In [938]:
```
# Loading the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0,8.0)
import seaborn as sns
from scipy import stats
from scipy.stats import norm

import sklearn as sklearn

from sklearn import linear_model
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn import model_selection
import sklearn.feature_selection as fs
from sklearn import tree

from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score

from sklearn import svm
from sklearn import metrics
import xgboost as xgboost
from xgboost import XGBRegressor
```

In [939]:
```
#Loading the dataset

train = pd.read_csv("train.csv")
test  = pd.read_csv("test.csv")

train.head(10)
```

Out[939]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandConto
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl
5	6	50	RL	85.0	14115	Pave	NaN	IR1	Lvl
6	7	20	RL	75.0	10084	Pave	NaN	Reg	Lvl
7	8	60	RL	NaN	10382	Pave	NaN	IR1	Lvl
8	9	50	RM	51.0	6120	Pave	NaN	Reg	Lvl
9	10	190	RL	50.0	7420	Pave	NaN	Reg	Lvl

10 rows × 81 columns

In [940]:
```
print(' No of rows and columns in train data are: {0} rows and {1} columns'.fo
rmat(train.shape[0], train.shape[1]))

print(' No of rows and columns in test data are: {0} rows and {1} columns'.for
mat(test.shape[0], test.shape[1]))
```

```
 No of rows and columns in train data are: 1460 rows and 81 columns
 No of rows and columns in test data are: 1459 rows and 80 columns
```

Data Exploration

In [941]:
```
# Lets see the structure of data
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
```

```
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

Checking for Missing values in the dataset:

In [942]:
```
# Checking columns with missing values
train.columns[train.isnull().any()]
```

Out[942]:
```
Index(['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Electrical', 'FireplaceQu', 'GarageType', 'GarageYrBlt',
       'GarageFinish', 'GarageQual', 'GarageCond', 'PoolQC', 'Fence',
       'MiscFeature'],
      dtype='object')
```

In [943]:
```
# Checking count of missing values
train.isnull().sum()
```

```
Out[943]: Id                 0
          MSSubClass         0
          MSZoning           0
          LotFrontage      259
          LotArea            0
          Street             0
          Alley           1369
          LotShape           0
          LandContour        0
          Utilities          0
          LotConfig          0
          LandSlope          0
          Neighborhood       0
          Condition1         0
          Condition2         0
          BldgType           0
          HouseStyle         0
          OverallQual        0
          OverallCond        0
          YearBuilt          0
          YearRemodAdd       0
          RoofStyle          0
          RoofMatl           0
          Exterior1st        0
          Exterior2nd        0
          MasVnrType         8
          MasVnrArea         8
          ExterQual          0
          ExterCond          0
          Foundation         0
                          ...
          BedroomAbvGr       0
          KitchenAbvGr       0
          KitchenQual        0
          TotRmsAbvGrd       0
          Functional         0
          Fireplaces         0
          FireplaceQu      690
          GarageType        81
          GarageYrBlt       81
          GarageFinish      81
          GarageCars         0
          GarageArea         0
          GarageQual        81
          GarageCond        81
          PavedDrive         0
          WoodDeckSF         0
          OpenPorchSF        0
          EnclosedPorch      0
          3SsnPorch          0
          ScreenPorch        0
          PoolArea           0
          PoolQC          1453
          Fence           1179
          MiscFeature     1406
          MiscVal            0
          MoSold             0
```

```
YrSold             0
SaleType           0
SaleCondition      0
SalePrice          0
Length: 81, dtype: int64
```

So, from a total of 81 attributes, 19 have missing values. Let us check the percentage of missing values in these attributes:

In [944]:
```
# Missing value counts
missing = train.isnull().sum()/len(train)*100
missing = missing[missing>0]
missing.sort_values(inplace = True)
missing
```

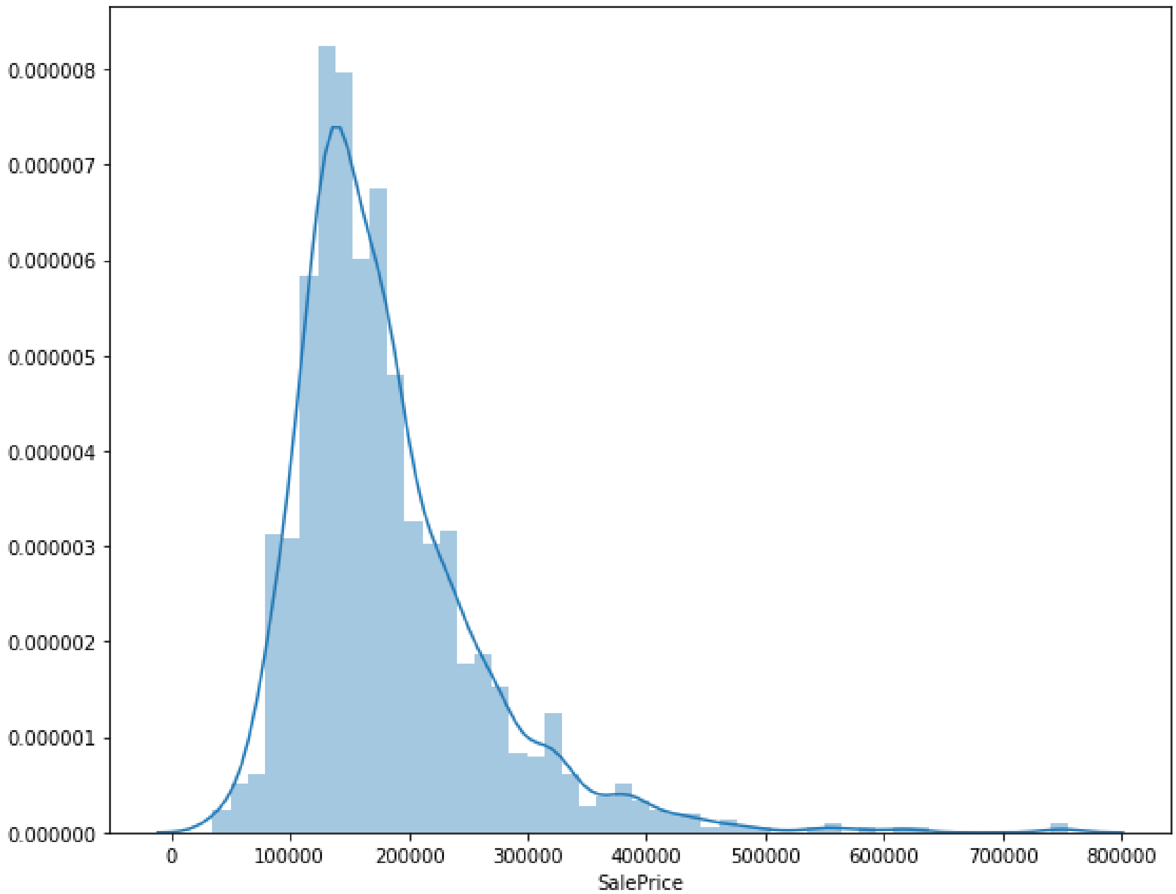
Out[944]:
```
Electrical       0.068493
MasVnrType       0.547945
MasVnrArea       0.547945
BsmtQual         2.534247
BsmtCond         2.534247
BsmtFinType1     2.534247
BsmtExposure     2.602740
BsmtFinType2     2.602740
GarageCond       5.547945
GarageQual       5.547945
GarageFinish     5.547945
GarageType       5.547945
GarageYrBlt      5.547945
LotFrontage     17.739726
FireplaceQu     47.260274
Fence           80.753425
Alley           93.767123
MiscFeature     96.301370
PoolQC          99.520548
dtype: float64
```

So, the variable with highest number of missing value is 'PoolQC' with 99.5% missing values followed by Miscfeature(96.3%), Alley(93.7%) and Fence(80.7%).

Lets check for the distribution of the response variable (SalePrice):

In [945]:
```
sns.distplot(train['SalePrice'])
```

Out[945]: <matplotlib.axes._subplots.AxesSubplot at 0x25b93bdfe10>



Here, we observe that the target variable 'SalePrice' is slightly skewed to the right. Target variables which have normal distribution helps in better modeling relationship between the dependent and independent variables. Hence we intend to transform the target variable. We can adopt log transformation to get rid of the skewness.

In [946]:
```
# Checking skewness
train['SalePrice'].skew()
```

Out[946]: 1.8828757597682129

Let us try to transform the target variable by applying log transformation:

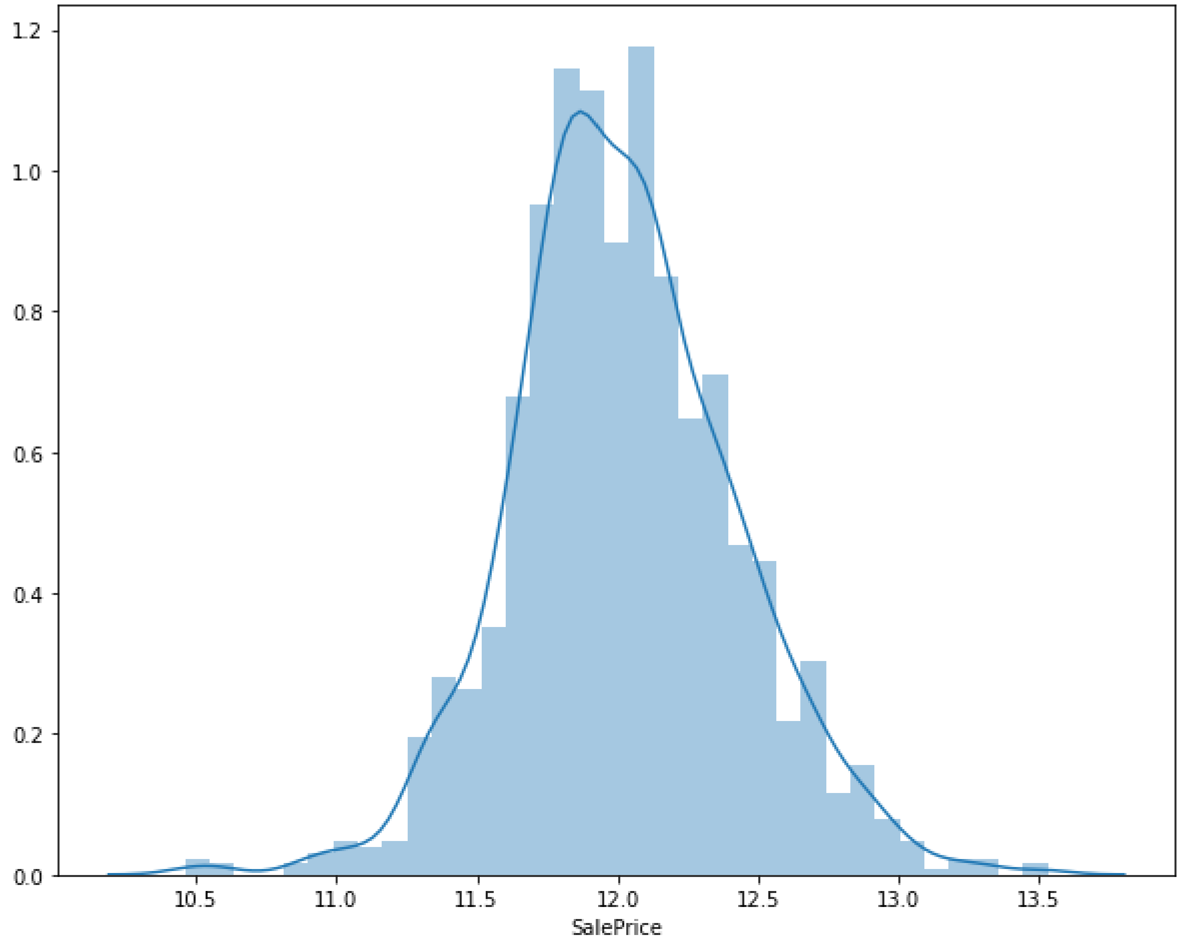
In [947]:
```
# Log tranforming the target variable
target = np.log(train['SalePrice'])
target.skew()
```

Out[947]: 0.12133506220520406

Thus, we observe that the skewness has considerably reduced. Now we plot and check the distribution of the transformed variable.

In [948]:
```
# Plotting the transformed variable
sns.distplot(target)
```

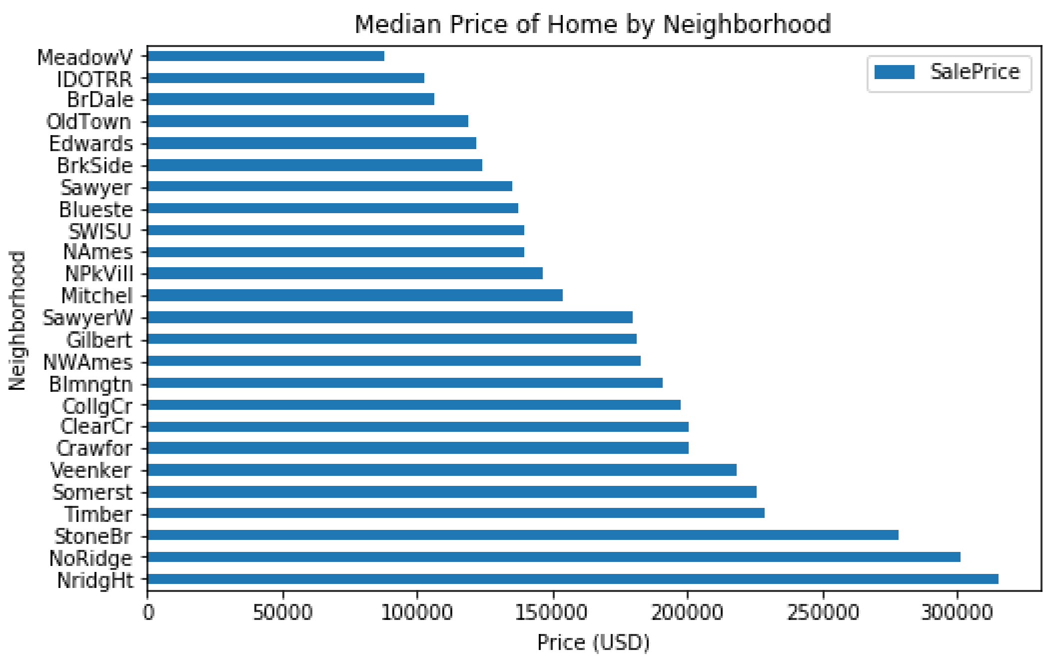
Out[948]: <matplotlib.axes._subplots.AxesSubplot at 0x25b93ccdc88>



Thus, we observe that log transformation has quite fixed the skewness to get a normal distribution.

In [949]:
```
# plot median price by neighborhood
a = pd.DataFrame(train.groupby('Neighborhood')['SalePrice'].median().sort_values(ascending = False))
a.plot.barh(figsize = (8,5))
plt.xlabel('Price (USD)')
plt.title('Median Price of Home by Neighborhood')
```

Out[949]: Text(0.5,1,'Median Price of Home by Neighborhood')



Also, lets seperate the categorical variable from the numeric variable for effective visualization.

In [950]:
```
# seperating categorical and numeric variables
numeric_data = train.select_dtypes(include=[np.number])
cat_data = train.select_dtypes(exclude=[np.number])

print("There are {} numeric and {} categorical columns in train data".format(numeric_data.shape[1], cat_data.shape[1]))
```

```
There are 38 numeric and 43 categorical columns in train data
```

We do not need the ID variable and so we can delete it as it does not make much sense.

In [951]:
```
del numeric_data['Id']
```

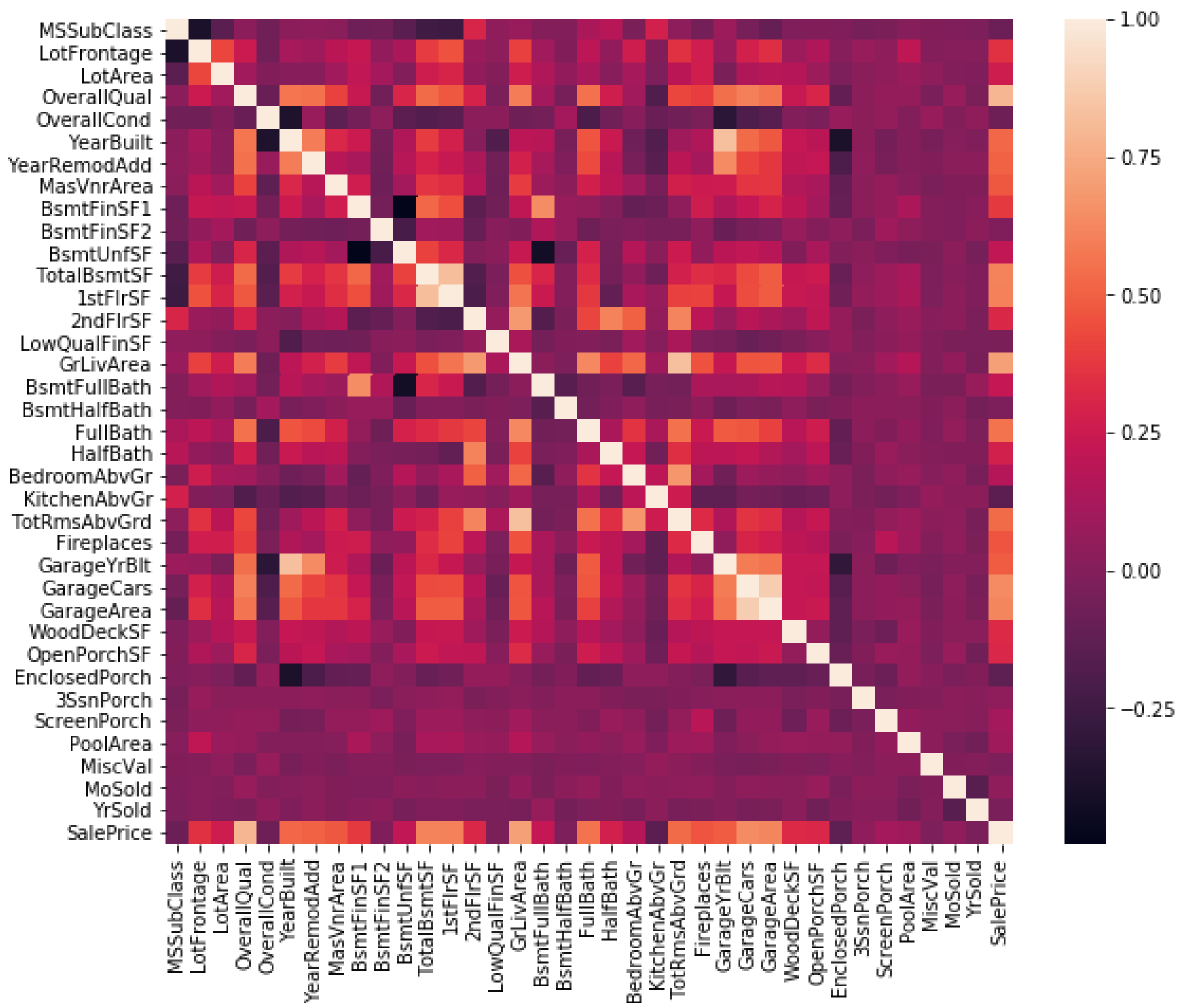
Now, since we have divided the dataset into categorical and numerical data points, we can now check for correlation in the numeric_data:

Let us try to check the correlation of the independent variables with the target variable. By this way, we can eliminate the variables which have a very low correlation with the target variable and thereby reduce the total number of variables to be considered for modeling.

Correlation analysis plot:

In [953]:
```
# Correlation plot
corr = numeric_data.corr()
sns.heatmap(corr)
```

Out[953]: <matplotlib.axes._subplots.AxesSubplot at 0x25b9e3c1e10>



The last row has 'SalePrice' which can be compared with correlations of other variables. Let us try to check the numeric correlation values of variables with 'SalePrice'

In [954]:
```
corr['SalePrice'].sort_values(ascending= False)
```

Out[954]:
```
SalePrice        1.000000
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      0.613581
1stFlrSF         0.605852
FullBath         0.560664
TotRmsAbvGrd     0.533723
YearBuilt        0.522897
YearRemodAdd     0.507101
GarageYrBlt      0.486362
MasVnrArea       0.477493
Fireplaces       0.466929
BsmtFinSF1       0.386420
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
BsmtFullBath     0.227122
BsmtUnfSF        0.214479
BedroomAbvGr     0.168213
ScreenPorch      0.111447
PoolArea         0.092404
MoSold           0.046432
3SsnPorch        0.044584
BsmtFinSF2      -0.011378
BsmtHalfBath    -0.016844
MiscVal         -0.021190
LowQualFinSF    -0.025606
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

