0.83333333,  0.        ,  0.8333333
3,
         0.        ,  0.14933515, -0.68589666,  1.        ,  0.75
,
         1.        ,  1.        , -2.52667414,  0.14869381,  0.
,
        -2.22033716,  0.        ,  0.        ,  0.66666667,  0.5
,
         0.375     ,  0.33333333,  0.33333333,  0.41666667,  1.
,
         0.33333333,  0.6       ,  0.83333333,  0.1588785 ,  1.
```

```
,
         0.5       ,  0.15846407,  0.6       ,  1.        ,  1.
,
         0.17051883,  0.19276367,  0.        ,  0.        ,  0.
,
         0.        ,  0.        ,  1.        ,  0.75      ,  0.
,
         0.27272727,  1.        ,  0.5       ,  0.8       ,  0.
,
         0.        ,  0.        ],
       [ 0.        ,  0.75      , -0.79582296, -2.46419335,  1.
,
         1.        ,  0.33333333,  0.33333333,  1.        ,  0.
,
         0.        ,  0.58333333,  0.5       ,  0.57142857,  1.
,
         0.71428571,  0.55555556,  0.75      ,  0.13235294,  0.0655737
7,
         0.2       ,  0.28571429,  0.57142857,  0.6       ,  0.25
,
         0.        ,  0.33333333,  0.5       ,  1.        ,  0.75
,
         0.75      ,  0.25      ,  0.66666667,  0.14939098,  0.8333333
3,
         0.        ,  0.14165235, -0.68131628,  1.        ,  1.
,
         1.        ,  1.        , -2.51681676,  0.        ,  0.
,
        -2.2264923 ,  0.33333333,  0.        ,  0.66666667,  0.
,
         0.375     ,  0.33333333,  0.33333333,  0.33333333,  1.
,
         0.        ,  0.2       ,  0.83333333,  0.1682243 ,  1.
,
         0.5       ,  0.15813936,  0.6       ,  1.        ,  1.
,
         0.17913993,  0.17098136,  0.        ,  0.        ,  0.
,
         0.        ,  0.        ,  0.75      ,  0.25      ,  0.0322580
```

```
6,
        0.18181818,  1.        ,  0.5       ,  0.8       ,  1.
,
        0.        ,  0.        ],
       [ 0.23529412,  0.75      , -0.79748355, -2.46356472,  1.
,
        1.        ,  0.33333333,  0.33333333,  1.        ,  0.
,
        0.        ,  0.58333333,  0.5       ,  0.57142857,  1.
,
        0.85714286,  0.55555556,  0.5       ,  0.08823529,  0.2131147
5,
        0.2       ,  0.28571429,  0.78571429,  0.8       ,  0.25
,
        0.        ,  0.33333333,  0.75      ,  1.        ,  0.75
,
        0.75      ,  0.25      ,  0.83333333,  0.        ,  0.8333333
3,
        0.        ,  0.14952897, -0.68552412,  1.        ,  0.75
,
        1.        ,  1.        , -2.52590066,  0.14710633,  0.
,
       -2.22255773,  0.        ,  0.        ,  0.66666667,  0.5
,
        0.375     ,  0.33333333,  0.33333333,  0.41666667,  1.
,
        0.33333333,  0.8       ,  0.83333333,  0.11214953,  1.
,
        0.5       ,  0.15767064,  0.6       ,  1.        ,  1.
,
        0.        ,  0.191258  ,  0.        ,  0.        ,  0.
,
        0.        ,  0.        ,  1.        ,  0.75      ,  0.
,
        0.36363636,  1.        ,  0.5       ,  0.8       ,  0.
,
        0.        ,  0.        ],
       [ 0.        ,  0.75      , -0.78963255, -2.46126415,  1.
,
```

```
          1.        ,  0.        ,  0.33333333,  1.        ,  0.
,
          0.        ,  0.58333333,  0.5       ,  0.57142857,  1.
,
          0.71428571,  0.66666667,  0.5       ,  0.14705882,  0.3442623
,
          0.2       ,  0.28571429,  0.57142857,  0.6       ,  0.25
,
          0.        ,  0.33333333,  0.75      ,  1.        ,  0.75
,
          0.75      ,  1.        ,  1.        ,  0.14718463,  0.8333333
3,
          0.        ,  0.14851019, -0.68021437,  1.        ,  0.75
,
          1.        ,  1.        , -2.51422013,  0.        ,  0.
,
         -2.2241952 ,  0.33333333,  0.        ,  0.33333333,  0.5
,
          0.25      ,  0.33333333,  0.66666667,  0.25      ,  1.
,
          0.33333333,  0.        ,  0.83333333,  0.18691589,  0.3333333
3,
          0.5       ,  0.15942294,  0.6       ,  1.        ,  1.
,
          0.17242555,  0.        ,  0.        ,  0.        ,  0.
,
          0.        ,  0.        ,  1.        ,  0.75      ,  0.
,
          0.09090909,  1.        ,  0.5       ,  0.8       ,  0.
,
          0.        ,  0.        ],
        [ 0.        ,  0.75      , -0.79465653, -2.46356761,  1.
,
          1.        ,  0.        ,  0.33333333,  1.        ,  0.5
,
          0.        ,  0.41666667,  0.5       ,  0.57142857,  1.
,
          0.71428571,  0.33333333,  0.5       ,  0.29411765,  0.6721311
5,
```

