In [141]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectFromModel
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from scipy import stats
from sklearn.feature_selection import VarianceThreshold, mutual_info_classif, mutual
from sklearn.feature_selection import SelectKBest, SelectPercentile
import warnings
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import VarianceThreshold
import sklearn.impute
import statsmodels.api as sm
from sklearn.metrics import accuracy_score
from sklearn import linear_model
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.metrics import make_scorer
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
warnings.filterwarnings("ignore")
%matplotlib inline
```

Introduction

```
Purpose of this Epxirment is to play with Missing data.
In cotagorical data and Numeric.
There are different ways we can deal with catagorical data as given below
1."Rmove Rows or columns" :Remove the complet row but problem is loss of inf
ormation
2."Replace with Most Frequenet":Put most frequently used value in that parti
cular colum
but problem is imblancing data
3."Apply classficiation algorithem" which is quite good then option "1" and
"2"
4."Apply clustering algorithem" which is consider a ver good solution
keep in mind all these could only apply to catagorical data
```

Reading Train and Test File

In [2]:

```
#reading train
df_train=pd.read_csv("train.csv")
df_test=pd.read_csv("test.csv")
```

checking shape

In [3]:

```
#How many rows and columns we have in both train and test
df_train.shape,df_test.shape
```

Out[3]:

```
((1460, 81), (1459, 80))
```

checking data types

In [4]:

```
#How many type of nature data we have
print(df_train.dtypes.value_counts())

print(df_test.dtypes.value_counts())
```

```
object     43
int64      35
float64     3
dtype: int64
object     43
int64      26
float64    11
dtype: int64
```

checking null sapratly

Catagorical

In [5]:

```
# Finding null in Catgorical dataset
# df_train[df_train.dtypes[df_train.dtypes=='object'].index].isnull().sum()
```

In [6]:

```
# Finding null in Catgorical dataset
# df_train[df_train.dtypes[df_train.dtypes!='object'].index].isnull().sum()
```

In [7]:

```
#Count the number of Nans each COlumns has
nans=pd.isnull(df_train).sum()
len(nans[nans>0].index),nans[nans>0].index
```

Out[7]:

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

In [8]:

```
nans=pd.isnull(df_test).sum()

len(nans[nans>0].index),nans[nans>0].index
```

Out[8]:

```
(33, Index(['MSZoning', 'LotFrontage', 'Alley', 'Utilities', 'Exterior
1st',
        'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCo
nd',
        'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
        'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath',
        'BsmtHalfBath', 'KitchenQual', 'Functional', 'FireplaceQu',
        'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'Ga
rageArea',
        'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeature',
        'SaleType'],
       dtype='object'))
```

In [ ]:

```

```

In [9]:

```
def extract_feat_name_for_nan(df_local):
    '''This function return feature having Nan'''
    nans=pd.isnull(df_local).sum()
    return list(nans[nans>0].index)

def feature_transformation(local_df):
    ''' SimpleImputer and Label Encoder'''
    ''' Purpose of this function is to fill nans in training feature 'X' for tempere
        Note: Same feature 'X' nan will be prediction when it have to be treated as

    for col in local_df.columns:

        # apply Imputer to nan feature of object datatype
        if (local_df[col].dtype=='object' and local_df[col].isnull().sum()>0) or loc
            imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
            imp=imp.fit(local_df[col].values.reshape(-1, 1))

            #Transformation of Label Encoding
            local_df[col]=imp.fit_transform(local_df[col].values.reshape(-1, 1))
            local_df[col]=LabelEncoder().fit_transform(list(local_df[col])).astype(i

        elif local_df[col].dtype!='object':
            imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
            imp=imp.fit(local_df[col].values.reshape(-1, 1))
            local_df[col]=imp.fit_transform(local_df[col].values.reshape(-1, 1)).ast
```

```
    return local_df
def get_df_into_cleaning(df_local):

    #clean feature will be replace into this dataframe
    df_local_clean=df_local

    #if condition mean it only work for training dataset
    if 'SalePrice' in df_local.columns:
        df_local=df_local.drop('SalePrice',axis=1)

    #return feature having 'nan'
    nan_feature=extract_feat_name_for_nan(df_local_clean)
    print(nan_feature)

    #iterating through feature having 'Nan'
    for cur_feature in nan_feature:


        #recopy it becouse
        df_local=df_local_clean.copy()


        print("Feature : ",cur_feature)
        print("Type    : ",df_local[cur_feature].dtypes)
        print(list(df_local[cur_feature].head(20).unique()))

        #Copy the dataset
        dummy=df_local.copy()