In [1005]:
```
print (corr['SalePrice'].sort_values(ascending=False)[:5], '\n') #top 5 values
print ('----------------------')
print (corr['SalePrice'].sort_values(ascending=False)[-5:]) #last 5 values`
```

```
SalePrice      1.000000
OverallQual    0.790982
GrLivArea      0.708624
GarageCars     0.640409
GarageArea     0.623431
Name: SalePrice, dtype: float64

----------------------
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

Let us visualize some of the highly correlated variables:

The variable 'Overallqual' is 79% correlated with the target variable. Next is GrLivArea which is 70% correlated. The high correlation of thest two variables makes practical sense as well.

Let us further try to visualize some of these highly corrlated variables:

In [955]:
```
train['OverallQual'].unique()
```

Out[955]:
```
array([ 7,  6,  8,  5,  9,  4, 10,  3,  1,  2], dtype=int64)
```

This variable seems to be rated on a scale of 1-10 and therefore is an ordinal variable. Lets further explore this variable by checking the median sale price of houses wrt OverallQual.(We use median since our target variable was found to be skewed which contains outliers and medians are robust to outliers.)

In [956]:

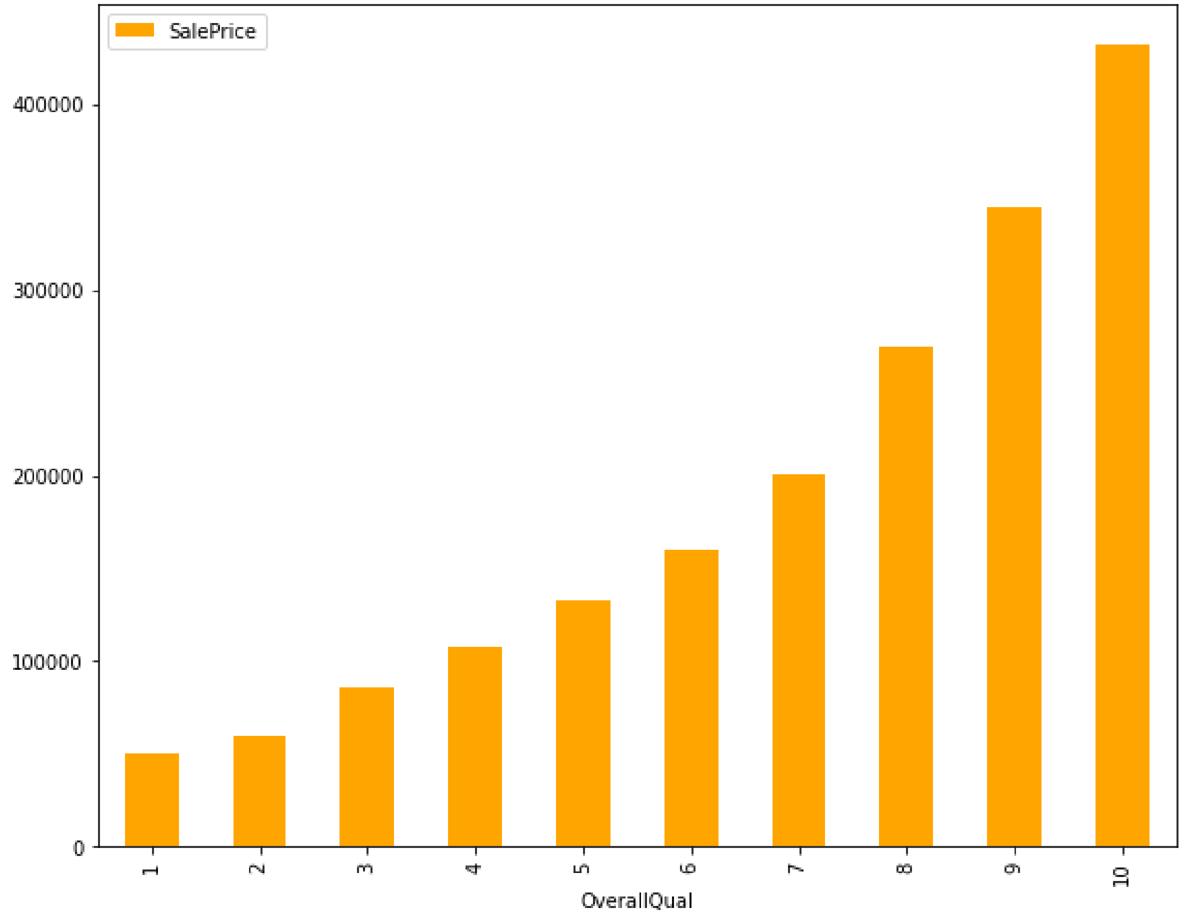
```
# Creating aggregated tables using Pandas pivot table
# Plotting median price per quality level
pivot = train.pivot_table(index='OverallQual', values='SalePrice', aggfunc=np.
median)
pivot
```

Out[956]:

	SalePrice
OverallQual	
1	50150
2	60000
3	86250
4	108000
5	133000
6	160000
7	200141
8	269750
9	345000
10	432390

In [957]:
```
# Plotting the table to observe Median SalePrices
pivot.plot(kind='bar', color='orange')
```

Out[957]: <matplotlib.axes._subplots.AxesSubplot at 0x25b9fdda1d0>

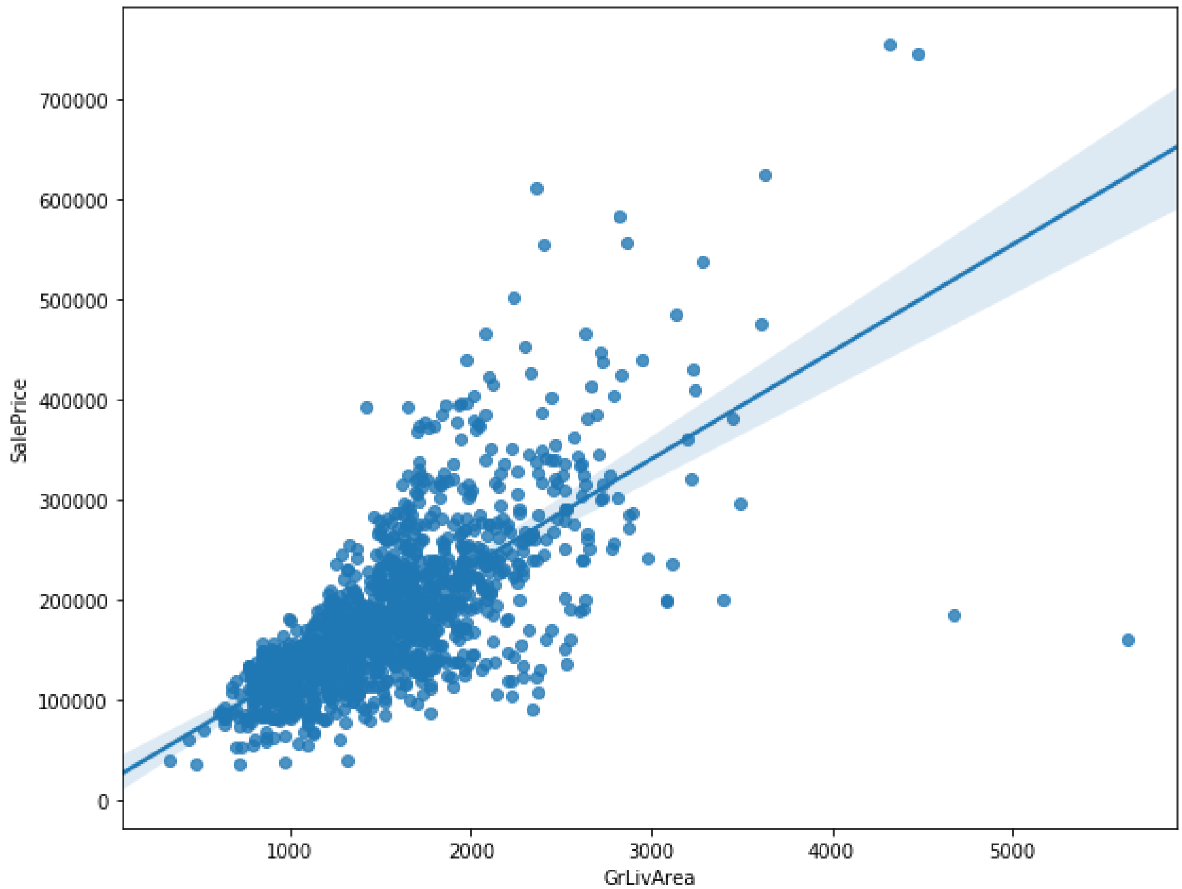


Thus, understandably, we observe that as the Overall Quality increases, the Median Sale price also rises.

Lets visualize the next highly correlated variable GrLivArea:

In [958]:
```
# Plotting GrLivArea
sns.regplot(x=train['GrLivArea'], y= train['SalePrice'])
```

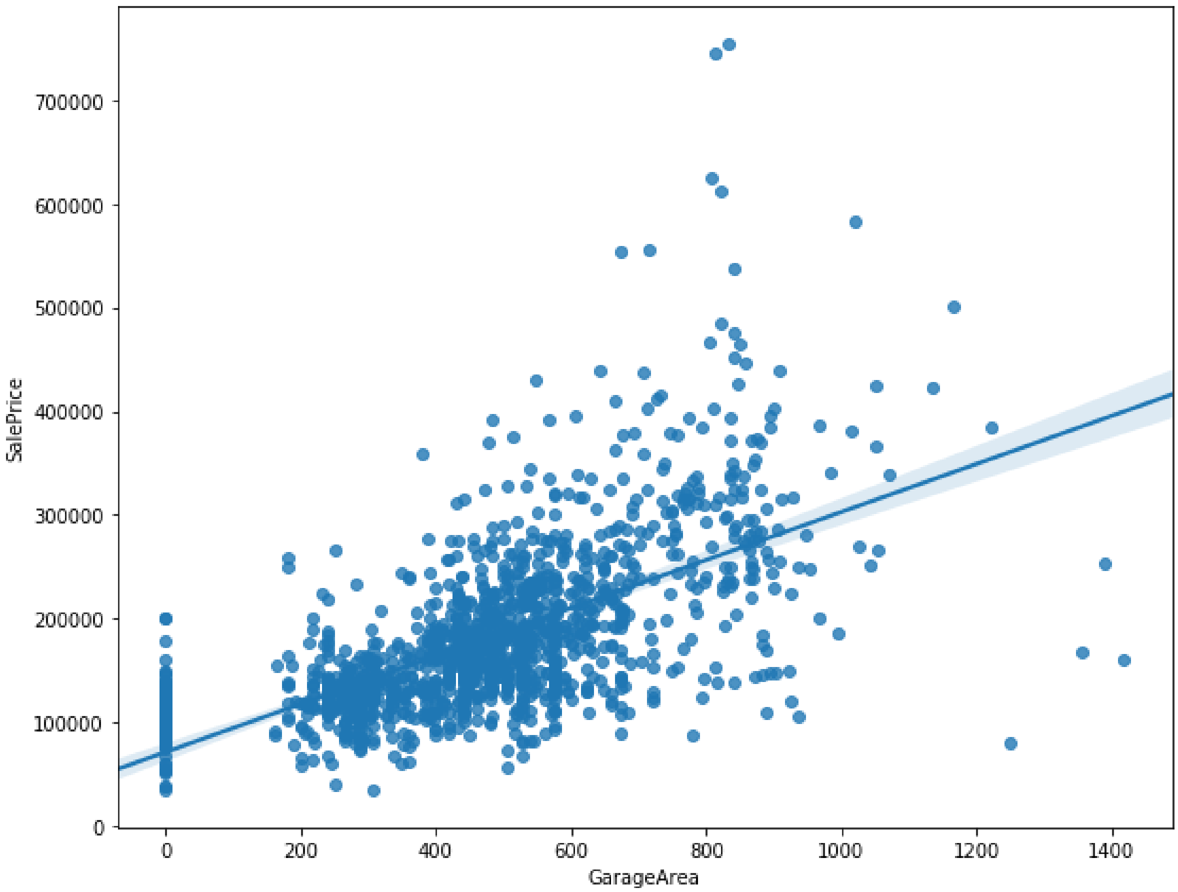
Out[958]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba2875128>



Thus, we observe that Sale Price and GrLivArea have a direct correlation. An oulier can be observed at GrLivArea > 4000 and therefore needs to be eliminated as such ouliers lower a model's performance.

In [959]:
```
sns.regplot(x=train['GarageArea'], y=train['SalePrice'])
```

Out[959]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba29d9cc0>



Similarly, for GarageArea also we can observe a direct correlation indicating higher the Garage Area higher is the Sale Price.

Let us now visualize the categorical variables:

In [960]:
```
cat_data.describe()
```

Out[960]:

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

4 rows × 43 columns

Let us check the median sale price of a house based on its Sale Condition:

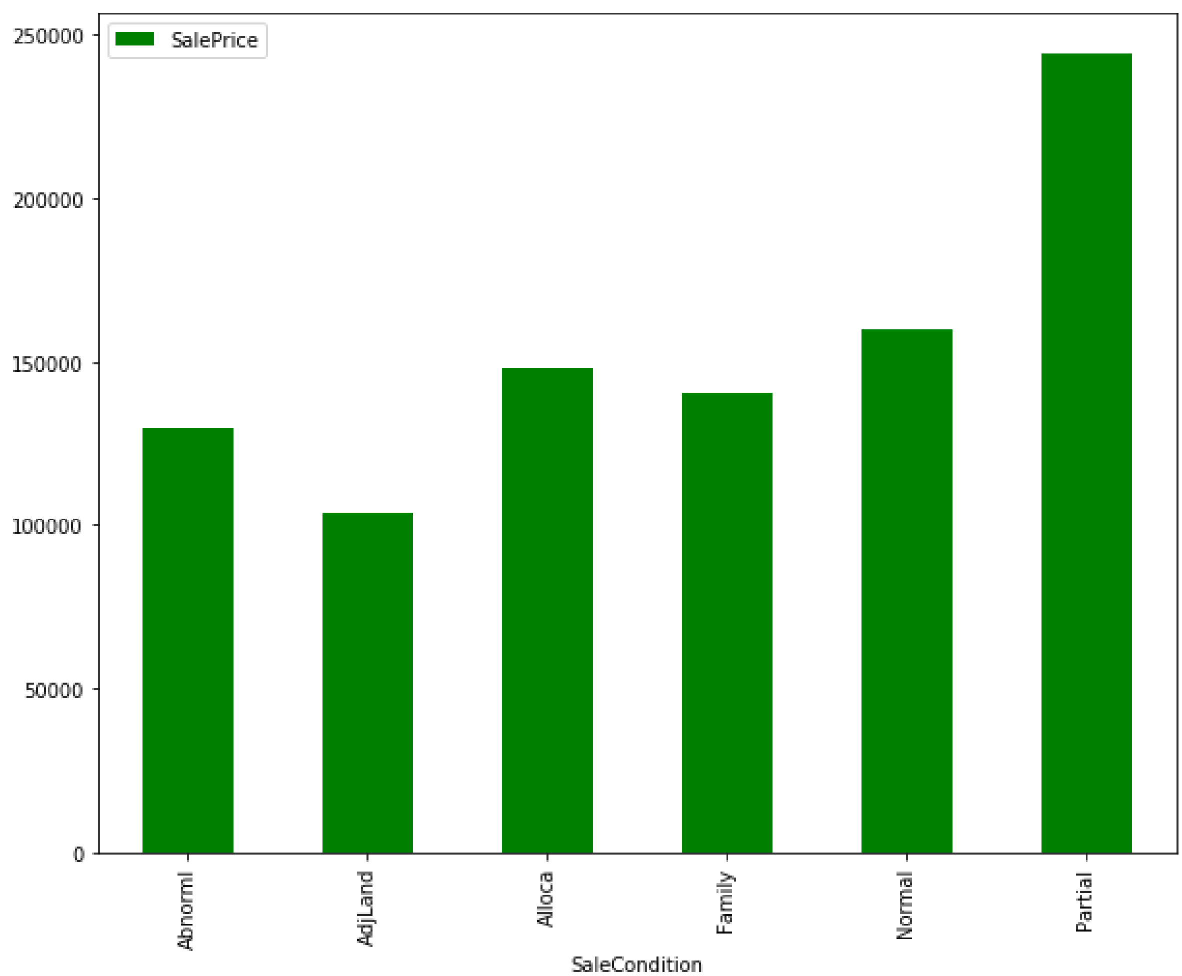
In [961]:
```
sp_pivot = train.pivot_table(index="SaleCondition", values="SalePrice", aggfun
c=np.median)
sp_pivot
```

Out[961]:

	SalePrice
SaleCondition	
Abnorml	130000
AdjLand	104000
Alloca	148145
Family	140500
Normal	160000
Partial	244600

In [962]:
```
sp_pivot.plot(kind="bar", color="green")
```

Out[962]: <matplotlib.axes._subplots.AxesSubplot at 0x25ba3c94240>

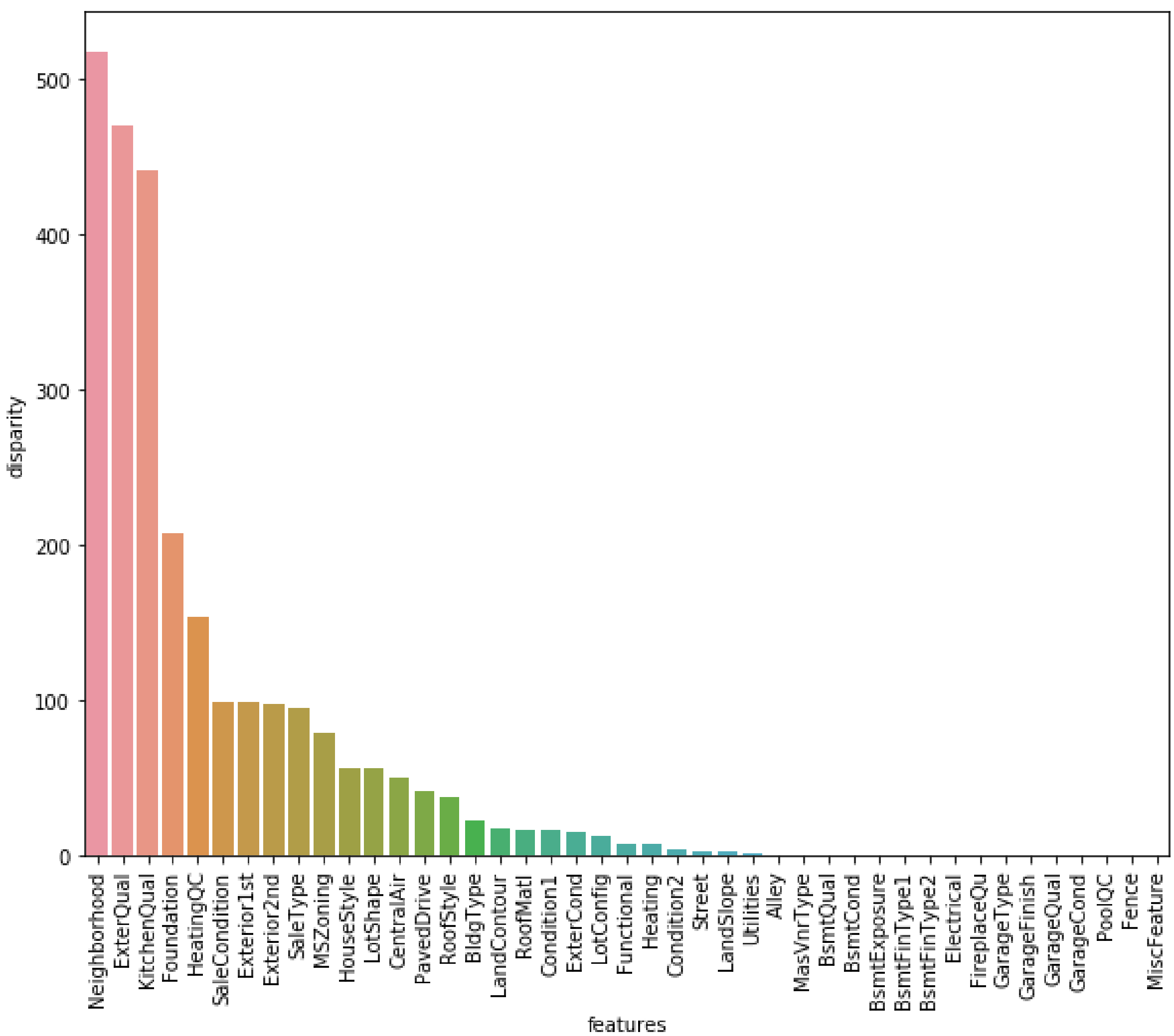


We observe that Sale Condition 'Partial' fetches the highest Median Price.

Let's define a function which calculates p values. From those p values, we'll calculate a disparity score. Higher the disparity score, better the feature in predicting sale price.

In [963]:
```
cat = [f for f in train.columns if train.dtypes[f] == 'object']
def anova(frame):
    anv = pd.DataFrame()
    anv['features'] = cat
    pvals = []
    for c in cat:
           samples = []
           for cls in frame[c].unique():
                  s = frame[frame[c] == cls]['SalePrice'].values
                  samples.append(s)
           pval = stats.f_oneway(*samples)[1]
           pvals.append(pval)
    anv['pval'] = pvals
    return anv.sort_values('pval')