```
         0.2       ,  0.28571429,  0.64285714,  0.66666667,  0.25
       ,
         0.        ,  0.33333333,  0.75      ,  0.4       ,  0.5
       ,
         0.75      ,  0.25      ,  0.66666667,  0.14854051,  0.5
       ,
         0.        ,  0.12759079, -0.68430057,  1.        ,  0.5
       ,
         1.        ,  1.        , -2.52336026,  0.        ,  0.
       ,
        -2.23228098,  0.33333333,  0.        ,  0.33333333,  0.
       ,
         0.25      ,  0.33333333,  0.33333333,  0.16666667,  1.
       ,
         0.        ,  0.2       ,  0.83333333,  0.37383178,  1.
       ,
         0.5       ,  0.15967177,  0.6       ,  1.        ,  1.
       ,
         0.17444781,  0.        ,  0.        ,  0.        ,  0.
       ,
         0.        ,  0.        ,  0.5       ,  0.75      ,  0.
       ,
         0.27272727,  1.        ,  0.5       ,  0.8       ,  0.
       ,
         0.        ,  0.        ]])
```

In [310]:
```
entry = entry[feature_scale]
```

In [ ]:
```

```

In [308]:
```
len(entry[feature_scale].columns)
```

Out[308]: 82

In [304]:
```
entry = pd.DataFrame(df4.iloc[5:10],columns = df4.columns)
```

In [307]:
```
X_train
```

Out[307]:

	LotArea	Neighborhood	OverallQual	YearRemodAdd	Foundation	BsmtQual	BsmtExposur
0	0.702292	0.625000	0.666667	0.098361	1.0	0.75	0.2
1	0.753770	0.833333	0.555556	0.524590	0.4	0.75	1.0
2	0.817759	0.625000	0.666667	0.114754	1.0	0.75	0.5
3	0.751663	0.708333	0.666667	0.606557	0.2	0.50	0.2
4	0.913414	1.000000	0.777778	0.147541	1.0	0.75	0.7
...	...	...	...	...	...	...	.
1455	0.676006	0.583333	0.555556	0.131148	1.0	0.75	0.2
1456	0.881485	0.541667	0.555556	0.377049	0.4	0.75	0.2
1457	0.729610	0.708333	0.666667	0.081967	0.6	0.50	0.2
1458	0.758657	0.416667	0.444444	0.245902	0.4	0.50	0.5
1459	0.767689	0.208333	0.444444	0.721311	0.4	0.50	0.2

1460 rows × 22 columns

In [262]:
```
df4 = df3
```

In [ ]:
```

```

In [263]:
```
df2
```

Out[263]:

	LotArea	Neighborhood	OverallQual	YearRemodAdd	Foundation	BsmtQual	BsmtExposur
0	2.338024	10	5	49	2	2	
1	2.357620	10	6	52	2	2	
2	2.354672	14	5	12	5	3	
3	2.323195	14	6	12	5	2	
4	2.253226	22	8	18	5	3	

	LotArea	Neighborhood	OverallQual	YearRemodAdd	Foundation	BsmtQual	BsmtExposur
...	...	...	...	...	...	...	.
1454	2.223847	0	4	36	2	2	
1455	2.223847	0	4	36	2	2	
1456	2.377858	11	5	10	2	2	
1457	2.327628	11	5	14	5	3	
1458	2.319681	11	7	12	5	3	

1459 rows × 22 columns

In [259]:
```
outlier_dict = dict()
outlier_dict = {a:1,2,3}
```

```
  File "<ipython-input-259-ed9478547cb2>", line 2
    outlier_dict = {a:1,2,3}
                      ^
SyntaxError: invalid syntax
```

In [164]:
```
#Numerical Variables
#Since the numerical variables are skewed, we will perform log normal d
istribution
```

In [191]:
```
num_features=['LotFrontage', 'LotArea', '1stFlrSF', 'GrLivArea']
#for feature in num_features:
    #df2[feature]=np.log(df2[feature])
```

In [108]:
```
df2.shape
```

Out[108]:
```
(1459, 91)
```

In [109]:
```
## Replace missing value with a new label
def replace_cat_feature(df,features_nan):
    data=df.copy()
```

```
    data[features_nan] = np.where(data[features_nan].isnull(),'Missing'
,data[features_nan])
    #data[features_nan]=data[features_nan].fillna('Missing')
    return data
df2=replace_cat_feature(df2,features_nan)
```

In [110]:
```
## Let us capture all the nan values
## First lets handle Categorical features which are missing

features_nan= [feature for feature in df2.columns if df2[feature].dtype
s=='O' and df2[feature].isnull().sum()>=1]

for feature in features_nan:
    print('{}: {} missing values'.format(feature,np.round(df2[feature].
isnull().mean(),4)))
```

In [111]:
```
df2.head()
```

Out[111]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1461	20	2.0	1.685370	2.338024	1	2	0	1
1	1462	20	3.0	1.687642	2.357620	1	2	1	1
2	1463	60	3.0	1.671001	2.354672	1	2	1	1
3	1464	60	3.0	1.680725	2.323195	1	2	1	1
4	1465	120	3.0	1.565317	2.253226	1	2	1	3

In [112]:
```
df2.to_csv('test_outlier_removed',index=False)
```

Prediciton and selecting the Algorithm

In [113]:
```
import xgboost
```

```
regressor=xgboost.XGBRegressor()
```

```
C:\Users\Hp\Anaconda3\lib\site-packages\dask\dataframe\utils.py:14: Fut
ureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [114]:
```
booster=['gbtree','gblinear']
base_score=[0.25,0.5,0.75,1]
```