cat_data['SalePrice'] = train.SalePrice.values
k = anova(cat_data)
k['disparity'] = np.log(1./k['pval'].values)
sns.barplot(data=k, x = 'features', y='disparity')
plt.xticks(rotation=90)
plt
```

Out[963]:
```
<module 'matplotlib.pyplot' from 'C:\\ProgramData\\Anaconda3\\lib\\site-packa
ges\\matplotlib\\pyplot.py'>
```

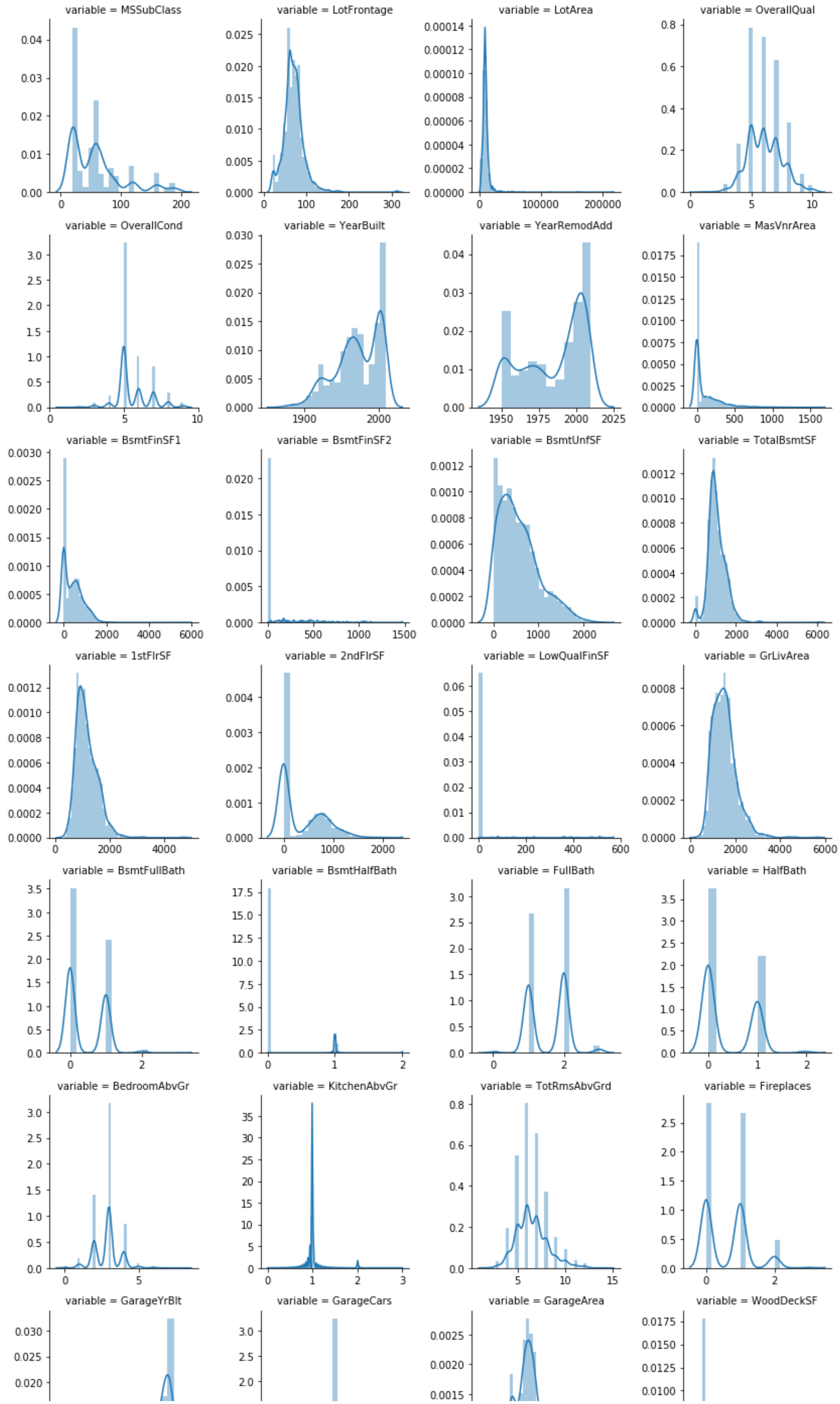


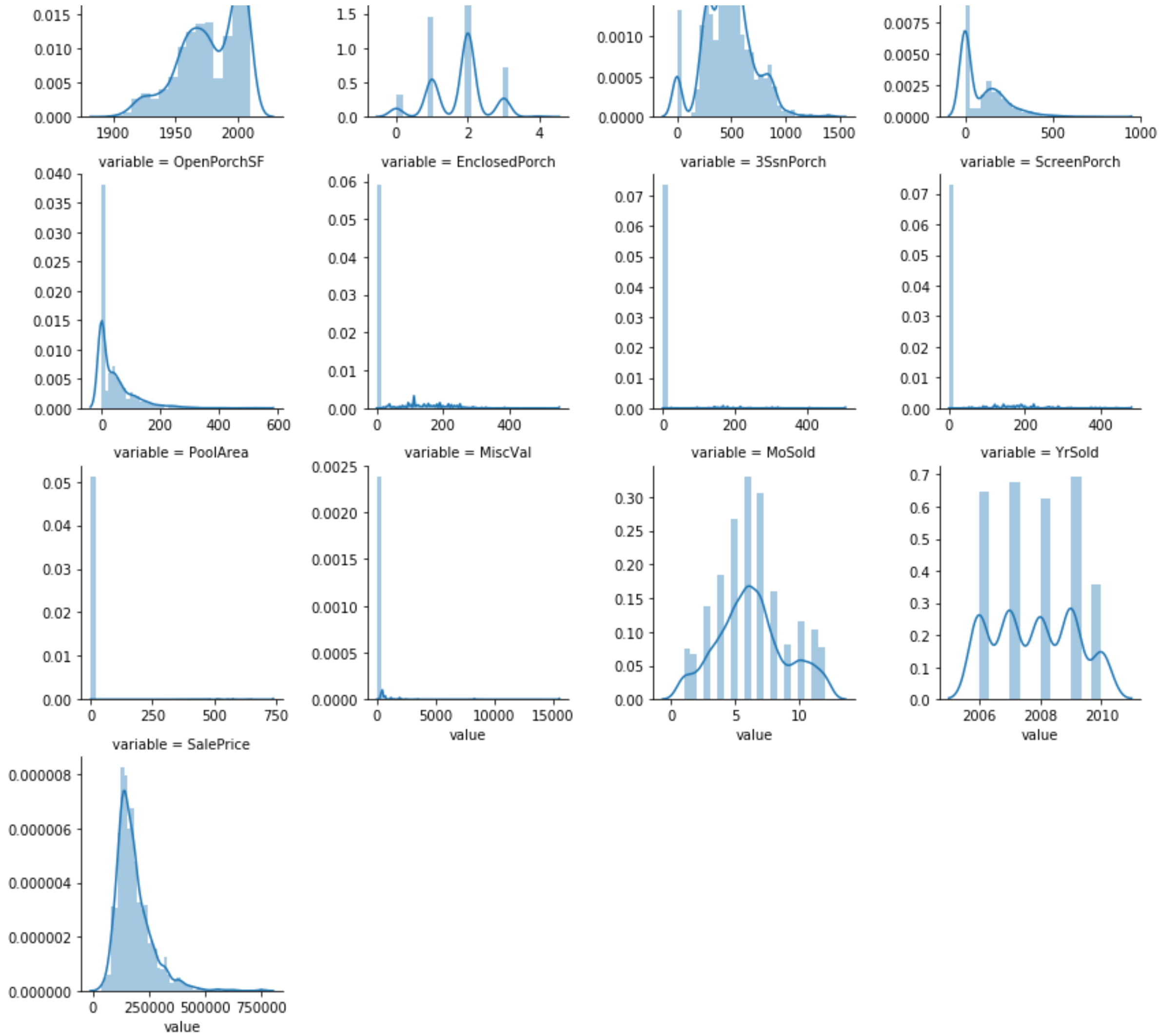
Among all categorical variables, Neighborhood turned out to be the most important feature followed by ExterQual, KitchenQual, etc. It means that people also consider the goodness of the neighborhood, the quality of the kitchen, the quality of the material used on the exterior walls etc prior to purchasing a house

Now, let us plot Histograms for the dependent variables and check for skewness

In [964]:
```
# Create numeric data histograms

num = [f for f in train.columns if train.dtypes[f] != 'object']
num.remove('Id')
nd = pd.melt(train, value_vars = num)
n1 = sns.FacetGrid (nd, col='variable', col_wrap=4, sharex=False, sharey = Fal
se)
n1 = n1.map(sns.distplot, 'value')
n1
```

```
Out[964]: <seaborn.axisgrid.FacetGrid at 0x25ba3cdc8d0>
```

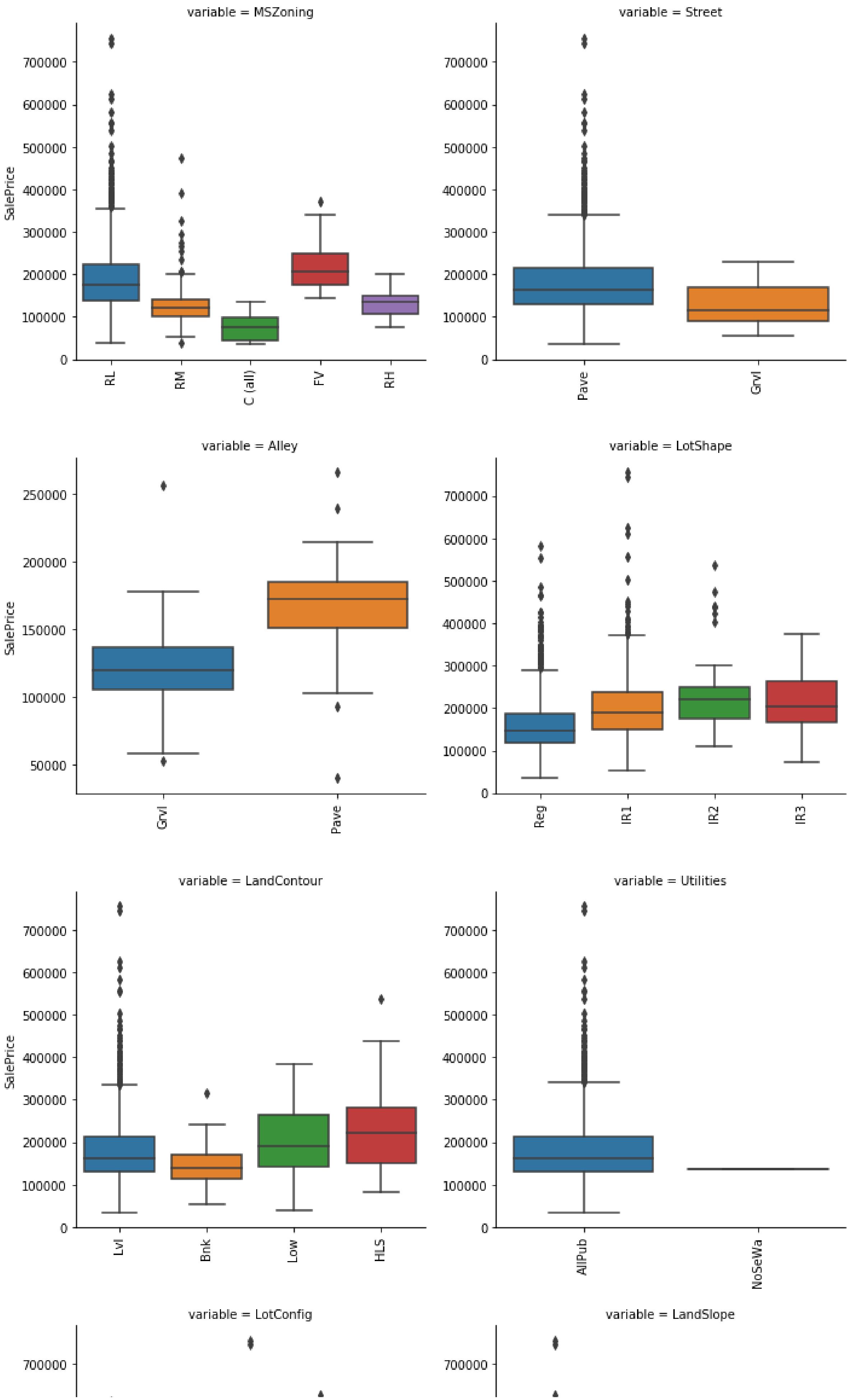

variable = MSSubClass
variable = LotFrontage
variable = LotArea
variable = OverallQual
variable = OverallCond
variable = YearBuilt
variable = YearRemodAdd
variable = MasVnrArea
variable = BsmtFinSF1
variable = BsmtFinSF2
variable = BsmtUnfSF
variable = TotalBsmtSF
variable = 1stFlrSF
variable = 2ndFlrSF
variable = LowQualFinSF
variable = GrLivArea
variable = BsmtFullBath
variable = BsmtHalfBath
variable = FullBath
variable = HalfBath
variable = BedroomAbvGr
variable = KitchenAbvGr
variable = TotRmsAbvGrd
variable = Fireplaces
variable = GarageYrBlt
variable = GarageCars
variable = GarageArea
variable = WoodDeckSF

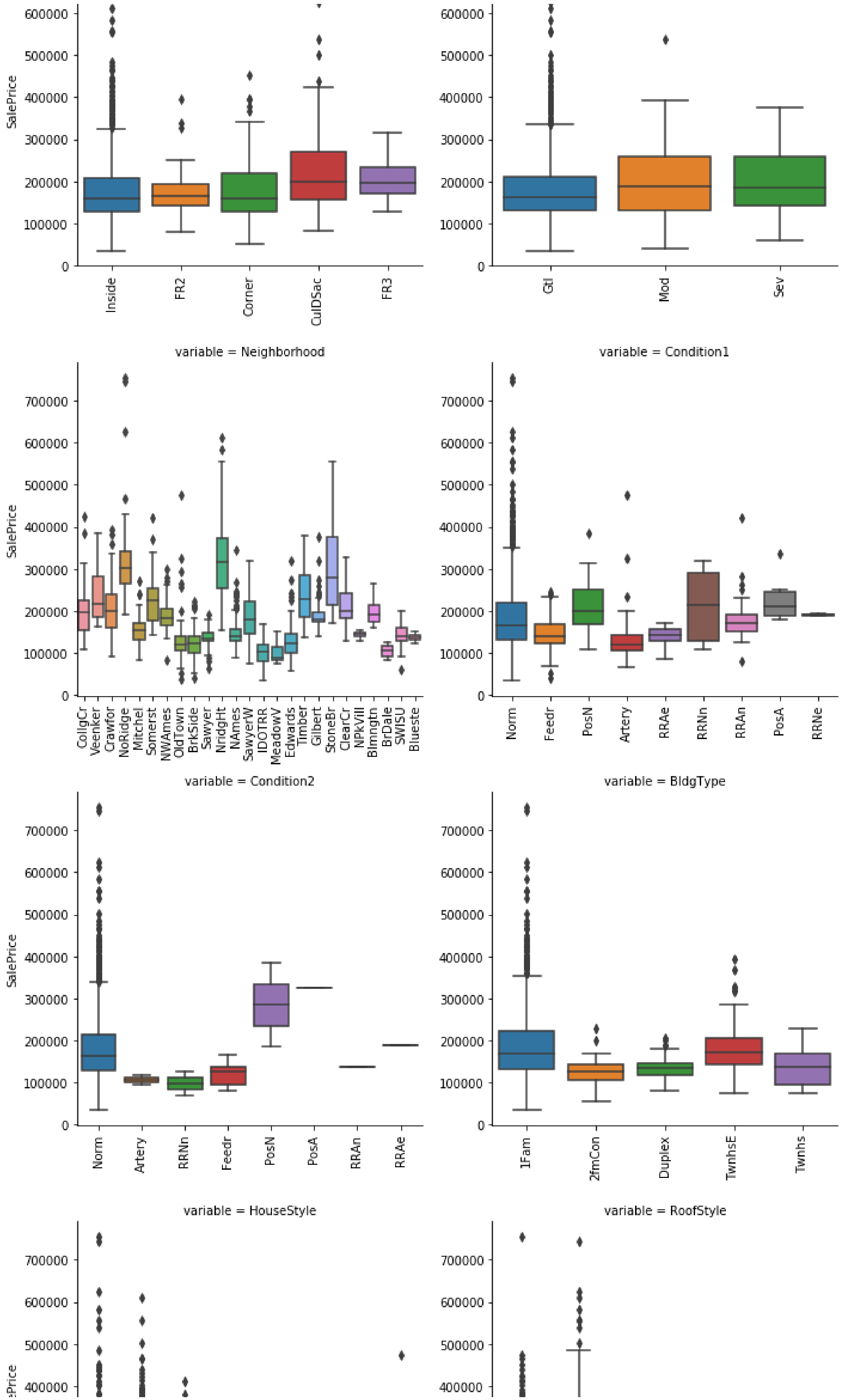


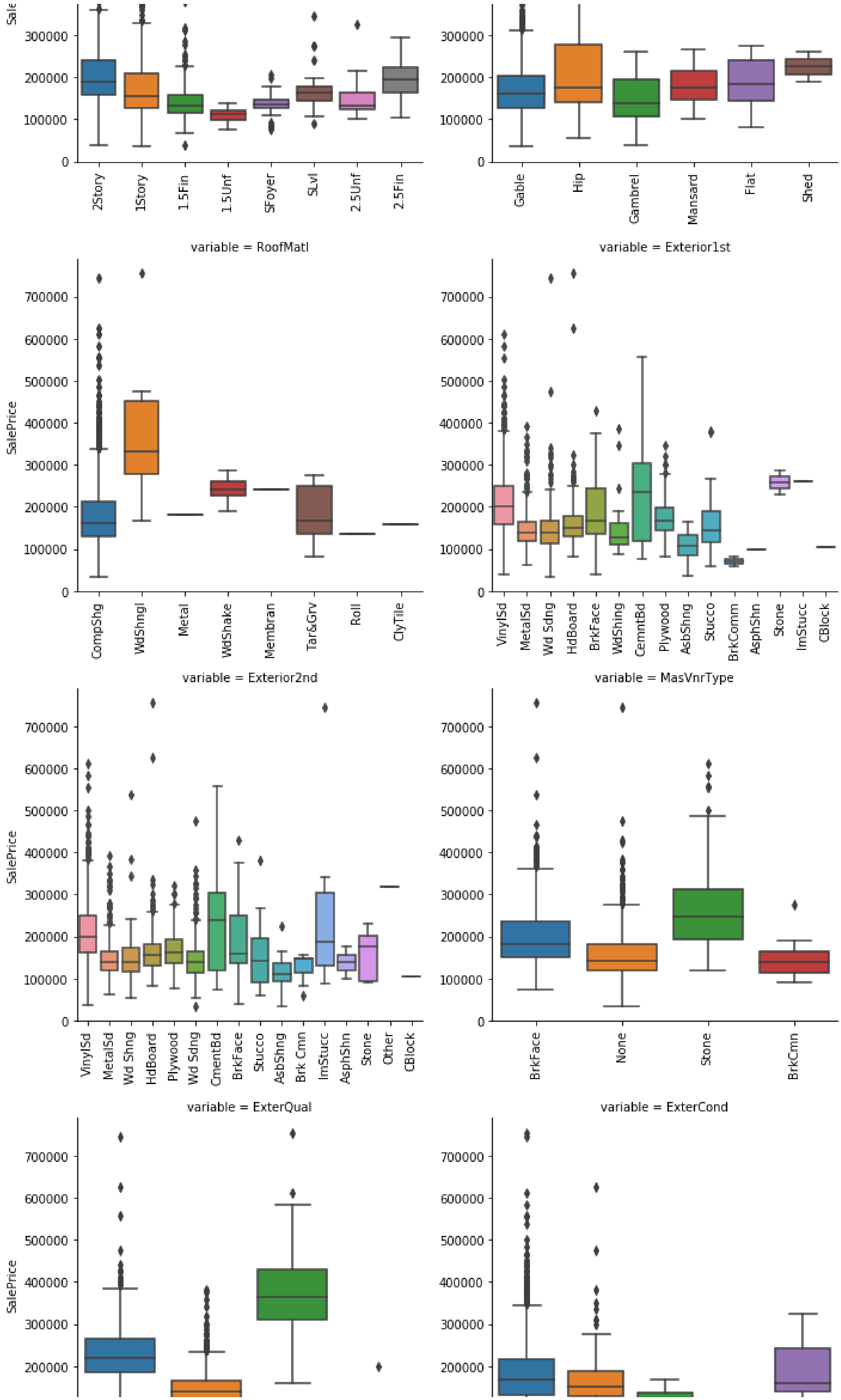
As we can observe, most of the variables seem to be rightly skewed which we will transform later on. Now let us plot box plots to check for the distribution of catgeorical variables:

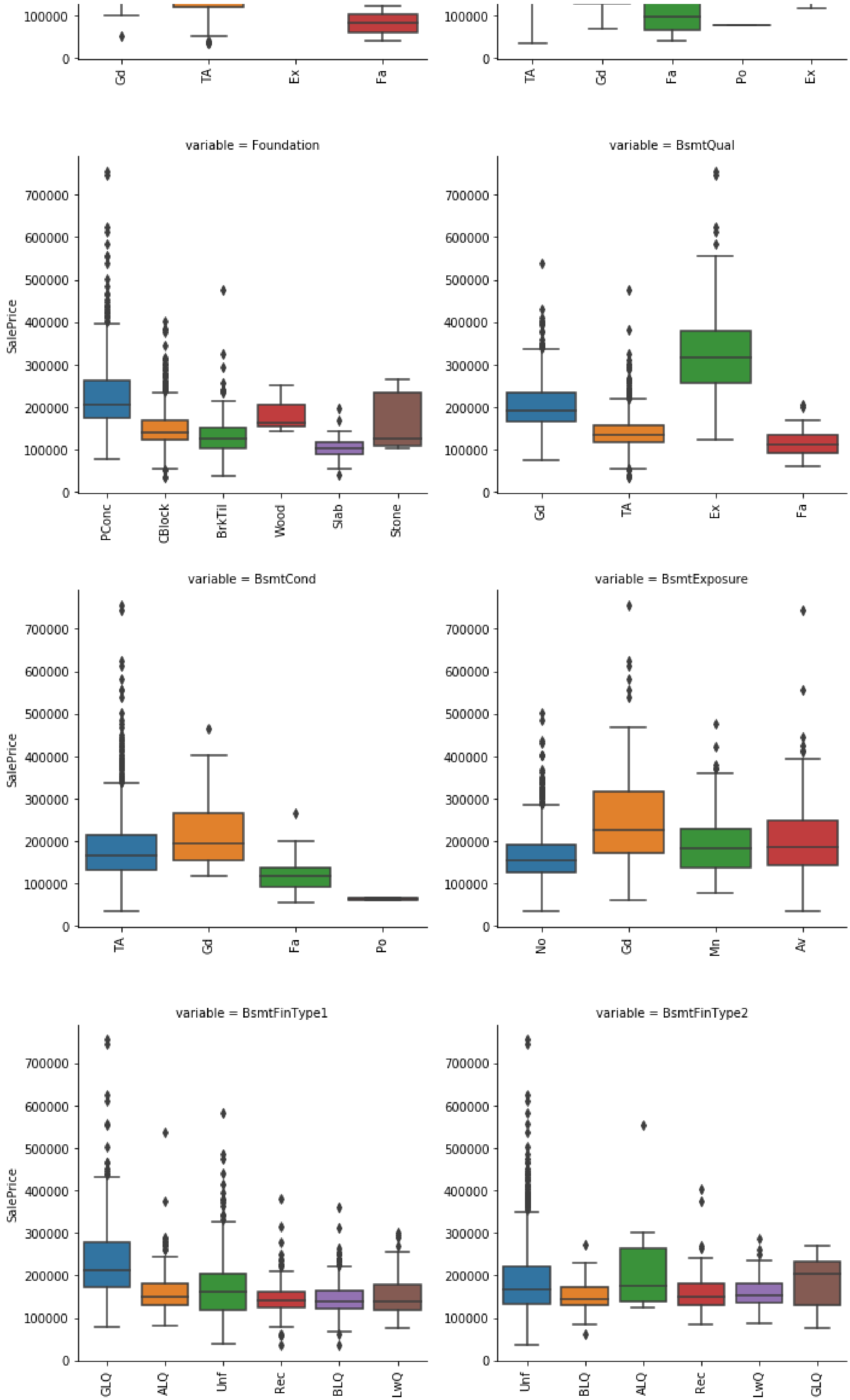
In [965]:
```
def boxplot(x,y,**kwargs):
            sns.boxplot(x=x,y=y)
            x = plt.xticks(rotation=90)

cat = [f for f in train.columns if train.dtypes[f] == 'object']

p = pd.melt(train, id_vars='SalePrice', value_vars=cat)
g = sns.FacetGrid (p, col='variable', col_wrap=2, sharex=False, sharey=False,
size=5)
g = g.map(boxplot, 'value','SalePrice')
g
```

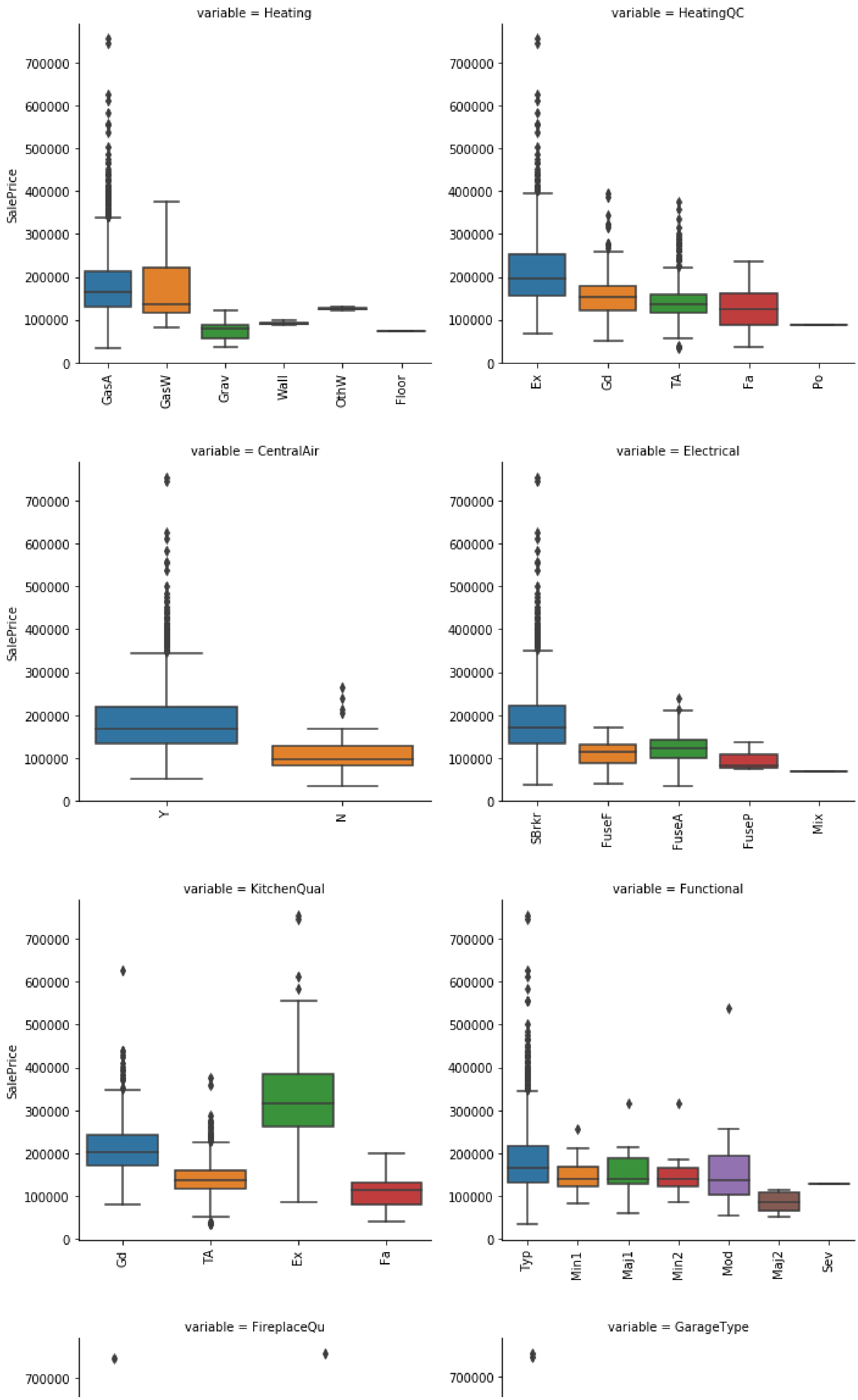
```
Out[965]: <seaborn.axisgrid.FacetGrid at 0x25ba4b409b0>
```

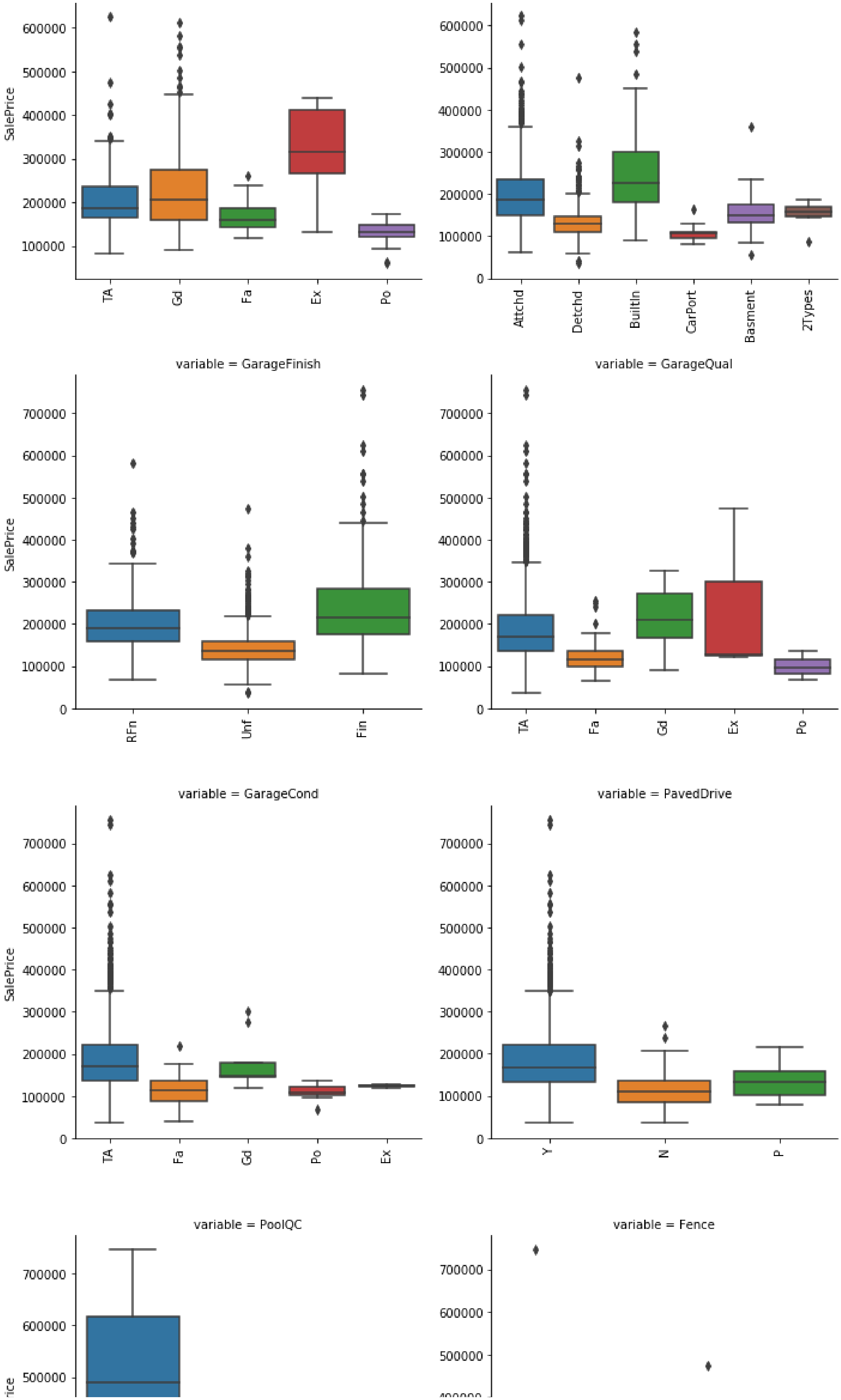

variable = MSZoning
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
RL
RM
C (all)
FV
RH
variable = Street
Pave
Grvl
variable = Alley
50000
100000
150000
200000
250000
Grvl
Pave
variable = LotShape
Reg
IR1
IR2
IR3
variable = LandContour
Lvl
Bnk
Low
HLS
variable = Utilities
AllPub
NoSeWa
variable = LotConfig
variable = LandSlope

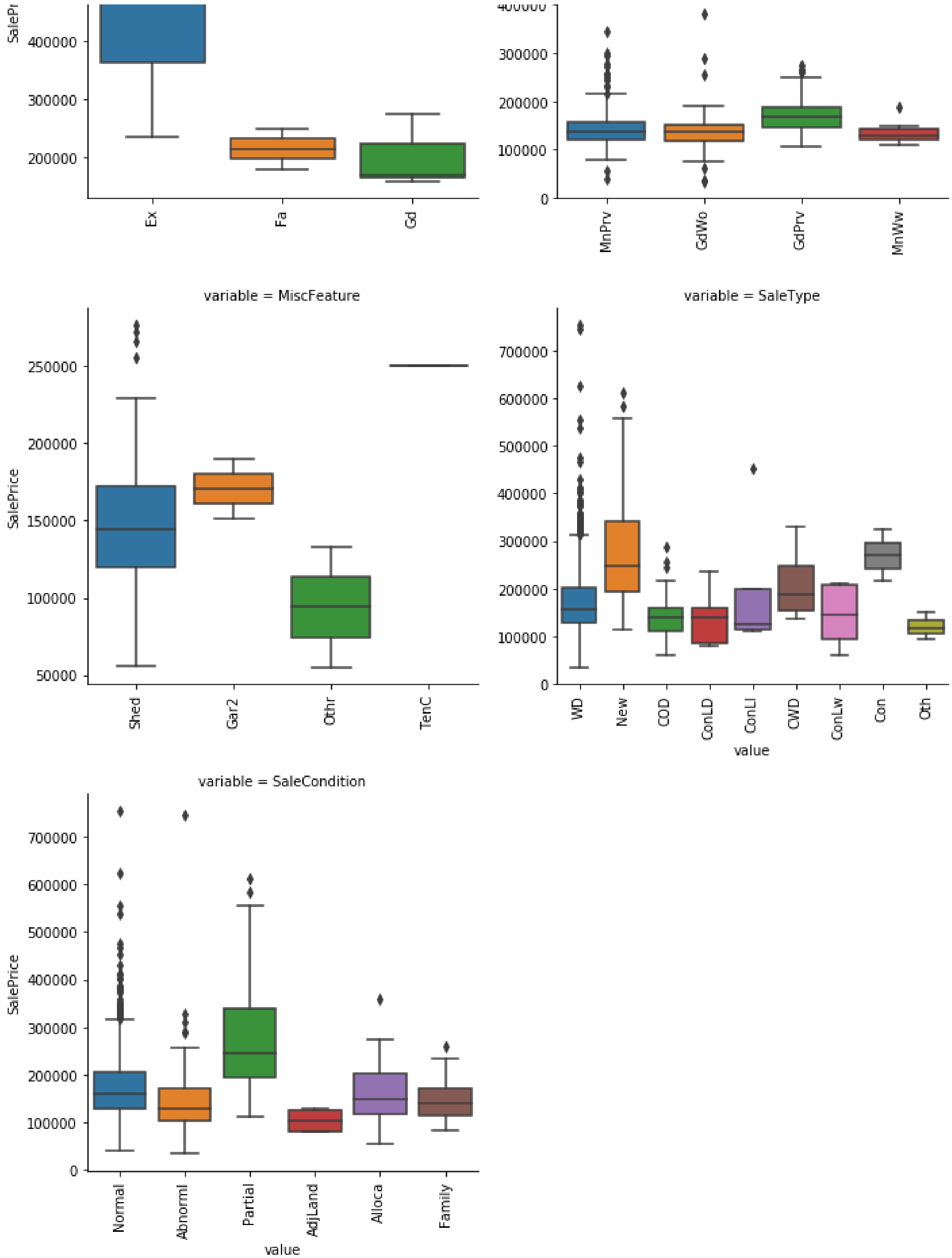

SalePrice
Inside
FR2
Corner
CulDSac
FR3
Gtl
Mod
Sev
variable = Neighborhood
CollgCr
Veenker
Crawfor
NoRidge
Mitchel
Somerst
NWAmes
OldTown
BrkSide
Sawyer
NridgHt
NAmes
SawyerW
IDOTRR
MeadowV
Edwards
Timber
Gilbert
StoneBr
ClearCr
NPkVill
Blmngtn
BrDale
SWISU
Blueste
variable = Condition1
Norm
Feedr
PosN
Artery
RRAe
RRNn
RRAn
PosA
RRNe
variable = Condition2
Norm
Artery
RRNn
Feedr
PosN
PosA
RRAn
RRAe
variable = BldgType
1Fam
2fmCon
Duplex
TwnhsE
Twnhs
variable = HouseStyle
variable = RoofStyle


SalePrice
2Story
1Story
1.5Fin
1.5Unf
SFoyer
SLvl
2.5Unf
2.5Fin
Gable
Hip
Gambrel
Mansard
Flat
Shed
variable = RoofMatl
CompShg
WdShngl
Metal
WdShake
Membran
Tar&Grv
Roll
ClyTile
variable = Exterior1st
VinylSd
MetalSd
Wd Sdng
HdBoard
BrkFace
WdShing
CemntBd
Plywood
AsbShng
Stucco
BrkComm
AsphShn
Stone
ImStucc
CBlock
variable = Exterior2nd
VinylSd
MetalSd
Wd Shng
HdBoard
Plywood
Wd Sdng
CmentBd
BrkFace
Stucco
AsbShng
Brk Cmn
ImStucc
AsphShn
Stone
Other
CBlock
variable = MasVnrType
BrkFace
None
Stone
BrkCmn
variable = ExterQual
variable = ExterCond
0
100000
200000
300000
400000
500000
600000
700000


variable = Foundation
variable = BsmtQual
variable = BsmtCond
variable = BsmtExposure
variable = BsmtFinType1
variable = BsmtFinType2
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
Gd TA Ex Fa
TA Gd Fa Po Ex
PConc CBlock BrkTil Wood Slab Stone
Gd TA Ex Fa
TA Gd Fa Po
No Gd Mn Av
GLQ ALQ Unf Rec BLQ LwQ
Unf BLQ ALQ Rec LwQ GLQ


variable = Heating
SalePrice
0
100000
200000
300000
400000
500000
600000
700000
GasA
GasW
Grav
Wall
OthW
Floor
variable = HeatingQC
Ex
Gd
TA
Fa
Po
variable = CentralAir
Y
N
variable = Electrical
SBrkr
FuseF
FuseA
FuseP
Mix
variable = KitchenQual
Gd
TA
Ex
Fa
variable = Functional
Typ
Min1
Maj1
Min2
Mod
Maj2
Sev
variable = FireplaceQu
variable = GarageType


SalePrice
TA
Gd
Fa
Ex
Po
Attchd
Detchd
BuiltIn
CarPort
Basment
2Types
variable = GarageFinish
RFn
Unf
Fin
variable = GarageQual
TA
Fa
Gd
Ex
Po
variable = GarageCond
TA
Fa
Gd
Po
Ex
variable = PavedDrive
Y
N
P
variable = PoolQC
variable = Fence



Here, we observe that most of the variables have outliers and it will be very inconvenient to get rid of them one by one. Hence, we will leave them as it is and let the algorithm take care of it. Tree based algorithms are usually robust to outliers.

Data Pre-Processing

In this stage, we will handle outlier values, impute missing values, encode categorical variables to make the data consistent.

In [966]:
```
# Removing the outlier from GrLivArea found earlier