In [115]:
```
## Hyper Parameter Optimization


n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster=['gbtree','gblinear']
learning_rate=[0.05,0.1,0.15,0.20]
min_child_weight=[1,2,3,4]

# Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators': n_estimators,
    'max_depth':max_depth,
    'learning_rate':learning_rate,
    'min_child_weight':min_child_weight,
    'booster':booster,
    'base_score':base_score
    }
```

In [116]:
```
# Set up the random search with 4-fold cross validation
from sklearn.model_selection import RandomizedSearchCV
#from sklearn.base import clone
# Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=regressor,
            param_distributions=hyperparameter_grid,
            cv=5, n_iter=50,
            scoring = 'neg_mean_absolute_error',n_jobs = 4,
            verbose = 5,
```

```
            return_train_score = True,
            random_state=42)
```

In [117]:
```
random_cv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:   13.3s
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed:   56.0s
[Parallel(n_jobs=4)]: Done 154 tasks      | elapsed:  1.4min
[Parallel(n_jobs=4)]: Done 250 out of 250 | elapsed:  1.9min finished
```

Out[117]:
```
RandomizedSearchCV(cv=5, error_score=nan,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_...
                   iid='deprecated', n_iter=50, n_jobs=4,
                   param_distributions={'base_score': [0.25, 0.5, 0.75, 1],
                                        'booster': ['gbtree', 'gblinear'],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2],
                                        'max_depth': [2, 3, 5, 10, 15],
                                        'min_child_weight': [1, 2, 3,
```

```
4],
                                                'n_estimators': [100, 500, 900,
1100,
                                                                1500]},
                         pre_dispatch='2*n_jobs', random_state=42, refit=Tru
e,
                         return_train_score=True, scoring='neg_mean_absolute_
error',
                         verbose=5)
```

In [185]:
```
random_cv.best_estimator_
```

Out[185]:
```
XGBRegressor(base_score=0.25, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-
1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=2,
             min_child_weight=4, missing=nan, monotone_constraints
='()',
             n_estimators=900, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method
='exact',
             validate_parameters=1, verbosity=None)
```

In [119]:
```
random_cv.best_params_
```

Out[119]:
```
{'n_estimators': 900,
 'min_child_weight': 4,
 'max_depth': 2,
 'learning_rate': 0.05,
 'booster': 'gbtree',
 'base_score': 0.25}
```

In [120]:
```
regressor=xgboost.XGBRegressor(base_score=0.25, booster='gbtree', colsa
mple_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1
,
```

```
              importance_type='gain', interaction_constraints='',
              learning_rate=0.05, max_delta_step=0, max_depth=2,
              min_child_weight=4, missing=None, monotone_constraints=
'()',
              n_estimators=900, n_jobs=0, num_parallel_tree=1,
              objective='reg:squarederror', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1)
```

In [121]:
```
regressor.fit(X_train,y_train)
```

Out[121]:
```
XGBRegressor(base_score=0.25, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-
1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.05, max_delta_step=0, max_depth=2,
             min_child_weight=4, missing=None, monotone_constraints
='()',
             n_estimators=900, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method
='exact',
             validate_parameters=1, verbosity=None)
```

In [152]:
```
df2 = pd.read_csv('test_outlier_removed')
```

In [153]:
```
df3=df2.copy()
```

In [167]:
```
df3
```

Out[167]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	20	2.0	1.685370	2.338024	1	2	0	
1	1462	20	3.0	1.687642	2.357620	1	2	1	
2	1463	60	3.0	1.671001	2.354672	1	2	1	
3	1464	60	3.0	1.680725	2.323195	1	2	1	

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
4	1465	120	3.0	1.565317	2.253226	1	2	1	
...	...	...	...	...	...	...	...	...	.
1454	2915	160	1.0	1.552447	2.223847	1	2	0	
1455	2916	160	1.0	1.552447	2.223847	1	2	0	
1456	2917	20	3.0	1.734031	2.377858	1	2	0	
1457	2918	85	3.0	1.637663	2.327628	1	2	0	
1458	2919	60	3.0	1.671001	2.319681	1	2	0	

1459 rows × 91 columns

In [168]:
```
df2=df3[feature_scale]
```

In [160]:
```
NaN_values
```

Out[160]:
```
['MSZoning', 'Utilities', 'Functional']
```

In [171]:
```
NaN_values = [f for f in df2.columns if df2[f].isnull().sum()>1]
for f in NaN_values:
    df2[f] = np.where(df2[f].isnull()==True,df2[f].median(),df2[f])
```

```
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imp
orts until
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imp
orts until
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imp
orts until
```

In [172]:
```
NaN_values = [f for f in df2.columns if df2[f].isnull().sum()>1]
```

In [174]:
```
df2.head()
```

Out[174]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	20	2.0	1.685370	2.338024	1	2	0	1	1.0
1	20	3.0	1.687642	2.357620	1	2	1	1	1.0
2	60	3.0	1.671001	2.354672	1	2	1	1	1.0
3	60	3.0	1.680725	2.323195	1	2	1	1	1.0
4	120	3.0	1.565317	2.253226	1	2	1	3	1.0