train.drop(train[train['GrLivArea']>4000].index, inplace = True)
train.shape
```

Out[966]: (1456, 81)

Row 666 in the test data has missing information for variables related to 'Garage'(GarageQual, GarageCond,GarageFinish, GarageYrBlt) Let us impute these values using the mode of these variables:

In [967]:
```
# imputing using mode
test.loc[666, 'GarageQual']="TA"
test.loc[666, "GarageCond"]="TA"
test.loc[666, "GarageFinish"]="Unf"
test.loc[666, "GarageYrBlt"]="1980"
```

In row 1116,in test data, all garage variables are NA except GarageType. Lets make it NA as well.

In [968]:
```
test.loc[1116,"GarageType"]=np.nan
```

Now, let us encode the categorical variables. LabelEncoder function from sklearn can be used to encode variables.

In [969]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
def factorize(data, var, fill_na= None):
    if fill_na is not None:
        data[var].fillna(fill_na, inplace = True)
    le.fit(data[var])
    data[var] = le.transform(data[var])
    return data
```

This function imputes blank levels with mode values. Let us now impute the missing values in LotFrontage variable by its median value of LotFrontage by neighborhood. To do this, we will combine our train and test set to modify both data sets at once.

In [970]:
```
# Combining train and test set
alldata = train.append(test)
alldata.shape
```

Out[970]: (2915, 81)

Now let us impute the LotFrontage variable

In [971]:
```
# imputing LotFrontage by median of Neighborhood
lot_fr_by_nhood = train['LotFrontage'].groupby(train['Neighborhood'])

for key, group in lot_fr_by_nhood:
    idx = (alldata['Neighborhood'] == key) & (alldata['LotFrontage'].isnull())
    alldata.loc[idx, 'LotFrontage'] = group.median()
```

Now, let us impute the missing values in other numeric variables by 0.

In [972]:
```
# Imputing missing values = 0
alldata['MasVnrArea'].fillna(0, inplace = True)
alldata["BsmtFinSF1"].fillna(0, inplace=True)
alldata["BsmtFinSF2"].fillna(0, inplace=True)
alldata["BsmtUnfSF"].fillna(0, inplace=True)
alldata["TotalBsmtSF"].fillna(0, inplace=True)
alldata["GarageArea"].fillna(0, inplace=True)
alldata["BsmtFullBath"].fillna(0, inplace=True)
alldata["BsmtHalfBath"].fillna(0, inplace=True)
alldata["GarageCars"].fillna(0, inplace=True)
alldata["GarageYrBlt"].fillna(0.0, inplace=True)
alldata["PoolArea"].fillna(0, inplace=True)
```

Now, let us convert the categorical variables into ordinal variables. To do this, we will simply create a dictionary of key-value pairs and map it to the variable in the data set.

In [973]:
```
qual_dict = {np.nan: 0, "Po": 1, "Fa": 2, "TA": 3, "Gd": 4, "Ex": 5}
name = np.array(['ExterQual','PoolQC' ,'ExterCond','BsmtQual','BsmtCond','Heat
ingQC','KitchenQual','FireplaceQu', 'GarageQual','GarageCond'])

for i in name:
     alldata[i] = alldata[i].map(qual_dict).astype(int)

alldata["BsmtExposure"] = alldata["BsmtExposure"].map({np.nan: 0, "No": 1, "M
n": 2, "Av": 3, "Gd": 4}).astype(int)

bsmt_fin_dict = {np.nan: 0, "Unf": 1, "LwQ": 2, "Rec": 3, "BLQ": 4, "ALQ": 5,
"GLQ": 6}
alldata["BsmtFinType1"] = alldata["BsmtFinType1"].map(bsmt_fin_dict).astype(in
t)
alldata["BsmtFinType2"] = alldata["BsmtFinType2"].map(bsmt_fin_dict).astype(in
t)
alldata["Functional"] = alldata["Functional"].map({np.nan: 0, "Sal": 1, "Sev":
 2, "Maj2": 3, "Maj1": 4, "Mod": 5, "Min2": 6, "Min1": 7, "Typ": 8}).astype(in
t)

alldata["GarageFinish"] = alldata["GarageFinish"].map({np.nan: 0, "Unf": 1, "R
Fn": 2, "Fin": 3}).astype(int)
alldata["Fence"] = alldata["Fence"].map({np.nan: 0, "MnWw": 1, "GdWo": 2, "MnP
rv": 3, "GdPrv": 4}).astype(int)

#encoding data
alldata["CentralAir"] = (alldata["CentralAir"] == "Y") * 1.0
varst = np.array(['MSSubClass','LotConfig','Neighborhood','Condition1','BldgTy
pe','HouseStyle','RoofStyle','Foundation','SaleCondition'])

for x in varst:
        factorize(alldata, x)

#encode variables and impute missing values
alldata = factorize(alldata, "MSZoning", "RL")
alldata = factorize(alldata, "Exterior1st", "Other")
alldata = factorize(alldata, "Exterior2nd", "Other")
alldata = factorize(alldata, "MasVnrType", "None")
alldata = factorize(alldata, "SaleType", "Oth")
```

Feature Engineering

Let us create some binary variables depicting the presence or absence of a category. The new feature will contain 0 or 1 values. Also we will create some more features which will be described in comments.

In [870]:
```
#creating new variable (1 or 0) based on irregular count levels
#The level with highest count is kept as 1 and rest as 0
alldata["IsRegularLotShape"] = (alldata["LotShape"] == "Reg") * 1
alldata["IsLandLevel"] = (alldata["LandContour"] == "Lvl") * 1
alldata["IsLandSlopeGentle"] = (alldata["LandSlope"] == "Gtl") * 1
alldata["IsElectricalSBrkr"] = (alldata["Electrical"] == "SBrkr") * 1
alldata["IsGarageDetached"] = (alldata["GarageType"] == "Detchd") * 1
alldata["IsPavedDrive"] = (alldata["PavedDrive"] == "Y") * 1
alldata["HasShed"] = (alldata["MiscFeature"] == "Shed") * 1
alldata["Remodeled"] = (alldata["YearRemodAdd"] != alldata["YearBuilt"]) * 1

#Did the modeling happen during the sale year?
alldata["RecentRemodel"] = (alldata["YearRemodAdd"] == alldata["YrSold"]) * 1

# Was this house sold in the year it was built?
alldata["VeryNewHouse"] = (alldata["YearBuilt"] == alldata["YrSold"]) * 1
alldata["Has2ndFloor"] = (alldata["2ndFlrSF"] == 0) * 1
alldata["HasMasVnr"] = (alldata["MasVnrArea"] == 0) * 1
alldata["HasWoodDeck"] = (alldata["WoodDeckSF"] == 0) * 1
alldata["HasOpenPorch"] = (alldata["OpenPorchSF"] == 0) * 1
alldata["HasEnclosedPorch"] = (alldata["EnclosedPorch"] == 0) * 1
alldata["Has3SsnPorch"] = (alldata["3SsnPorch"] == 0) * 1
alldata["HasScreenPorch"] = (alldata["ScreenPorch"] == 0) * 1

#setting levels with high count as 1 and the rest as 0
#you can check for them using the value_counts function
alldata["HighSeason"] = alldata["MoSold"].replace({1: 0, 2: 0, 3: 0, 4: 1, 5:
1, 6: 1, 7: 1, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0})
alldata["NewerDwelling"] = alldata["MSSubClass"].replace({20: 1, 30: 0, 40: 0,
 45: 0,50: 0, 60: 1, 70: 0, 75: 0, 80: 0, 85: 0,90: 0, 120: 1, 150: 0, 160: 0,
 180: 0, 190: 0})
```

In [974]:
```
# Checking dimensions
alldata.shape
```

Out[974]: (2915, 81)

So, now we have added 19 more columns to our initial 81 columns.

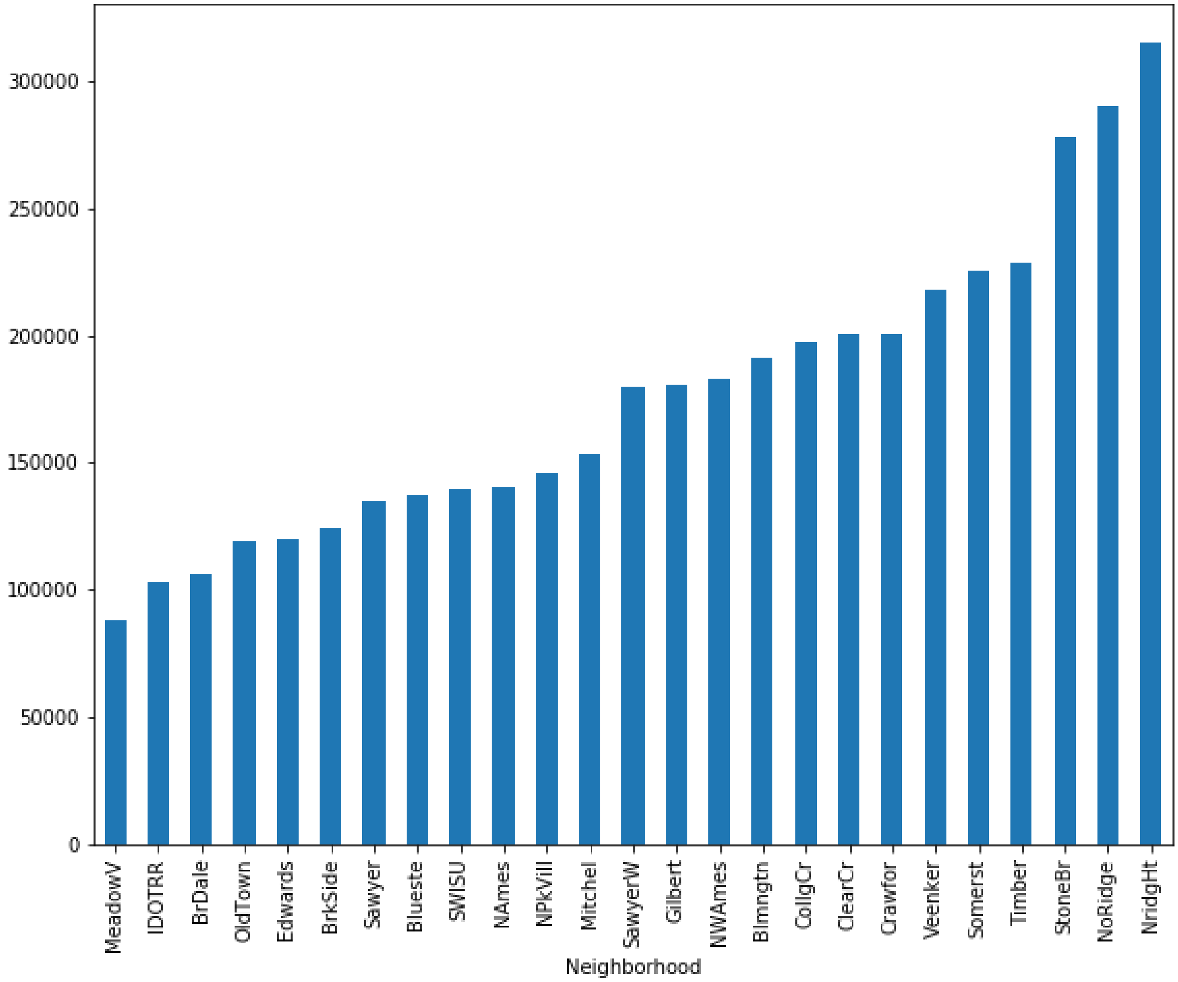
We will again append the original train and test set to create a new alldata2 file containing original features. We will use this file as a reference to create additional features:

In [975]:
```
# create alldata2
alldata2 = train.append(test)

alldata["SaleCondition_PriceDown"] = alldata2.SaleCondition.replace({'Abnorml'
: 1, 'Alloca': 1, 'AdjLand': 1, 'Family': 1, 'Normal': 0, 'Partial': 0})

# house completed before sale or not
alldata["BoughtOffPlan"] = alldata2.SaleCondition.replace({"Abnorml" : 0, "All
oca" : 0, "AdjLand" : 0, "Family" : 0, "Normal" : 0, "Partial" : 1})
alldata["BadHeating"] = alldata2.HeatingQC.replace({'Ex': 0, 'Gd': 0, 'TA': 0,
 'Fa': 1, 'Po': 1})
```

Just like Garage, we have several columns associated with the area of the property. An interesting variable could be the sum of all areas for a particular house. In addition, we can also create new features based on the year the house was built.

In [976]:
```
#calculating total area using all area columns
area_cols = ['LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF
2', 'BsmtUnfSF','TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'GrLivArea', 'GarageAre
a', 'WoodDeckSF','OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch',
'LowQualFinSF', 'PoolArea' ]

alldata["TotalArea"] = alldata[area_cols].sum(axis=1)
alldata["TotalArea1st2nd"] = alldata["1stFlrSF"] + alldata["2ndFlrSF"]
alldata["Age"] = 2010 - alldata["YearBuilt"]
alldata["TimeSinceSold"] = 2010 - alldata["YrSold"]
alldata["SeasonSold"] = alldata["MoSold"].map({12:0, 1:0, 2:0, 3:1, 4:1, 5:1,
6:2, 7:2, 8:2, 9:3, 10:3, 11:3}).astype(int)
alldata["YearsSinceRemodel"] = alldata["YrSold"] - alldata["YearRemodAdd"]

# Simplifications of existing features into bad/average/good based on counts
alldata["SimplOverallQual"] = alldata.OverallQual.replace({1 : 1, 2 : 1, 3 : 1
, 4 : 2, 5 : 2, 6 : 2, 7 : 3, 8 : 3, 9 : 3, 10 : 3})
alldata["SimplOverallCond"] = alldata.OverallCond.replace({1 : 1, 2 : 1, 3 : 1
, 4 : 2, 5 : 2, 6 : 2, 7 : 3, 8 : 3, 9 : 3, 10 : 3})
alldata["SimplPoolQC"] = alldata.PoolQC.replace({1 : 1, 2 : 1, 3 : 2, 4 : 2})
alldata["SimplGarageCond"] = alldata.GarageCond.replace({1 : 1, 2 : 1, 3 : 1,
4 : 2, 5 : 2})
alldata["SimplGarageQual"] = alldata.GarageQual.replace({1 : 1, 2 : 1, 3 : 1,
4 : 2, 5 : 2})
alldata["SimplFireplaceQu"] = alldata.FireplaceQu.replace({1 : 1, 2 : 1, 3 : 1
, 4 : 2, 5 : 2})
alldata["SimplFireplaceQu"] = alldata.FireplaceQu.replace({1 : 1, 2 : 1, 3 : 1
, 4 : 2, 5 : 2})
alldata["SimplFunctional"] = alldata.Functional.replace({1 : 1, 2 : 1, 3 : 2,
4 : 2, 5 : 3, 6 : 3, 7 : 3, 8 : 4})
alldata["SimplKitchenQual"] = alldata.KitchenQual.replace({1 : 1, 2 : 1, 3 : 1
, 4 : 2, 5 : 2})
alldata["SimplHeatingQC"] = alldata.HeatingQC.replace({1 : 1, 2 : 1, 3 : 1, 4
: 2, 5 : 2})
alldata["SimplBsmtFinType1"] = alldata.BsmtFinType1.replace({1 : 1, 2 : 1, 3 :
 1, 4 : 2, 5 : 2, 6 : 2})
alldata["SimplBsmtFinType2"] = alldata.BsmtFinType2.replace({1 : 1, 2 : 1, 3 :
 1, 4 : 2, 5 : 2, 6 : 2})
alldata["SimplBsmtCond"] = alldata.BsmtCond.replace({1 : 1, 2 : 1, 3 : 1, 4 :
2, 5 : 2})
alldata["SimplBsmtQual"] = alldata.BsmtQual.replace({1 : 1, 2 : 1, 3 : 1, 4 :
2, 5 : 2})
alldata["SimplExterCond"] = alldata.ExterCond.replace({1 : 1, 2 : 1, 3 : 1, 4
: 2, 5 : 2})
alldata["SimplExterQual"] = alldata.ExterQual.replace({1 : 1, 2 : 1, 3 : 1, 4
: 2, 5 : 2})

#grouping neighborhood variable based on this plot
train['SalePrice'].groupby(train['Neighborhood']).median().sort_values().plot(
kind='bar')
```

Out[976]: <matplotlib.axes._subplots.AxesSubplot at 0x25baa6e7940>



The graph above gives us a good hint on how to combine levels of the neighborhood variable into fewer levels. We can combine bars of somewhat equal height in one category. To do this, we'll simply create a dictionary and map it with variable values.

In [977]:
```
neighborhood_map = {"MeadowV" : 0, "IDOTRR" : 1, "BrDale" : 1, "OldTown" : 1,
"Edwards" : 1, "BrkSide" : 1, "Sawyer" : 1, "Blueste" : 1, "SWISU" : 2, "NAme
s" : 2, "NPkVill" : 2, "Mitchel" : 2, "SawyerW" : 2, "Gilbert" : 2, "NWAmes" :
 2, "Blmngtn" : 2, "CollgCr" : 2, "ClearCr" : 3, "Crawfor" : 3, "Veenker" : 3,
 "Somerst" : 3, "Timber" : 3, "StoneBr" : 4, "NoRidge" : 4, "NridgHt" : 4}

alldata['NeighborhoodBin'] = alldata2['Neighborhood'].map(neighborhood_map)
alldata.loc[alldata2.Neighborhood == 'NridgHt', "Neighborhood_Good"] = 1
alldata.loc[alldata2.Neighborhood == 'Crawfor', "Neighborhood_Good"] = 1
alldata.loc[alldata2.Neighborhood == 'StoneBr', "Neighborhood_Good"] = 1
alldata.loc[alldata2.Neighborhood == 'Somerst', "Neighborhood_Good"] = 1
alldata.loc[alldata2.Neighborhood == 'NoRidge', "Neighborhood_Good"] = 1
alldata["Neighborhood_Good"].fillna(0, inplace=True)
alldata["SaleCondition_PriceDown"] = alldata2.SaleCondition.replace({'Abnorml'
: 1, 'Alloca': 1, 'AdjLand': 1, 'Family': 1, 'Normal': 0, 'Partial': 0})

# House completed before sale or not
alldata["BoughtOffPlan"] = alldata2.SaleCondition.replace({"Abnorml" : 0, "All
oca" : 0, "AdjLand" : 0, "Family" : 0, "Normal" : 0, "Partial" : 1})
alldata["BadHeating"] = alldata2.HeatingQC.replace({'Ex': 0, 'Gd': 0, 'TA': 0,
 'Fa': 1, 'Po': 1})
alldata.shape
```

Out[977]:
```
(2915, 107)
```

Thus, we have created 43 more variables. Let us split it into train and test set and create some more features.

In [978]:
```
# creating new data

train_new = alldata[alldata['SalePrice'].notnull()]

test_new  = alldata[alldata['SalePrice'].isnull()]

print(train_new.shape)
print(test_new.shape)
```

```
(1456, 107)
(1459, 107)
```

Now, we'll transform numeric features and remove their skewness.

In [979]:
```
#get numeric features
numeric_features = [f for f in train_new.columns if train_new[f].dtype != obje
ct]

#transform the numeric features using log(x + 1)
from scipy.stats import skew
skewed = train_new[numeric_features].apply(lambda x: skew(x.dropna().astype(fl
oat)))
skewed = skewed[skewed > 0.75]
skewed = skewed.index
train_new[skewed] = np.log1p(train_new[skewed])
test_new[skewed] = np.log1p(test_new[skewed])
del test_new['SalePrice']
```

Now, let us standardize the numeric features

In [980]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(train_new[numeric_features])
scaled = scaler.transform(train_new[numeric_features])

for i, col in enumerate(numeric_features):
       train_new[col] = scaled[:,i]

numeric_features.remove('SalePrice')
scaled = scaler.fit_transform(test_new[numeric_features])

for i, col in enumerate(numeric_features):
      test_new[col] = scaled[:,i]
```

Now let us one hot encode the categorical variable, so that every level of a categorical variable results in a new variable with binary values (0 or 1):

In [981]:
```
def onehot(onehot_df, df, column_name, fill_na):
       onehot_df[column_name] = df[column_name]
       if fill_na is not None:
            onehot_df[column_name].fillna(fill_na, inplace=True)

       dummies = pd.get_dummies(onehot_df[column_name], prefix="_"+column_name
)
       onehot_df = onehot_df.join(dummies)
       onehot_df = onehot_df.drop([column_name], axis=1)
       return onehot_df

def munge_onehot(df):
       onehot_df = pd.DataFrame(index = df.index)

       onehot_df = onehot(onehot_df, df, "MSSubClass", None)
       onehot_df = onehot(onehot_df, df, "MSZoning", "RL")
       onehot_df = onehot(onehot_df, df, "LotConfig", None)
       onehot_df = onehot(onehot_df, df, "Neighborhood", None)
       onehot_df = onehot(onehot_df, df, "Condition1", None)
       onehot_df = onehot(onehot_df, df, "BldgType", None)
       onehot_df = onehot(onehot_df, df, "HouseStyle", None)
       onehot_df = onehot(onehot_df, df, "RoofStyle", None)
       onehot_df = onehot(onehot_df, df, "Exterior1st", "VinylSd")
       onehot_df = onehot(onehot_df, df, "Exterior2nd", "VinylSd")
       onehot_df = onehot(onehot_df, df, "Foundation", None)
       onehot_df = onehot(onehot_df, df, "SaleType", "WD")
       onehot_df = onehot(onehot_df, df, "SaleCondition", "Normal")

       #Fill in missing MasVnrType for rows that do have a MasVnrArea.
       temp_df = df[["MasVnrType", "MasVnrArea"]].copy()
       idx = (df["MasVnrArea"] != 0) & ((df["MasVnrType"] == "None") | (df["Ma
sVnrType"].isnull()))
       temp_df.loc[idx, "MasVnrType"] = "BrkFace"
       onehot_df = onehot(onehot_df, temp_df, "MasVnrType", "None")

       onehot_df = onehot(onehot_df, df, "LotShape", None)
       onehot_df = onehot(onehot_df, df, "LandContour", None)
       onehot_df = onehot(onehot_df, df, "LandSlope", None)
       onehot_df = onehot(onehot_df, df, "Electrical", "SBrkr")
       onehot_df = onehot(onehot_df, df, "GarageType", "None")
       onehot_df = onehot(onehot_df, df, "PavedDrive", None)
       onehot_df = onehot(onehot_df, df, "MiscFeature", "None")
       onehot_df = onehot(onehot_df, df, "Street", None)
       onehot_df = onehot(onehot_df, df, "Alley", "None")
       onehot_df = onehot(onehot_df, df, "Condition2", None)
       onehot_df = onehot(onehot_df, df, "RoofMatl", None)
       onehot_df = onehot(onehot_df, df, "Heating", None)

       # we'll have these as numerical variables too
       onehot_df = onehot(onehot_df, df, "ExterQual", "None")
       onehot_df = onehot(onehot_df, df, "ExterCond", "None")
       onehot_df = onehot(onehot_df, df, "BsmtQual", "None")
       onehot_df = onehot(onehot_df, df, "BsmtCond", "None")
       onehot_df = onehot(onehot_df, df, "HeatingQC", "None")
       onehot_df = onehot(onehot_df, df, "KitchenQual", "TA")
       onehot_df = onehot(onehot_df, df, "FireplaceQu", "None")
```

```
        onehot_df = onehot(onehot_df, df, "GarageQual", "None")
        onehot_df = onehot(onehot_df, df, "GarageCond", "None")
        onehot_df = onehot(onehot_df, df, "PoolQC", "None")
        onehot_df = onehot(onehot_df, df, "BsmtExposure", "None")
        onehot_df = onehot(onehot_df, df, "BsmtFinType1", "None")
        onehot_df = onehot(onehot_df, df, "BsmtFinType2", "None")
        onehot_df = onehot(onehot_df, df, "Functional", "Typ")
        onehot_df = onehot(onehot_df, df, "GarageFinish", "None")
        onehot_df = onehot(onehot_df, df, "Fence", "None")
        onehot_df = onehot(onehot_df, df, "MoSold", None)

        # Divide  the years between 1871 and 2010 into slices of 20 years
        year_map = pd.concat(pd.Series("YearBin" + str(i+1), index=range(1871+i
*20,1891+i*20))  for i in range(0, 7))
        yearbin_df = pd.DataFrame(index = df.index)
        yearbin_df["GarageYrBltBin"] = df.GarageYrBlt.map(year_map)
        yearbin_df["GarageYrBltBin"].fillna("NoGarage", inplace=True)
        yearbin_df["YearBuiltBin"] = df.YearBuilt.map(year_map)
        yearbin_df["YearRemodAddBin"] = df.YearRemodAdd.map(year_map)

        onehot_df = onehot(onehot_df, yearbin_df, "GarageYrBltBin", None)
        onehot_df = onehot(onehot_df, yearbin_df, "YearBuiltBin", None)
        onehot_df = onehot(onehot_df, yearbin_df, "YearRemodAddBin", None)
        return onehot_df

#create one-hot features
onehot_df = munge_onehot(train)

neighborhood_train = pd.DataFrame(index=train_new.shape)
neighborhood_train['NeighborhoodBin'] = train_new['NeighborhoodBin']
neighborhood_test = pd.DataFrame(index=test_new.shape)
neighborhood_test['NeighborhoodBin'] = test_new['NeighborhoodBin']

onehot_df = onehot(onehot_df, neighborhood_train, 'NeighborhoodBin', None)
```

Let's add the one-hot variables in our train data set.

In [982]:
```
train_new = train_new.join(onehot_df)
```

In [983]:
```
print('The number of rows is {0} and {1} number of columns'.format(train_new.s
hape[0], train_new.shape[1]))
```

```
The number of rows is 1456 and 415 number of columns
```

Similarly, let us add one hot encoded data in test set as well:

In [984]:
```
# test_new.drop('Alley', axis = 1)
#adding one hot features to test
onehot_df_te = munge_onehot(test)
onehot_df_te = onehot(onehot_df_te, neighborhood_test, "NeighborhoodBin", None
)
test_new = test_new.join(onehot_df_te)
test_new.shape
```

Out[984]: (1459, 398)

We osberve that the train set has more number of columns than the test set and so we will have to remove those variables and keep an equal number of columns in train and test data.

In [985]:
```
#dropping some columns from the train data as they are not found in test
drop_cols = ["_Exterior1st_ImStucc", "_Exterior1st_Stone","_Exterior2nd_Other"
,"_HouseStyle_2.5Fin","_RoofMatl_Membran", "_RoofMatl_Metal", "_RoofMatl_Roll"
, "_Condition2_RRAe", "_Condition2_RRAn", "_Condition2_RRNn", "_Heating_Floor"
, "_Heating_OthW", "_Electrical_Mix", "_MiscFeature_TenC", "_GarageQual_Ex",
"_PoolQC_Fa"]
train_new.drop(drop_cols, axis = 1, inplace=True)
train_new.shape
```

Out[985]: (1456, 399)

In [986]:
```
# Removing columns with lots of zeroes
#removing one column missing from train data
test_new.drop(["_MSSubClass_150"], axis=1, inplace=True)

# Drop these columns
drop_cols = ["_Condition2_PosN", # only two are not zero
         "_MSZoning_C (all)",
         "_MSSubClass_160"]

train_new.drop(drop_cols, axis=1, inplace=True)
test_new.drop(drop_cols, axis=1, inplace=True)
```

Let us transform the response variable and store it in a new array:

In [987]:
```
# Creating a label set
label_df = pd.DataFrame(index = train_new.index, columns=['SalePrice'])
label_df['SalePrice']= np.log(train['SalePrice'])
print("Train set size:", train_new.shape)
print("Test set size:", test_new.shape)
```

```
Train set size: (1456, 396)
Test set size: (1459, 394)
```

Model Training and Evaluation

So, we are done with the Data Preprocessing part. Let us move onto training and evaluating the model. We will use different Machine Learning algorithms: Multiple Linear regression, Lasso Regression,Elastic Net regression, Support Vector Regression, ensemble methods such as XGBoost, Random Forest and lastly Artificial Neural Network.

In [988]:
```
from sklearn import preprocessing
for f in train_new.columns:
    if train_new[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(train_new[f].values))
        train_new[f] = lbl.transform(list(train_new[f].values))

for f in label_df.columns:
    if label_df[f].dtype=='object':
        lbl = preprocessing.LabelEncoder()
        lbl.fit(list(label_df[f].values))
        label_df[f] = lbl.transform(list(label_df[f].values))

# train.fillna((-999), inplace=True)
# test.fillna((-999), inplace=True)

train_new=np.array(train_new)
label_df=np.array(label_df)
train_new = train_new.astype(float)
label_df = label_df.astype(float)
```

Model 1: Multiple Linear Regression

In [989]:
```
# Fitting the Model
import statsmodels.api as sm

model = sm.OLS(label_df, train_new).fit()
```

In [990]:
```
from sklearn.metrics import mean_squared_error
def rmse(y_test,y_pred):
      return np.sqrt(mean_squared_error(y_test,y_pred))

# run prediction on training set to get an idea of how well it does
y_pred = model.predict(train_new)
y_test = label_df
print("Linear Regression score on training set: ", rmse(y_test, y_pred))
```