In [175]:
```
df_Test = scaler.transform(df2)
```

In [176]:
```
df_Test
```

Out[176]:
```
array([[ 0.        ,  0.5       , -0.528776  , ...,  0.        ,
         0.        ,  0.        ],
```

```
[ 0.        ,  0.75      , -0.52792134, ...,  0.        ,
  0.        ,  0.        ],
[ 0.23529412,  0.75      , -0.5341814 , ...,  0.        ,
  0.        ,  0.        ],
...,
[ 0.        ,  0.75      , -0.5104711 , ...,  0.        ,
  0.        ,  0.        ],
[ 0.38235294,  0.75      , -0.54672237, ...,  0.        ,
  0.        ,  1.        ],
[ 0.23529412,  0.75      , -0.5341814 , ...,  0.        ,
  0.        ,  0.        ]])
```

In [177]:
```
#Converting transformed Test data into dataframe, and adding on the Id
 variables
data2 = pd.concat([df3[['Id']].reset_index(drop=True),
                   pd.DataFrame(df_Test,columns=feature_scale)],axis=1)
```

In [181]:
```
data2
```

Out[181]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	0.000000	0.50	-0.528776	-2.002693	1.0	1.0	0.000000	0.33333
1	1462	0.000000	0.75	-0.527921	-1.994787	1.0	1.0	0.333333	0.33333
2	1463	0.235294	0.75	-0.534181	-1.995976	1.0	1.0	0.333333	0.33333
3	1464	0.235294	0.75	-0.530523	-2.008677	1.0	1.0	0.333333	0.33333
4	1465	0.588235	0.75	-0.573937	-2.036908	1.0	1.0	0.333333	1.00000
...	...	...	...	...	...	...	...	...	
1454	2915	0.823529	0.25	-0.578778	-2.048762	1.0	1.0	0.000000	0.33333
1455	2916	0.823529	0.25	-0.578778	-2.048762	1.0	1.0	0.000000	0.33333
1456	2917	0.000000	0.75	-0.510471	-1.986621	1.0	1.0	0.000000	0.33333
1457	2918	0.382353	0.75	-0.546722	-2.006888	1.0	1.0	0.000000	0.33333
1458	2919	0.235294	0.75	-0.534181	-2.010094	1.0	1.0	0.000000	0.33333

1459 rows × 83 columns

In [182]:
```
data_transformed = data2
```

In [183]:
```
y_pred=regressor.predict(data2[selected_feat])
```

In [184]:
```
import pickle
filename='finalized_model.pkl'
pickle.dump(classifier,open(filename, 'wb'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-184-5aa639eacc20> in <module>
      1 import pickle
      2 filename='finalized_model.pkl'
----> 3 pickle.dump(classifier,open(filename, 'wb'))

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

In [186]:
```
y_pred
```

Out[186]:
```
array([11.300088, 11.361431, 11.551709, ..., 11.304848, 11.272297,
       11.654555], dtype=float32)
```

In [187]:
```
pred = np.exp(y_pred)
```

In [188]:
```
pred
```

Out[188]:
```
array([ 80828.74,  85942.27, 103954.56, ...,  81214.39,  78613.36,
       115215.01], dtype=float32)
```

In [683]:
```
##Create Sample Submission file and Submit
pred=pd.DataFrame(pred)
sub_df=pd.read_csv('sample_submission.csv')
```

```
datasets=pd.concat([sub_df['Id'],pred],axis=1)
datasets.columns=['Id','SalePrice']
datasets.to_csv('sample_submission_outlier_removed_4.csv',index=False)
```

Random Forest

In [191]:
```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, n
um = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 1000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['mse']}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 200
0], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 120, 23
0, 340, 450, 560, 670, 780, 890, 1000], 'min_samples_split': [2, 5, 10,
14], 'min_samples_leaf': [1, 2, 4, 6, 8], 'criterion': ['mse']}
```

In [192]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf_randomcv = RandomizedSearchCV(estimator=rf,param_distributions=rando
m_grid,n_iter=100,verbose=2,random_state=100,n_jobs=-1)
```

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

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  4.7min
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed: 11.8min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 16.4min finishe
d
C:\Users\Hp\Anaconda3\lib\site-packages\sklearn\model_selection\_sear
ch.py:739: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples,),
for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

Out[192]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='aut
o',
                                                   max_leaf_nodes=Non
e,
                                                   max_samples=None,
                                                   min_impurity_decre
ase=0.0,
                                                   min_impurity_split
=None,
                                                   min_samples_leaf=
1,
                                                   min_samples_split=
2,
                                                   min_weight_fractio
n_leaf=0.0,
                                                   n_estimators=100,
                                                   n_jobs=None, oob_s

core=F
```

```
core=-1...
                   param_distributions={'criterion': ['mse'],
                                        'max_depth': [10, 120, 230, 3
40, 450,
                                                      560, 670, 780,
890,
                                                      1000],
                                        'max_features': ['auto', 'sqr
t',
                                                         'log2'],
                                        'min_samples_leaf': [1, 2, 4,
6, 8],
                                        'min_samples_split': [2, 5, 1
0, 14],
                                        'n_estimators': [200, 400, 60
0, 800,
                                                         1000, 1200,
1400, 1600,
                                                         1800, 200
0]},
                   pre_dispatch='2*n_jobs', random_state=100, refit=T
rue,
                   return_train_score=False, scoring=None, verbose=2)
```

In [193]:
```
rf_randomcv.best_params_
```

Out[193]:
```
{'n_estimators': 1400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 780,
 'criterion': 'mse'}
```

In [194]:
```
rf_randomcv.best_estimator_
```

Out[194]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=780, max_features='sqrt', max_leaf_node
s=None,
                      max_samples=None, min_impurity_decrease=0.0,
```

```
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.
0,
                       n_estimators=1400, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

In [197]:
```
rf_model = RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criteri
on='mse',
                       max_depth=1000, max_features='sqrt', max_leaf_nod
es=None,
                       max_samples=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0
,
                       n_estimators=1400, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

In [198]:
```
rf_model.fit(X_train,y_train)
```

```
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples,), for example using ra
vel().
  """Entry point for launching an IPython kernel.
```

Out[198]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                       max_depth=1000, max_features='sqrt', max_leaf_nod
es=None,
                       max_samples=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.
0,
                       n_estimators=1400, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
```

In [212]:
```
df2_copy = df2
```

In [239]:
```
data_copy = data2
```

In [242]:
```
data2 = data2[selected_feat]
```

In [244]:
```
len(data2.columns)
```

Out[244]:
```
22
```

In [246]:
```
data2['KitchenQual'] = np.where(data2['KitchenQual'].isnull(),data2['KitchenQual'].mode()[0],data2['KitchenQual'])
```

```
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [247]:
```
data2.isnull().sum()
```

Out[247]:
```
LotArea          0
Neighborhood     0
OverallQual      0
YearRemodAdd     0
Foundation       0
BsmtQual         0
BsmtExposure     0
BsmtFinSF1       0
TotalBsmtSF      0
HeatingQC        0
CentralAir       0
1stFlrSF         0
GrLivArea        0
KitchenQual      0
FireplaceQu      0
GarageType       0
GarageFinish     0
GarageCars       0
```

```
GarageCond       0
WoodDeckSF       0
OpenPorchSF      0
SaleCondition    0
dtype: int64
```

In [248]:
```
y_pred_rf = rf_model.predict(data2)
```

In [249]:
```
y_pred_rf
```

Out[249]:
```
array([11.43936272, 11.48522739, 11.7722309 , ..., 11.59621426,
       11.42732151, 11.86304742])
```

In [251]:
```
pred_rf = np.exp(y_pred_rf)
```

In [252]:
```
pred_rf
```

Out[252]:
```
array([ 92907.78462916,  97268.19970787, 129602.96013692, ...,
       108685.56442052,  91795.77136121, 141924.06222125])
```

In [253]:
```
##Create Sample Submission file and Submit
pred=pd.DataFrame(pred_rf)
sub_df=pd.read_csv('sample_submission.csv')
datasets_rf=pd.concat([sub_df['Id'],pred],axis=1)
datasets_rf.columns=['Id','SalePrice']
datasets_rf.to_csv('sample_submission_random_forest.csv',index=False)
```

Artificial Neuron Network Implementation

In [167]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LeakyReLU,PReLU,ELU
from keras.layers import Dropout
```

In [235]:
```
# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(output_dim = 50, init = 'he_uniform',activation='r
elu',input_dim =18))

# Adding the second hidden layer
classifier.add(Dense(output_dim = 50, init = 'he_uniform',activation='r
elu'))

# Adding the third hidden layer
classifier.add(Dense(output_dim = 50, init = 'he_uniform',activation='r
elu'))
# Adding the output layer
classifier.add(Dense(output_dim = 1, init = 'he_uniform'))

# Compiling the ANN
classifier.compile(loss=root_mean_squared_error, optimizer='Adamax')

# Fitting the ANN to the Training set
model=classifier.fit(X_train2[selected_feat].values, y_train.values,val
idation_split=0.20, batch_size = 10, nb_epoch = 100)
```

```
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: UserWa
rning: Update your `Dense` call to the Keras 2 API: `Dense(activation
="relu", input_dim=18, units=50, kernel_initializer="he_uniform")`
  """
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWa
rning: Update your `Dense` call to the Keras 2 API: `Dense(activation
="relu", units=50, kernel_initializer="he_uniform")`

C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:11: UserW
arning: Update your `Dense` call to the Keras 2 API: `Dense(activation
="relu", units=50, kernel_initializer="he_uniform")`
  # This is added back by InteractiveShellApp.init_path()
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:13: UserW
arning: Update your `Dense` call to the Keras 2 API: `Dense(units=1, ke
rnel_initializer="he_uniform")`
  del sys.path[0]
```

```
C:\Users\Hp\Anaconda3\lib\site-packages\ipykernel_launcher.py:19: UserW
arning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
```

```
Train on 1168 samples, validate on 292 samples
Epoch 1/100
1168/1168 [==============================] - 0s 376us/step - loss: 2.
0856 - val_loss: 0.9168
Epoch 2/100
1168/1168 [==============================] - 0s 212us/step - loss: 0.
8518 - val_loss: 0.7471
Epoch 3/100
1168/1168 [==============================] - 0s 236us/step - loss: 0.
6976 - val_loss: 0.6179
Epoch 4/100
1168/1168 [==============================] - 0s 181us/step - loss: 0.
5818 - val_loss: 0.5579
Epoch 5/100
1168/1168 [==============================] - 0s 190us/step - loss: 0.
5083 - val_loss: 0.4413
Epoch 6/100
1168/1168 [==============================] - 0s 225us/step - loss: 0.
4466 - val_loss: 0.4219
Epoch 7/100
1168/1168 [==============================] - 0s 234us/step - loss: 0.
3956 - val_loss: 0.3692
Epoch 8/100
1168/1168 [==============================] - 0s 242us/step - loss: 0.
3478 - val_loss: 0.3261
Epoch 9/100
1168/1168 [==============================] - 0s 189us/step - loss: 0.
3150 - val_loss: 0.3011
Epoch 10/100
1168/1168 [==============================] - 0s 226us/step - loss: 0.
2880 - val_loss: 0.2719
Epoch 11/100
1168/1168 [==============================] - 0s 190us/step - loss: 0.
2797 - val_loss: 0.2710
Epoch 12/100
1168/1168 [==============================] - 0s 224us/step - loss: 0.
2385 - val_loss: 0.2455
```