```
Linear Regression score on training set:  5.202388192270092e-07
```

The rmse is very less, almost equal to zero. It seems, there might be a chance of Overfitting, let us check the model diagnostics:

In [991]:
```
model.summary()
```

Out[991]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	1.000
Model:	OLS	**Adj. R-squared:**	1.000
Method:	Least Squares	**F-statistic:**	2.278e+12
Date:	Sun, 02 Dec 2018	**Prob (F-statistic):**	0.00
Time:	18:48:57	**Log-Likelihood:**	19001.
No. Observations:	1456	**AIC:**	-3.741e+04
Df Residuals:	1160	**BIC:**	-3.585e+04
Df Model:	295		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
x1	-5.572e-08	8.77e-08	-0.635	0.525	-2.28e-07	1.16e-07
x2	-4.302e-07	1.32e-07	-3.258	0.001	-6.89e-07	-1.71e-07
x3	-2.237e-08	1.75e-08	-1.276	0.202	-5.68e-08	1.2e-08
x4	0.2769	1.19e-07	2.32e+06	0.000	0.277	0.277
x5	6.38e-08	3.04e-08	2.098	0.036	4.13e-09	1.23e-07
x6	0.1650	1.5e-07	1.1e+06	0.000	0.165	0.165
x7	-0.1271	1.78e-07	-7.12e+05	0.000	-0.127	-0.127
x8	0.0110	1.2e-07	9.17e+04	0.000	0.011	0.011
x9	-1.183e-07	3.94e-08	-2.999	0.003	-1.96e-07	-4.09e-08
x10	-9.765e-08	1.33e-07	-0.735	0.463	-3.58e-07	1.63e-07
x11	0.0036	2.21e-08	1.61e+05	0.000	0.004	0.004
x12	0.1814	1.38e-07	1.32e+06	0.000	0.181	0.181
x13	2.43e-08	2.71e-08	0.897	0.370	-2.89e-08	7.75e-08
x14	1.359e-08	1.91e-08	0.710	0.478	-2.4e-08	5.11e-08
x15	-0.0456	5.37e-08	-8.48e+05	0.000	-0.046	-0.046
x16	-7.45e-09	3.22e-08	-0.232	0.817	-7.06e-08	5.57e-08
x17	1.229e-07	2.62e-08	4.699	0.000	7.16e-08	1.74e-07
x18	0.0187	2.96e-08	6.31e+05	0.000	0.019	0.019
x19	-1.9e-07	2.58e-07	-0.738	0.461	-6.95e-07	3.15e-07
x20	2.465e-07	6.29e-07	0.392	0.695	-9.88e-07	1.48e-06
x21	2.747e-08	2.13e-08	1.291	0.197	-1.43e-08	6.92e-08

x22	-0.0609	7.9e-08	-7.72e+05	0.000	-0.061	-0.061
x23	-0.0452	6.58e-08	-6.88e+05	0.000	-0.045	-0.045
x24	-8.18e-07	7.76e-07	-1.054	0.292	-2.34e-06	7.05e-07
x25	-0.0105	1.52e-07	-6.93e+04	0.000	-0.011	-0.011
x26	0.1641	7.11e-08	2.31e+06	0.000	0.164	0.164
x27	0.0018	4.95e-08	3.69e+04	0.000	0.002	0.002
x28	-7.568e-08	4.35e-08	-1.738	0.082	-1.61e-07	9.75e-09
x29	0.0368	4.64e-08	7.93e+05	0.000	0.037	0.037
x30	-9.47e-08	3.26e-08	-2.905	0.004	-1.59e-07	-3.08e-08
x31	-0.0101	7.25e-08	-1.39e+05	0.000	-0.010	-0.010
x32	-7.808e-08	4.54e-08	-1.719	0.086	-1.67e-07	1.1e-08
x33	-6.231e-08	4.52e-08	-1.378	0.168	-1.51e-07	2.64e-08
x34	-0.1439	1.59e-07	-9.06e+05	0.000	-0.144	-0.144
x35	-0.0222	3.68e-08	-6.02e+05	0.000	-0.022	-0.022
x36	-0.1045	2.23e-07	-4.69e+05	0.000	-0.105	-0.105
x37	0.1414	7.26e-08	1.95e+06	0.000	0.141	0.141
x38	2.747e-09	6e-09	0.458	0.647	-9.02e-09	1.45e-08
x39	1.569e-07	4.62e-07	0.340	0.734	-7.49e-07	1.06e-06
x40	-3.031e-08	2.83e-08	-1.069	0.285	-8.59e-08	2.53e-08
x41	1.423e-07	2.14e-07	0.664	0.507	-2.78e-07	5.63e-07
x42	-0.0710	1.53e-07	-4.64e+05	0.000	-0.071	-0.071
x43	-0.0011	1.14e-07	-9629.875	0.000	-0.001	-0.001
x44	2.394e-09	1.72e-08	0.139	0.889	-3.13e-08	3.61e-08
x45	2.044e-08	3.68e-08	0.556	0.578	-5.17e-08	9.26e-08
x46	-0.0143	3.39e-08	-4.21e+05	0.000	-0.014	-0.014
x47	0.2077	9.07e-08	2.29e+06	0.000	0.208	0.208
x48	0.2769	1.44e-07	1.92e+06	0.000	0.277	0.277
x49	-2.648e-07	1.99e-07	-1.332	0.183	-6.55e-07	1.25e-07
x50	-0.1092	6.25e-08	-1.75e+06	0.000	-0.109	-0.109
x51	5.268e-08	3.11e-08	1.695	0.090	-8.29e-09	1.14e-07
x52	0.2077	8.81e-08	2.36e+06	0.000	0.208	0.208
x53	3.88e-08	4.84e-08	0.802	0.423	-5.62e-08	1.34e-07
x54	0.0632	2.91e-07	2.17e+05	0.000	0.063	0.063

x55	-0.0524	6.05e-08	-8.65e+05	0.000	-0.052	-0.052
x56	-7.235e-08	8.59e-08	-0.843	0.400	-2.41e-07	9.61e-08
x57	-2.696e-08	1.34e-07	-0.201	0.841	-2.9e-07	2.36e-07
x58	0.1557	3.86e-07	4.04e+05	0.000	0.156	0.156
x59	-1.534e-07	9.78e-08	-1.568	0.117	-3.45e-07	3.85e-08
x60	0.0081	2.11e-08	3.84e+05	0.000	0.008	0.008
x61	-0.0016	3.46e-08	-4.71e+04	0.000	-0.002	-0.002
x62	3.784e-09	2.19e-08	0.173	0.863	-3.92e-08	4.68e-08
x63	-5.344e-08	4.39e-08	-1.217	0.224	-1.4e-07	3.27e-08
x64	-4.363e-08	4.97e-08	-0.877	0.381	-1.41e-07	5.4e-08
x65	0.2769	1.18e-07	2.35e+06	0.000	0.277	0.277
x66	-1.411e-06	1.52e-06	-0.927	0.354	-4.4e-06	1.58e-06
x67	0.0939	3.1e-06	3.03e+04	0.000	0.094	0.094
x68	5.924e-07	5.33e-07	1.111	0.267	-4.54e-07	1.64e-06
x69	0.0283	9.26e-08	3.06e+05	0.000	0.028	0.028
x70	-0.0026	5.48e-08	-4.66e+04	0.000	-0.003	-0.003
x71	0.3959	6.96e-08	5.69e+06	0.000	0.396	0.396
x72	-0.0726	5.91e-08	-1.23e+06	0.000	-0.073	-0.073
x73	2.001e-09	1.81e-08	0.111	0.912	-3.34e-08	3.74e-08
x74	0.2769	1.57e-07	1.77e+06	0.000	0.277	0.277
x75	-1.802e-07	3.97e-08	-4.540	0.000	-2.58e-07	-1.02e-07
x76	-3.173e-07	5.93e-08	-5.351	0.000	-4.34e-07	-2.01e-07
x77	1.529e-07	7.45e-07	0.205	0.837	-1.31e-06	1.61e-06
x78	-6.958e-09	2.02e-08	-0.344	0.731	-4.66e-08	3.27e-08
x79	-1.463e-07	9.58e-08	-1.527	0.127	-3.34e-07	4.16e-08
x80	3.658e-08	5.61e-08	0.652	0.515	-7.35e-08	1.47e-07
x81	3.298e-09	9.06e-09	0.364	0.716	-1.45e-08	2.11e-08
x82	0.1631	8.24e-08	1.98e+06	0.000	0.163	0.163
x83	0.1367	1.27e-07	1.07e+06	0.000	0.137	0.137
x84	0.0185	2.38e-08	7.78e+05	0.000	0.019	0.019
x85	1.076e-07	2.19e-07	0.491	0.624	-3.23e-07	5.38e-07
x86	1.237e-07	4.72e-07	0.262	0.793	-8.02e-07	1.05e-06
x87	1.463e-07	9.58e-08	1.527	0.127	-4.16e-08	3.34e-07

x88	-3.298e-09	9.06e-09	-0.364	0.716	-2.11e-08	1.45e-08
x89	-0.0101	2.13e-08	-4.74e+05	0.000	-0.010	-0.010
x90	-3.637e-08	5.61e-08	-0.649	0.517	-1.46e-07	7.36e-08
x91	-2.972e-08	3.7e-08	-0.803	0.422	-1.02e-07	4.29e-08
x92	5.303e-08	3.83e-08	1.386	0.166	-2.2e-08	1.28e-07
x93	-0.0476	1.63e-06	-2.92e+04	0.000	-0.048	-0.048
x94	0.1478	9.15e-08	1.62e+06	0.000	0.148	0.148
x95	0.1550	1.07e-07	1.45e+06	0.000	0.155	0.155
x96	0.0706	4.11e-08	1.72e+06	0.000	0.071	0.071
x97	-0.0525	1.18e-07	-4.44e+05	0.000	-0.052	-0.052
x98	0.0278	2.11e-08	1.32e+06	0.000	0.028	0.028
x99	0.0257	8.42e-08	3.05e+05	0.000	0.026	0.026
x100	-0.0536	5.37e-08	-9.99e+05	0.000	-0.054	-0.054
x101	-0.1355	1.3e-07	-1.04e+06	0.000	-0.136	-0.136
x102	0.1365	1.61e-07	8.48e+05	0.000	0.137	0.137
x103	-0.0158	3.08e-08	-5.12e+05	0.000	-0.016	-0.016
x104	0.1899	1.26e-07	1.51e+06	0.000	0.190	0.190
x105	0.1077	5.55e-08	1.94e+06	0.000	0.108	0.108
x106	0.0018	5.96e-08	2.98e+04	0.000	0.002	0.002
x107	-0.0002	4.36e-08	-3646.717	0.000	-0.000	-0.000
x108	0.1648	3.27e-07	5.04e+05	0.000	0.165	0.165
x109	0.1215	1.69e-07	7.18e+05	0.000	0.122	0.122
x110	0.0962	4.13e-07	2.33e+05	0.000	0.096	0.096
x111	0.0782	6.12e-07	1.28e+05	0.000	0.078	0.078
x112	0.0643	2.59e-07	2.48e+05	0.000	0.064	0.064
x113	0.0529	2.4e-07	2.21e+05	0.000	0.053	0.053
x114	0.0433	2.89e-07	1.5e+05	0.000	0.043	0.043
x115	0.0350	5.25e-07	6.66e+04	0.000	0.035	0.035
x116	0.0276	4.68e-07	5.9e+04	0.000	0.028	0.028
x117	0.0210	4.67e-07	4.5e+04	0.000	0.021	0.021
x118	0.0738	1.61e-07	4.58e+05	0.000	0.074	0.074
x119	0.0096	2.42e-07	3.98e+04	0.000	0.010	0.010
x120	-0.0043	3.8e-07	-1.13e+04	0.000	-0.004	-0.004

x121	-0.0083	8.97e-07	-9299.453	0.000	-0.008	-0.008
x122	0.0828	2.61e-07	3.17e+05	0.000	0.083	0.083
x123	0.1655	1.99e-07	8.34e+05	0.000	0.166	0.166
x124	0.2483	1.18e-07	2.1e+06	0.000	0.248	0.248
x125	0.3310	1.81e-07	1.83e+06	0.000	0.331	0.331
x126	0.0312	4.4e-08	7.09e+05	0.000	0.031	0.031
x127	0.0987	8.29e-08	1.19e+06	0.000	0.099	0.099
x128	0.1661	1.2e-07	1.39e+06	0.000	0.166	0.166
x129	0.2336	2.57e-07	9.1e+05	0.000	0.234	0.234
x130	0.3010	1.69e-07	1.78e+06	0.000	0.301	0.301
x131	0.0301	1.91e-07	1.57e+05	0.000	0.030	0.030
x132	0.0322	4.21e-07	7.65e+04	0.000	0.032	0.032
x133	0.0325	2.03e-07	1.6e+05	0.000	0.033	0.033
x134	0.0328	1.45e-07	2.26e+05	0.000	0.033	0.033
x135	0.0292	1.37e-07	2.13e+05	0.000	0.029	0.029
x136	0.0314	9.93e-08	3.16e+05	0.000	0.031	0.031
x137	0.0302	1.28e-07	2.35e+05	0.000	0.030	0.030
x138	0.0339	1.1e-07	3.09e+05	0.000	0.034	0.034
x139	0.0322	1.09e-07	2.96e+05	0.000	0.032	0.032
x140	0.0344	1.69e-07	2.04e+05	0.000	0.034	0.034
x141	0.0366	1.85e-07	1.98e+05	0.000	0.037	0.037
x142	0.0330	1.06e-07	3.12e+05	0.000	0.033	0.033
x143	0.0333	8e-08	4.16e+05	0.000	0.033	0.033
x144	0.0336	3.18e-07	1.06e+05	0.000	0.034	0.034
x145	0.0338	9.52e-08	3.55e+05	0.000	0.034	0.034
x146	0.0307	1.09e-07	2.81e+05	0.000	0.031	0.031
x147	0.0309	8.91e-08	3.47e+05	0.000	0.031	0.031
x148	0.0366	1.14e-07	3.21e+05	0.000	0.037	0.037
x149	0.0349	1.62e-07	2.16e+05	0.000	0.035	0.035
x150	0.0371	1.03e-07	3.59e+05	0.000	0.037	0.037
x151	0.0355	9.88e-08	3.59e+05	0.000	0.035	0.035
x152	0.0342	1.3e-07	2.62e+05	0.000	0.034	0.034
x153	0.0326	1.3e-07	2.5e+05	0.000	0.033	0.033

x154	0.0343	1.24e-07	2.77e+05	0.000	0.034	0.034
x155	0.0346	1.72e-07	2.01e+05	0.000	0.035	0.035
x156	0.1902	1.22e-07	1.55e+06	0.000	0.190	0.190
x157	0.1425	9.94e-08	1.43e+06	0.000	0.142	0.142
x158	0.1146	7.63e-08	1.5e+06	0.000	0.115	0.115
x159	0.0948	2.24e-07	4.23e+05	0.000	0.095	0.095
x160	0.0794	1.66e-07	4.78e+05	0.000	0.079	0.079
x161	0.0669	2.13e-07	3.14e+05	0.000	0.067	0.067
x162	0.0563	1.63e-07	3.44e+05	0.000	0.056	0.056
x163	0.0471	3.6e-07	1.31e+05	0.000	0.047	0.047
x164	0.0390	2.64e-07	1.48e+05	0.000	0.039	0.039
x165	0.4853	3.11e-07	1.56e+06	0.000	0.485	0.485
x166	0.2628	5.79e-07	4.54e+05	0.000	0.263	0.263
x167	0.0738	1.61e-07	4.58e+05	0.000	0.074	0.074
x168	0.0402	1.66e-07	2.43e+05	0.000	0.040	0.040
x169	-0.0315	1.2e-07	-2.62e+05	0.000	-0.031	-0.031
x170	-0.0017	6.13e-07	-2814.736	0.000	-0.002	-0.002
x171	-0.0011	8.63e-07	-1332.155	0.000	-0.001	-0.001
x172	-0.0006	5.92e-07	-974.096	0.000	-0.001	-0.001
x173	0.0006	3.79e-07	1520.052	0.000	0.001	0.001
x174	0.0011	4.15e-07	2768.998	0.000	0.001	0.001
x175	0.0017	4.87e-07	3540.028	0.000	0.002	0.002
x176	0.0023	4.36e-07	5275.490	0.000	0.002	0.002
x177	0.2446	2.11e-07	1.16e+06	0.000	0.245	0.245
x178	0.1775	1.96e-07	9.05e+05	0.000	0.177	0.177
x179	0.1382	2.48e-07	5.58e+05	0.000	0.138	0.138
x180	0.1104	2.11e-07	5.24e+05	0.000	0.110	0.110
x181	0.0888	2.87e-07	3.1e+05	0.000	0.089	0.089
x182	0.0711	4.29e-07	1.66e+05	0.000	0.071	0.071
x183	-1.748e-06	2.08e-06	-0.843	0.400	-5.82e-06	2.32e-06
x184	-1.717e-06	2.02e-06	-0.850	0.396	-5.68e-06	2.25e-06
x185	-3.068e-06	1.76e-06	-1.741	0.082	-6.53e-06	3.9e-07
x186	-1.108e-06	1.44e-06	-0.769	0.442	-3.94e-06	1.72e-06

x187	-0.0074	7.28e-07	-1.02e+04	0.000	-0.007	-0.007
x188	-5.067e-07	1.1e-06	-0.460	0.645	-2.67e-06	1.65e-06
x189	-7.296e-08	8.46e-07	-0.086	0.931	-1.73e-06	1.59e-06
x190	4.117e-07	5.59e-07	0.736	0.462	-6.86e-07	1.51e-06
x191	7.855e-07	4.49e-07	1.748	0.081	-9.63e-08	1.67e-06
x192	9.465e-07	7.25e-07	1.305	0.192	-4.76e-07	2.37e-06
x193	1.543e-06	8.94e-07	1.727	0.084	-2.1e-07	3.3e-06
x194	1.672e-06	1.07e-06	1.567	0.117	-4.22e-07	3.77e-06
x195	1.932e-06	1.27e-06	1.518	0.129	-5.66e-07	4.43e-06
x196	-0.0268	4.23e-07	-6.34e+04	0.000	-0.027	-0.027
x197	-0.0238	5.56e-07	-4.28e+04	0.000	-0.024	-0.024
x198	-0.0208	5.68e-07	-3.67e+04	0.000	-0.021	-0.021
x199	-0.0179	4.74e-07	-3.77e+04	0.000	-0.018	-0.018
x200	-0.0074	7.28e-07	-1.02e+04	0.000	-0.007	-0.007
x201	-0.0119	6.43e-07	-1.85e+04	0.000	-0.012	-0.012
x202	-0.0089	5.52e-07	-1.62e+04	0.000	-0.009	-0.009
x203	-0.0060	6.18e-07	-9639.199	0.000	-0.006	-0.006
x204	-0.0030	6.55e-07	-4543.984	0.000	-0.003	-0.003
x205	0.0030	6.99e-07	4256.416	0.000	0.003	0.003
x206	0.0060	8.01e-07	7430.751	0.000	0.006	0.006
x207	0.0089	7.94e-07	1.12e+04	0.000	0.009	0.009
x208	0.0119	7.7e-07	1.55e+04	0.000	0.012	0.012
x209	0.0149	8.54e-07	1.74e+04	0.000	0.015	0.015
x210	0.0179	8.92e-07	2e+04	0.000	0.018	0.018
x211	0.2658	1.41e-07	1.88e+06	0.000	0.266	0.266
x212	0.2148	1.04e-07	2.07e+06	0.000	0.215	0.215
x213	0.1639	1.02e-07	1.61e+06	0.000	0.164	0.164
x214	0.1130	2.4e-07	4.7e+05	0.000	0.113	0.113
x215	0.0620	2.65e-07	2.34e+05	0.000	0.062	0.062
x216	0.0111	2.25e-07	4.94e+04	0.000	0.011	0.011
x217	-0.0948	1.77e-07	-5.36e+05	0.000	-0.095	-0.095
x218	-0.0480	2.6e-07	-1.85e+05	0.000	-0.048	-0.048
x219	-0.0013	3.77e-07	-3334.426	0.000	-0.001	-0.001

x220	0.0455	2.26e-07	2.01e+05	0.000	0.046	0.046
x221	0.0923	2.78e-07	3.32e+05	0.000	0.092	0.092
x222	0.1391	2.91e-07	4.78e+05	0.000	0.139	0.139
x223	0.1858	4.21e-07	4.42e+05	0.000	0.186	0.186
x224	0.2326	3.36e-07	6.91e+05	0.000	0.233	0.233
x225	0.2794	2.18e-07	1.28e+06	0.000	0.279	0.279
x226	0.0278	1.5e-07	1.85e+05	0.000	0.028	0.028
x227	0.0301	2.78e-07	1.08e+05	0.000	0.030	0.030
x228	0.0324	1.92e-07	1.68e+05	0.000	0.032	0.032
x229	0.0347	1.13e-07	3.07e+05	0.000	0.035	0.035
x230	0.5974	2.47e-07	2.42e+06	0.000	0.597	0.597
x231	0.1082	5.34e-08	2.03e+06	0.000	0.108	0.108
x232	0.2077	3.46e-07	6e+05	0.000	0.208	0.208
x233	0.2077	1.32e-07	1.58e+06	0.000	0.208	0.208
x234	0.2077	1.49e-07	1.4e+06	0.000	0.208	0.208
x235	0.2077	3.66e-07	5.67e+05	0.000	0.208	0.208
x236	0.5191	2.18e-07	2.38e+06	0.000	0.519	0.519
x237	0.3115	1.54e-07	2.03e+06	0.000	0.311	0.311
x238	0.1038	1.67e-07	6.21e+05	0.000	0.104	0.104
x239	-0.1038	9.89e-08	-1.05e+06	0.000	-0.104	-0.104
x240	0.5191	2.17e-07	2.39e+06	0.000	0.519	0.519
x241	0.3115	1.53e-07	2.04e+06	0.000	0.311	0.311
x242	0.1038	1.13e-07	9.16e+05	0.000	0.104	0.104
x243	-0.1038	7.57e-08	-1.37e+06	0.000	-0.104	-0.104
x244	0.5537	2.31e-07	2.4e+06	0.000	0.554	0.554
x245	0.2769	1.53e-07	1.81e+06	0.000	0.277	0.277
x246	8.058e-08	9.19e-08	0.877	0.381	-9.98e-08	2.61e-07
x247	1.108e-06	2.22e-06	0.499	0.618	-3.25e-06	5.47e-06
x248	6.212e-07	1.73e-06	0.359	0.719	-2.77e-06	4.01e-06
x249	-5.234e-07	1.36e-06	-0.385	0.701	-3.19e-06	2.15e-06
x250	6.644e-08	1.33e-06	0.050	0.960	-2.54e-06	2.67e-06
x251	0.4663	2.52e-07	1.85e+06	0.000	0.466	0.466
x252	0.3249	1.75e-07	1.86e+06	0.000	0.325	0.325

x253	0.1834	1.78e-07	1.03e+06	0.000	0.183	0.183
x254	0.0420	1.03e-07	4.08e+05	0.000	0.042	0.042
x255	-0.0994	2e-07	-4.98e+05	0.000	-0.099	-0.099
x256	-0.2409	1.49e-07	-1.62e+06	0.000	-0.241	-0.241
x257	0.1543	8.1e-08	1.9e+06	0.000	0.154	0.154
x258	0.5537	2.31e-07	2.4e+06	0.000	0.554	0.554
x259	0.2769	1.44e-07	1.93e+06	0.000	0.277	0.277
x260	-4.498e-08	4.29e-08	-1.047	0.295	-1.29e-07	3.93e-08
x261	0.4672	4.13e-07	1.13e+06	0.000	0.467	0.467
x262	-0.1557	1.76e-06	-8.86e+04	0.000	-0.156	-0.156
x263	0.3115	8.18e-07	3.81e+05	0.000	0.311	0.311
x264	0.1557	9.39e-07	1.66e+05	0.000	0.156	0.156
x265	0.5537	2.54e-07	2.18e+06	0.000	0.554	0.554
x266	0.2769	1.57e-07	1.77e+06	0.000	0.277	0.277
x267	0.5537	2.32e-07	2.39e+06	0.000	0.554	0.554
x268	-2.801e-08	4.4e-08	-0.636	0.525	-1.14e-07	5.84e-08
x269	0.2769	1.47e-07	1.88e+06	0.000	0.277	0.277
x270	-2.628e-06	1.67e-06	-1.570	0.117	-5.91e-06	6.56e-07
x271	-3.096e-07	1.35e-06	-0.229	0.819	-2.96e-06	2.34e-06
x272	-7.857e-07	1.05e-06	-0.745	0.457	-2.86e-06	1.28e-06
x273	-2.538e-06	1.19e-06	-2.141	0.033	-4.86e-06	-2.12e-07
x274	1.242e-06	1.32e-06	0.943	0.346	-1.34e-06	3.83e-06
x275	-1.05e-06	1.18e-06	-0.893	0.372	-3.36e-06	1.26e-06
x276	-1.634e-06	1.57e-06	-1.040	0.299	-4.72e-06	1.45e-06
x277	-2.642e-06	2.07e-06	-1.278	0.201	-6.7e-06	1.41e-06
x278	-9.176e-08	5.24e-07	-0.175	0.861	-1.12e-06	9.37e-07
x279	2.155e-07	4.54e-07	0.474	0.635	-6.76e-07	1.11e-06
x280	-2.236e-06	5.14e-07	-4.354	0.000	-3.24e-06	-1.23e-06
x281	1.829e-07	7.76e-07	0.236	0.814	-1.34e-06	1.71e-06
x282	0.2118	1.12e-07	1.89e+06	0.000	0.212	0.212
x283	0.1845	1.05e-07	1.76e+06	0.000	0.184	0.184
x284	0.1459	8.69e-08	1.68e+06	0.000	0.146	0.146
x285	0.2885	1.48e-07	1.95e+06	0.000	0.288	0.288

x286	0.1379	2.42e-07	5.7e+05	0.000	0.138	0.138
x287	0.2580	2.53e-07	1.02e+06	0.000	0.258	0.258
x288	0.0047	2.17e-07	2.15e+04	0.000	0.005	0.005
x289	-0.0382	1.49e-07	-2.57e+05	0.000	-0.038	-0.038
x290	0.4682	2.1e-07	2.23e+06	0.000	0.468	0.468
x291	0.2738	1.32e-07	2.08e+06	0.000	0.274	0.274
x292	0.0886	1.23e-07	7.22e+05	0.000	0.089	0.089
x293	0.2216	1.01e-07	2.19e+06	0.000	0.222	0.222
x294	0.1059	1.02e-07	1.04e+06	0.000	0.106	0.106
x295	0.1407	1.01e-07	1.39e+06	0.000	0.141	0.141
x296	0.1652	2.8e-07	5.9e+05	0.000	0.165	0.165
x297	0.2291	1.52e-07	1.51e+06	0.000	0.229	0.229
x298	0.1059	1.02e-07	1.04e+06	0.000	0.106	0.106
x299	-0.0647	2.21e-07	-2.92e+05	0.000	-0.065	-0.065
x300	0.3951	1.99e-07	1.99e+06	0.000	0.395	0.395
x301	0.3220	1.63e-07	1.98e+06	0.000	0.322	0.322
x302	0.0530	2.78e-07	1.91e+05	0.000	0.053	0.053
x303	0.2480	1.22e-07	2.04e+06	0.000	0.248	0.248
x304	-0.0211	2.79e-07	-7.55e+04	0.000	-0.021	-0.021
x305	0.2287	1.03e-07	2.22e+06	0.000	0.229	0.229
x306	0.2123	9.55e-08	2.22e+06	0.000	0.212	0.212
x307	0.2030	9.66e-08	2.1e+06	0.000	0.203	0.203
x308	0.1906	8.76e-08	2.18e+06	0.000	0.191	0.191
x309	0.2247	1.03e-07	2.18e+06	0.000	0.225	0.225
x310	0.0661	8.19e-08	8.07e+05	0.000	0.066	0.066
x311	0.1529	1.23e-07	1.25e+06	0.000	0.153	0.153
x312	0.0671	6.76e-08	9.94e+05	0.000	0.067	0.067
x313	0.2387	1.21e-07	1.97e+06	0.000	0.239	0.239
x314	0.1539	1.16e-07	1.33e+06	0.000	0.154	0.154
x315	0.1519	8.17e-08	1.86e+06	0.000	0.152	0.152
x316	0.1730	2.43e-07	7.12e+05	0.000	0.173	0.173
x317	-0.1444	6.2e-07	-2.33e+05	0.000	-0.144	-0.144
x318	0.1543	8.1e-08	1.9e+06	0.000	0.154	0.154

x319	0.0285	3.58e-07	7.98e+04	0.000	0.029	0.029
x320	0.3174	3.03e-07	1.05e+06	0.000	0.317	0.317
x321	0.1753	4.08e-07	4.29e+05	0.000	0.175	0.175
x322	0.1752	1.75e-07	1e+06	0.000	0.175	0.175
x323	-0.0245	4.16e-07	-5.89e+04	0.000	-0.024	-0.024
x324	0.1543	8.1e-08	1.9e+06	0.000	0.154	0.154
x325	-0.0246	2.2e-07	-1.11e+05	0.000	-0.025	-0.025
x326	0.3749	2.22e-07	1.69e+06	0.000	0.375	0.375
x327	0.0222	6.77e-07	3.27e+04	0.000	0.022	0.022
x328	-0.2702	8.71e-06	-3.1e+04	0.000	-0.270	-0.270
x329	0.7079	3.62e-06	1.96e+05	0.000	0.708	0.708
x330	0.1532	7.95e-08	1.93e+06	0.000	0.153	0.153
x331	0.1465	1.19e-07	1.23e+06	0.000	0.146	0.146
x332	0.1619	1.13e-07	1.43e+06	0.000	0.162	0.162
x333	0.1741	2.19e-07	7.94e+05	0.000	0.174	0.174
x334	0.1950	1.84e-07	1.06e+06	0.000	0.195	0.195
x335	0.1672	8.16e-08	2.05e+06	0.000	0.167	0.167
x336	0.1688	8.55e-08	1.97e+06	0.000	0.169	0.169
x337	0.1655	7.97e-08	2.08e+06	0.000	0.165	0.165
x338	0.0744	8.9e-08	8.36e+05	0.000	0.074	0.074
x339	0.1059	1.02e-07	1.04e+06	0.000	0.106	0.106
x340	0.0727	7.96e-08	9.14e+05	0.000	0.073	0.073
x341	0.0761	7.3e-08	1.04e+06	0.000	0.076	0.076
x342	0.0695	1.64e-07	4.24e+05	0.000	0.069	0.069
x343	0.1858	1.74e-07	1.07e+06	0.000	0.186	0.186
x344	-0.0289	1.58e-07	-1.82e+05	0.000	-0.029	-0.029
x345	0.1231	1.73e-07	7.1e+05	0.000	0.123	0.123
x346	0.1597	1.95e-07	8.21e+05	0.000	0.160	0.160
x347	-0.0604	1.7e-07	-3.55e+05	0.000	-0.060	-0.060
x348	0.3818	2.71e-07	1.41e+06	0.000	0.382	0.382
x349	0.0116	2.85e-07	4.09e+04	0.000	0.012	0.012
x350	-0.0035	2.85e-07	-1.22e+04	0.000	-0.003	-0.003
x351	0.2178	1.47e-07	1.48e+06	0.000	0.218	0.218

x352	0.2027	1.43e-07	1.42e+06	0.000	0.203	0.203
x353	0.1876	1.78e-07	1.05e+06	0.000	0.188	0.188
x354	-0.1794	3.44e-07	-5.22e+05	0.000	-0.179	-0.179
x355	0.3938	2.41e-07	1.63e+06	0.000	0.394	0.394
x356	0.2503	1.07e-07	2.34e+06	0.000	0.250	0.250
x357	0.1543	8.1e-08	1.9e+06	0.000	0.154	0.154
x358	0.2254	9.6e-08	2.35e+06	0.000	0.225	0.225
x359	0.2006	9.19e-08	2.18e+06	0.000	0.201	0.201
x360	-0.0320	6.67e-08	-4.8e+05	0.000	-0.032	-0.032
x361	0.1232	9.74e-08	1.26e+06	0.000	0.123	0.123
x362	0.0358	6.21e-08	5.77e+05	0.000	0.036	0.036
x363	0.2465	1.83e-07	1.34e+06	0.000	0.246	0.246
x364	0.4571	1.97e-07	2.32e+06	0.000	0.457	0.457
x365	0.0687	7.42e-08	9.25e+05	0.000	0.069	0.069
x366	0.0657	7.77e-08	8.45e+05	0.000	0.066	0.066
x367	0.0740	6.68e-08	1.11e+06	0.000	0.074	0.074
x368	0.0710	6.13e-08	1.16e+06	0.000	0.071	0.071
x369	0.0680	5.56e-08	1.22e+06	0.000	0.068	0.068
x370	0.0764	5.31e-08	1.44e+06	0.000	0.076	0.076
x371	0.0734	5.19e-08	1.42e+06	0.000	0.073	0.073
x372	0.0704	6.48e-08	1.09e+06	0.000	0.070	0.070
x373	0.0788	7.6e-08	1.04e+06	0.000	0.079	0.079
x374	0.0758	6.99e-08	1.08e+06	0.000	0.076	0.076
x375	0.0728	6.91e-08	1.05e+06	0.000	0.073	0.073
x376	0.0356	2.31e-08	1.54e+06	0.000	0.036	0.036
x377	0.1543	8.1e-08	1.9e+06	0.000	0.154	0.154
x378	0.1127	3.67e-07	3.07e+05	0.000	0.113	0.113
x379	0.1127	2.01e-07	5.6e+05	0.000	0.113	0.113
x380	0.1127	1.23e-07	9.17e+05	0.000	0.113	0.113
x381	0.1127	1.1e-07	1.03e+06	0.000	0.113	0.113
x382	0.1127	1.9e-07	5.93e+05	0.000	0.113	0.113
x383	0.1127	2.88e-07	3.91e+05	0.000	0.113	0.113
x384	0.1187	4.07e-07	2.92e+05	0.000	0.119	0.119

x385	0.1187	2.6e-07	4.56e+05	0.000	0.119	0.119
x386	0.1187	1.59e-07	7.47e+05	0.000	0.119	0.119
x387	0.1187	1.09e-07	1.09e+06	0.000	0.119	0.119
x388	0.1187	1.57e-07	7.54e+05	0.000	0.119	0.119
x389	0.1187	2.55e-07	4.66e+05	0.000	0.119	0.119
x390	0.1187	3.7e-07	3.21e+05	0.000	0.119	0.119
x391	0.2077	1.64e-07	1.26e+06	0.000	0.208	0.208
x392	0.2077	1.24e-07	1.67e+06	0.000	0.208	0.208
x393	0.2077	1.12e-07	1.85e+06	0.000	0.208	0.208
x394	0.2077	1.72e-07	1.21e+06	0.000	0.208	0.208
x395	4.431e-07	7.99e-07	0.555	0.579	-1.12e-06	2.01e-06
x396	-7.04e-07	6.36e-07	-1.108	0.268	-1.95e-06	5.43e-07

Omnibus:	899.359	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	35044.569
Skew:	-2.273	**Prob(JB):**	0.00
Kurtosis:	26.601	**Cond. No.**	1.08e+16

Here, we see that the Adjusted R-sq value is close to 1, which indicates Model overfit as we had predicted from the rmse value earlier. This is probably due to a very large number of variables as compared to data points which is increasing the model complexity. We will try to reduce the model complexity by reducing the number of variables.

Let us use Lasso regression for that:

Model 2: Lasso regression

We will be computing rmse score for Lasso regression with a 10-fold cross validation and 5000 iterations to find the best value.

In [992]:
```
# Lasso Regression with Cross-Validation
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LassoCV
from sklearn.model_selection import cross_val_score
model_lasso = LassoCV(alphas = np.logspace(-5, 5, 100), max_iter=5000) #lassoC
V does CV on alpha
model_lasso.fit(train_new,label_df)