```
Epoch 13/100
1168/1168 [==============================] - 0s 198us/step - loss: 0.
2570 - val_loss: 0.2356
Epoch 14/100
1168/1168 [==============================] - 0s 199us/step - loss: 0.
2227 - val_loss: 0.2326
Epoch 15/100
1168/1168 [==============================] - 0s 250us/step - loss: 0.
2075 - val_loss: 0.2185
Epoch 16/100
1168/1168 [==============================] - 0s 237us/step - loss: 0.
2049 - val_loss: 0.2178
Epoch 17/100
1168/1168 [==============================] - 0s 229us/step - loss: 0.
1993 - val_loss: 0.2756
Epoch 18/100
1168/1168 [==============================] - 0s 226us/step - loss: 0.
1786 - val_loss: 0.1976
Epoch 19/100
1168/1168 [==============================] - 0s 232us/step - loss: 0.
1821 - val_loss: 0.1895
Epoch 20/100
1168/1168 [==============================] - 0s 183us/step - loss: 0.
1773 - val_loss: 0.2176
Epoch 21/100
1168/1168 [==============================] - 0s 231us/step - loss: 0.
1806 - val_loss: 0.2028
Epoch 22/100
1168/1168 [==============================] - 0s 191us/step - loss: 0.
1870 - val_loss: 0.2719
Epoch 23/100
1168/1168 [==============================] - 0s 223us/step - loss: 0.
1779 - val_loss: 0.1856
Epoch 24/100
1168/1168 [==============================] - 0s 197us/step - loss: 0.
1581 - val_loss: 0.2005
Epoch 25/100
1168/1168 [==============================] - 0s 237us/step - loss: 0.
1674 - val_loss: 0.1793
```

```
Epoch 26/100
1168/1168 [==============================] - 0s 187us/step - loss: 0.
1638 - val_loss: 0.2133
Epoch 27/100
1168/1168 [==============================] - 0s 230us/step - loss: 0.
1644 - val_loss: 0.1894
Epoch 28/100
1168/1168 [==============================] - 0s 242us/step - loss: 0.
1474 - val_loss: 0.1724
Epoch 29/100
1168/1168 [==============================] - 0s 199us/step - loss: 0.
1586 - val_loss: 0.2318
Epoch 30/100
1168/1168 [==============================] - 0s 228us/step - loss: 0.
1747 - val_loss: 0.1772
Epoch 31/100
1168/1168 [==============================] - 0s 184us/step - loss: 0.
1468 - val_loss: 0.1912
Epoch 32/100
1168/1168 [==============================] - 0s 285us/step - loss: 0.
1574 - val_loss: 0.2353
Epoch 33/100
1168/1168 [==============================] - 0s 294us/step - loss: 0.
1490 - val_loss: 0.2075
Epoch 34/100
1168/1168 [==============================] - 0s 284us/step - loss: 0.
1418 - val_loss: 0.1767
Epoch 35/100
1168/1168 [==============================] - 0s 310us/step - loss: 0.
1675 - val_loss: 0.2312
Epoch 36/100
1168/1168 [==============================] - 0s 293us/step - loss: 0.
1496 - val_loss: 0.1742
Epoch 37/100
1168/1168 [==============================] - 0s 298us/step - loss: 0.
1391 - val_loss: 0.1826
Epoch 38/100
1168/1168 [==============================] - 0s 314us/step - loss: 0.
1509 - val_loss: 0.1857
```

```
Epoch 39/100
1168/1168 [==============================] - 0s 310us/step - loss: 0.
1476 - val_loss: 0.2243
Epoch 40/100
1168/1168 [==============================] - 0s 303us/step - loss: 0.
1454 - val_loss: 0.1811
Epoch 41/100
1168/1168 [==============================] - 0s 284us/step - loss: 0.
1391 - val_loss: 0.2497
Epoch 42/100
1168/1168 [==============================] - 0s 293us/step - loss: 0.
1498 - val_loss: 0.1785
Epoch 43/100
1168/1168 [==============================] - 0s 246us/step - loss: 0.
1379 - val_loss: 0.1679
Epoch 44/100
1168/1168 [==============================] - 0s 213us/step - loss: 0.
1435 - val_loss: 0.2048
Epoch 45/100
1168/1168 [==============================] - 0s 226us/step - loss: 0.
1490 - val_loss: 0.1810
Epoch 46/100
1168/1168 [==============================] - 0s 237us/step - loss: 0.
1545 - val_loss: 0.1651
Epoch 47/100
1168/1168 [==============================] - 0s 197us/step - loss: 0.
1389 - val_loss: 0.1740
Epoch 48/100
1168/1168 [==============================] - 0s 231us/step - loss: 0.
1496 - val_loss: 0.1910
Epoch 49/100
1168/1168 [==============================] - 0s 177us/step - loss: 0.
1321 - val_loss: 0.2061
Epoch 50/100
1168/1168 [==============================] - 0s 194us/step - loss: 0.
1467 - val_loss: 0.1683
Epoch 51/100
1168/1168 [==============================] - 0s 254us/step - loss: 0.
1499 - val_loss: 0.1742
```

```
Epoch 52/100
1168/1168 [==============================] - 0s 275us/step - loss: 0.
1361 - val_loss: 0.1697
Epoch 53/100
1168/1168 [==============================] - 0s 187us/step - loss: 0.
1337 - val_loss: 0.1735
Epoch 54/100
1168/1168 [==============================] - 0s 279us/step - loss: 0.
1450 - val_loss: 0.1683
Epoch 55/100
1168/1168 [==============================] - 0s 246us/step - loss: 0.
1320 - val_loss: 0.1990
Epoch 56/100
1168/1168 [==============================] - 0s 242us/step - loss: 0.
1279 - val_loss: 0.1675
Epoch 57/100
1168/1168 [==============================] - 0s 180us/step - loss: 0.
1314 - val_loss: 0.1813
Epoch 58/100
1168/1168 [==============================] - 0s 191us/step - loss: 0.
1441 - val_loss: 0.1750
Epoch 59/100
1168/1168 [==============================] - 0s 199us/step - loss: 0.
1289 - val_loss: 0.2319
Epoch 60/100
1168/1168 [==============================] - 0s 238us/step - loss: 0.
1337 - val_loss: 0.1788
Epoch 61/100
1168/1168 [==============================] - 0s 217us/step - loss: 0.
1345 - val_loss: 0.2189
Epoch 62/100
1168/1168 [==============================] - 0s 240us/step - loss: 0.
1250 - val_loss: 0.1623
Epoch 63/100
1168/1168 [==============================] - 0s 246us/step - loss: 0.
1299 - val_loss: 0.1690
Epoch 64/100
1168/1168 [==============================] - 0s 234us/step - loss: 0.
1255 - val_loss: 0.1658
```

```
Epoch 65/100
1168/1168 [==============================] - 0s 246us/step - loss: 0.
1255 - val_loss: 0.2062
Epoch 66/100
1168/1168 [==============================] - 0s 204us/step - loss: 0.
1397 - val_loss: 0.2213
Epoch 67/100
1168/1168 [==============================] - 0s 267us/step - loss: 0.
1450 - val_loss: 0.1736
Epoch 68/100
1168/1168 [==============================] - 0s 182us/step - loss: 0.
1353 - val_loss: 0.1964
Epoch 69/100
1168/1168 [==============================] - 0s 268us/step - loss: 0.
1261 - val_loss: 0.1772
Epoch 70/100
1168/1168 [==============================] - 0s 217us/step - loss: 0.
1280 - val_loss: 0.1909
Epoch 71/100
1168/1168 [==============================] - 0s 246us/step - loss: 0.
1295 - val_loss: 0.1912
Epoch 72/100
1168/1168 [==============================] - 0s 197us/step - loss: 0.
1325 - val_loss: 0.1709
Epoch 73/100
1168/1168 [==============================] - 0s 257us/step - loss: 0.
1474 - val_loss: 0.2074
Epoch 74/100
1168/1168 [==============================] - 0s 234us/step - loss: 0.
1331 - val_loss: 0.1660
Epoch 75/100
1168/1168 [==============================] - 0s 252us/step - loss: 0.
1326 - val_loss: 0.1885
Epoch 76/100
1168/1168 [==============================] - 0s 209us/step - loss: 0.
1267 - val_loss: 0.1654
Epoch 77/100
1168/1168 [==============================] - 0s 237us/step - loss: 0.
1341 - val_loss: 0.2183
```

```
Epoch 78/100
1168/1168 [==============================] - 0s 187us/step - loss: 0.
1257 - val_loss: 0.1774
Epoch 79/100
1168/1168 [==============================] - 0s 186us/step - loss: 0.
1234 - val_loss: 0.1714
Epoch 80/100
1168/1168 [==============================] - 0s 188us/step - loss: 0.
1225 - val_loss: 0.1939
Epoch 81/100
1168/1168 [==============================] - 0s 175us/step - loss: 0.
1275 - val_loss: 0.1700
Epoch 82/100
1168/1168 [==============================] - 0s 216us/step - loss: 0.
1207 - val_loss: 0.1588
Epoch 83/100
1168/1168 [==============================] - 0s 208us/step - loss: 0.
1273 - val_loss: 0.1668
Epoch 84/100
1168/1168 [==============================] - 0s 236us/step - loss: 0.
1201 - val_loss: 0.1577
Epoch 85/100
1168/1168 [==============================] - 0s 180us/step - loss: 0.
1178 - val_loss: 0.1766
Epoch 86/100
1168/1168 [==============================] - 0s 199us/step - loss: 0.
1237 - val_loss: 0.1676
Epoch 87/100
1168/1168 [==============================] - 0s 223us/step - loss: 0.
1362 - val_loss: 0.1808
Epoch 88/100
1168/1168 [==============================] - 0s 220us/step - loss: 0.
1147 - val_loss: 0.1658
Epoch 89/100
1168/1168 [==============================] - 0s 274us/step - loss: 0.
1253 - val_loss: 0.2225
Epoch 90/100
1168/1168 [==============================] - 0s 298us/step - loss: 0.
1217 - val_loss: 0.1904
```