from sklearn.metrics import mean_squared_error
def rmse(y_test,y_pred):
      return np.sqrt(mean_squared_error(y_test,y_pred))

# run prediction on training set to get an idea of how well it does
y_pred = model_lasso.predict(train_new)
y_test = label_df
print("Lasso regression score on training set: ", rmse(y_test, y_pred))
```

```
Lasso regression score on training set:  0.0004799373186509731
```

Looks, like Lasso performs far better than Multiple Linear Regression since its rmse score is lower.

Model 3: Elastic Net Regression

Let us now check our performance with Elasticnet Regression which is a mixed technqiue of Ridge and Lasso and which helps to solve the limitations of Ridge and Lasso.

In [993]:
```
from sklearn.linear_model import ElasticNetCV
from sklearn import linear_model
elastic_cv = linear_model.ElasticNetCV(l1_ratio= np.linspace(0.0001,1,20),alph
as = np.logspace(-5,10,50), cv=10, max_iter=2000)
elastic_cv.fit(train_new, label_df)

from sklearn.metrics import mean_squared_error
def rmse(y_test,y_pred):
      return np.sqrt(mean_squared_error(y_test,y_pred))

# run prediction on training set to get an idea of how well it does
y_pred = elastic_cv.predict(train_new)
y_test = label_df
print("Elastic score on training set: ", rmse(y_test, y_pred))
```

```
Elastic score on training set:  0.0015590740610246206
```

Thus, we see here that lasso regression outperforms ElasticNet in model performance.

Model 4: Support Vector Regression

In [994]:
```
#svr rbf kernel
svr_rbf = sklearn.svm.SVR(kernel = 'rbf',C=1e3, epsilon=0.1, gamma = 0.00002)
y_rbf=svr_rbf.fit(train_new, label_df)
#SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.2, gamma='auto',
#    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
#sklearn.svm.SVR(kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001, C
=1.0, epsilon=0.1, shrinking=True, cache_size=200,verbose=False, max_iter=-1)

def rmse(y_test,y_pred):
      return np.sqrt(mean_squared_error(y_test,y_pred))

# run prediction on training set to get an idea of how well it does
y_pred = y_rbf.predict(train_new)
y_test = label_df
print("SVR score: ", rmse(y_test, y_pred))
```

```
SVR score:  0.05066550428039031
```

Thus, we see that Support Vector regression performance is less as compared to the previous models of Lasso and ElasticNet.

Let us apply some Ensemble methods here and check their performances:

Model 5: XgBoost

In [995]:
```
# import os
## Installing XgBoost
# !pip install xgboost
# mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-5.3.0-posix-seh-rt_v4-rev
0\\mingw64\\bin'
# os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']

import xgboost as xgb

regr = xgb.XGBRegressor(colsample_bytree=0.2, gamma=0.0, learning_rate=0.05,ma
x_depth=6,
                       min_child_weight = 1.5, n_estimators =7200, reg_alpha =
0.9,
                       reg_lambda =0.6, subsample =0.2, seed = 42, silent=1)
# These parameters' values are derived using cross-validation.

regr.fit(train_new, label_df)


from sklearn.metrics import mean_squared_error
def rmse(y_test,y_pred):
      return np.sqrt(mean_squared_error(y_test,y_pred))

# run prediction on training set to get an idea of how well it does
y_pred = regr.predict(train_new)
y_test = label_df
print("XGBoost score on training set: ", rmse(y_test, y_pred))
```

```
XGBoost score on training set:  0.030622177814047802
```

Performance of XgBoost is lesser as compared to Lasso and ElasticNet but slightly better than SVR.

Model 6: Random Forest

In [996]:
```
from sklearn.ensemble import RandomForestClassifier
model_RF= RandomForestRegressor()
model_RF.fit(train_new, label_df)

def rmse(y_test,y_pred):
      return np.sqrt(mean_squared_error(y_test,y_pred))

# run prediction on training set to get an idea of how well it does
y_pred = model_RF.predict(train_new)
y_test = label_df
print("Random Forest score on training set: ", rmse(y_test, y_pred))
```

```
Random Forest score on training set:  0.003217039244326305
```

Since the data set is high dimensional (means large number of features), let us build a neural network model as well. We'll use the keras library to train the neural network.

Model 7: Artificial Neural Network

In [ ]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.preprocessing import StandardScaler

np.random.seed(10)

# create Model
# define base model
def base_model():
     model = Sequential()
     model.add(Dense(20, input_dim=398, init='normal', activation='relu'))
     model.add(Dense(10, init='normal', activation='relu'))
     model.add(Dense(1, init='normal'))
     model.compile(loss='mean_squared_error', optimizer = 'adam')
     return model

# train_new = train_new.astype(np.float)
 # test_new = test_new.astype(np.float)

seed = 7
np.random.seed(seed)

scale = StandardScaler()
X_train = scale.fit_transform(train_new)
X_test = scale.fit_transform(test_new)


keras_label = np.asmatrix(label_df)
clf = KerasRegressor(build_fn=base_model, nb_epoch=1000, batch_size=5,verbose=
0)
clf.fit(X_train,keras_label)

#make predictions
kpred = clf.predict(X_test)
kpred = np.exp(kpred)
pred_df = pd.DataFrame(kpred, index=test["Id"], columns=["SalePrice"])
# pred_df.to_csv('keras1.csv', header=True, index_label='Id')
```

We get RMSE as 0.35346 for the ANN model which is worse than the previous models.

Performances of the ML techniques in decreasing order:

In [1004]:
```
from IPython.display import HTML, display

data = [['ML Technqiue', 'RMSE performance'],
         ['Lasso Regression',0.0004],
         ['Elastic Net Regression',0.0015],
         ['Random Forest', 0.003],
         ['XgBoost', 0.0306],
         ['Support vector regression', 0.0506],
         ['ANN', 0.35],
        ]

display(HTML(
    '<table><tr>{}</tr></table>'.format(
        '</tr><tr>'.join(
            '<td>{}</td>'.format('</td><td>'.join(str(_) for _ in row)) for ro
w in data)
        )
 ))
```

ML Technqiue	RMSE performance
Lasso Regression	0.0004
Elastic Net Regression	0.0015
Random Forest	0.003
XgBoost	0.0306
Support vector regression	0.0506
ANN	0.35

Conclusions:

Thus we see that Lasso Regression is the best model in this problem case with rmse as 0.0004 far outweighing the performance of all other models.

This is mostly due to its important regularization property of variable reduction which seems to be very helpful on this dataset with a very high number of variables (396)

ElasticNet regression is the 2nd best model with rmse of 0.0015 which is fairly understandable as this algorithm has more proximity to Lasso Regression.