```
Epoch 91/100
1168/1168 [==============================] - 0s 297us/step - loss: 0.
1214 - val_loss: 0.1718
Epoch 92/100
1168/1168 [==============================] - 0s 296us/step - loss: 0.
1244 - val_loss: 0.1548
Epoch 93/100
1168/1168 [==============================] - 0s 299us/step - loss: 0.
1221 - val_loss: 0.1591
Epoch 94/100
1168/1168 [==============================] - 0s 323us/step - loss: 0.
1156 - val_loss: 0.1645
Epoch 95/100
1168/1168 [==============================] - 0s 295us/step - loss: 0.
1190 - val_loss: 0.1709
Epoch 96/100
1168/1168 [==============================] - 0s 287us/step - loss: 0.
1318 - val_loss: 0.1668
Epoch 97/100
1168/1168 [==============================] - 0s 327us/step - loss: 0.
1253 - val_loss: 0.1876
Epoch 98/100
1168/1168 [==============================] - 0s 305us/step - loss: 0.
1219 - val_loss: 0.1695
Epoch 99/100
1168/1168 [==============================] - 0s 293us/step - loss: 0.
1282 - val_loss: 0.1962
Epoch 100/100
1168/1168 [==============================] - 0s 257us/step - loss: 0.
1266 - val_loss: 0.1569
```

In [171]:
```
from keras import backend as K
def root_mean_squared_error(y_true, y_pred):
        return K.sqrt(K.mean(K.square(y_pred - y_true)))
```

In [178]:
```
X_train2.shape
```

Out[178]:
```
(1460, 82)
```

In [175]:
```
y_train.shape
```

Out[175]:
```
(1460, 1)
```

In [233]:
```
data2[selected_feat].shape
```

Out[233]:
```
(1459, 18)
```

In [177]:
```
X_train2=dataset.drop(['Id','SalePrice'],axis=1)
```

In [236]:
```
ann_pred=classifier.predict(data2[selected_feat])#.iloc[:,1:].values)
```

In [217]:
```
data2.isnull()==True
```

Out[217]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	..
1454	False	False	False	False	False	False	False	False	Fals
1455	False	False	False	False	False	False	False	False	Fals
1456	False	False	False	False	False	False	False	False	Fals
1457	False	False	False	False	False	False	False	False	Fals
1458	False	False	False	False	False	False	False	False	Fals

1459 rows × 83 columns

In [212]:
```
data2
```

Out[212]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
0	1461	0.000000	0.50	0.495064	0.428726	1.0	1.0	0.000000	0.33333
1	1462	0.000000	0.75	0.499662	0.468857	1.0	1.0	0.333333	0.33333
2	1463	0.235294	0.75	0.466207	0.462769	1.0	1.0	0.333333	0.33333
3	1464	0.235294	0.75	0.485693	0.398875	1.0	1.0	0.333333	0.33333
4	1465	0.588235	0.75	0.265271	0.263841	1.0	1.0	0.333333	1.00000
...	...	...	...	...	...	...	...	...	.
1454	2915	0.823529	0.25	0.000000	0.077946	1.0	1.0	0.000000	0.33333
1455	2916	0.823529	0.25	0.000000	0.073654	1.0	1.0	0.000000	0.33333
1456	2917	0.000000	0.75	0.751625	0.534967	1.0	1.0	0.000000	0.33333
1457	2918	0.382353	0.75	0.400718	0.407753	1.0	1.0	0.000000	0.33333
1458	2919	0.235294	0.75	0.466207	0.391866	1.0	1.0	0.000000	0.33333

1459 rows × 83 columns

In [231]:
```
data2[data2['MSZoning'].isnull()==True]
```

Out[231]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContou
455	1916	0.058824	NaN	0.609556	0.551654	0.0	1.0	0.000000	0.33333
756	2217	0.000000	NaN	0.495064	0.473158	1.0	1.0	0.000000	0.66666
790	2251	0.294118	NaN	0.429425	0.738567	1.0	1.0	0.333333	0.66666
1444	2905	0.000000	NaN	0.660252	0.622313	1.0	1.0	0.000000	0.33333

In [237]:
```
ann_pred
```

Out[237]:
```
array([[11.637968 ],
       [11.928009 ],
       [12.033077 ],
       ...,
       [11.865556 ],
       [11.660313 ],
       [12.2346115]], dtype=float32)
```

In [238]:
```
np.exp(ann_pred)
```

Out[238]:
```
array([[113319.67],
       [151449.73],
       [168228.3 ],
       ...,
       [142280.5 ],
       [115880.26],
       [205790.  ]], dtype=float32)
```

In [240]:
```
##Create Sample Submission file and Submit using ANN
pred_ann=pd.DataFrame(np.exp(ann_pred))
sub_df=pd.read_csv('sample_submission.csv')
datasets=pd.concat([sub_df['Id'],pred_ann],axis=1)
datasets.columns=['Id','SalePrice']
datasets.to_csv('sample_submission_ann2.csv',index=False)
```

In [197]:
```
pred_ann.isnull().sum()
```

Out[197]:
```
0    9
dtype: int64
```

In [193]:
```
pred
```

Out[193]:

	0
0	119480.039062
1	137401.000000

	0
2	169420.203125
3	178905.687500
4	179471.828125
...	...
1454	79608.859375
1455	71195.601562
1456	139554.359375
1457	109234.429688
1458	208175.281250

1459 rows × 1 columns

In [ ]:
```
##Create Sample Submission file and Submit
pred=pd.DataFrame(pred)
sub_df=pd.read_csv('sample_submission.csv')
datasets=pd.concat([sub_df['Id'],pred],axis=1)
datasets.columns=['Id','SalePrice']
datasets.to_csv('sample_submission_3.csv',index=False)
```