        #DataFrame without Current nan feature missing variable
        dummy_data=dummy[dummy.columns[dummy.columns!=cur_feature]]

        print(dummy_data.index)

        # replace all other nan feature with most feaquent number for temporory time
        dummy_data=feature_transformation(dummy_data)

        # put back Alley column in Dataset
        dummy_data[cur_feature]=dummy[cur_feature]


        # Split into Train and Test based on Nan
        train_data_exp=dummy_data[pd.notnull(dummy_data[cur_feature])]
        test_data_exp=dummy_data[pd.isnull(dummy_data[cur_feature])] #predicted wil

        # Testing data for predicting
        X_test_E=test_data_exp.drop(cur_feature,axis=1)



        #Label Encoding target Column for train only where no Nan
        train_data_exp[cur_feature]=LabelEncoder().fit_transform(list(train_data_exp
```

```
        train_data_exp[cur_feature]=LabelEncoder().fit_transform(list(train_data_exp

        #Apply model for prediction number for nan
        from sklearn.neighbors import KNeighborsClassifier

        X_train_E=train_data_exp.drop(cur_feature,axis=1)
        y_train_E=train_data_exp[cur_feature]

        classes=len(train_data_exp[cur_feature].value_counts())

        #Create KNN Classifier
        knn = KNeighborsClassifier(n_neighbors=classes)

        #Train the model using the training sets
        knn.fit(X_train_E, y_train_E)
        #Predict the response for test dataset
        y_pred = knn.predict(X_test_E)


        #Predicted  value will replace in dummy based on index in next step
        dummy=feature_transformation(dummy)


        #Predicted value will put back on same indexes
        Nan_indexes=X_test_E.index

        #Predicted value will replace Nan
        dummy[cur_feature][Nan_indexes]=y_pred

        # ''' Check predicted value against Orignal value'''
        # for index in zip(dummy[cur_feature],df_together[cur_feature]):
        #     print(index)
        '''After checking replace in Test and Train : all_dataset'''
        df_local_clean[cur_feature]=dummy[cur_feature]

        print("Done")
        nans=pd.isnull(df_local_clean).sum()
        print("Remaining feature : ",len(nans[nans>0]))

    return df_local_clean


df_train_clean=get_df_into_cleaning(df_train)
# df_test_clean=get_df_into_cleaning(df_test,df_test_clean)
```

```
['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'Bsmt
Cond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical', '
FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual
', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeature']
Feature :  LotFrontage
Type    :  float64
[65.0, 80.0, 68.0, 60.0, 84.0, 85.0, 75.0, nan, 51.0, 50.0, 70.0, 91.0
```

```
, 72.0, 66.0]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  18
Feature :  Alley
Type    :  object
[nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  17
Feature :  MasVnrType
Type    :  object
['BrkFace', 'None', 'Stone']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  16
Feature :  MasVnrArea
Type    :  float64
[196.0, 0.0, 162.0, 350.0, 186.0, 240.0, 286.0, 306.0, 212.0, 180.0]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  15
Feature :  BsmtQual
Type    :  object
['Gd', 'TA', 'Ex', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  14
Feature :  BsmtCond
Type    :  object
['TA', 'Gd', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  13
Feature :  BsmtExposure
Type    :  object
['No', 'Gd', 'Mn', 'Av', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  12
Feature :  BsmtFinType1
Type    :  object
['GLQ', 'ALQ', 'Unf', 'Rec', 'BLQ', nan, 'LwQ']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  11
Feature :  BsmtFinType2
Type    :  object
['Unf', 'BLQ', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  10
Feature :  Electrical
```

```
Type    :  object
['SBrkr', 'FuseF', 'FuseA']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  9
Feature :  FireplaceQu
Type    :  object
[nan, 'TA', 'Gd', 'Fa']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  8
Feature :  GarageType
Type    :  object
['Attchd', 'Detchd', 'BuiltIn', 'CarPort']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  7
Feature :  GarageYrBlt
Type    :  float64
[2003.0, 1976.0, 2001.0, 1998.0, 2000.0, 1993.0, 2004.0, 1973.0, 1931.
0, 1939.0, 1965.0, 2005.0, 1962.0, 2006.0, 1960.0, 1991.0, 1970.0, 196
7.0, 1958.0]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  6
Feature :  GarageFinish
Type    :  object
['RFn', 'Unf', 'Fin']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  5
Feature :  GarageQual
Type    :  object
['TA', 'Fa', 'Gd']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  4
Feature :  GarageCond
Type    :  object
['TA']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  3
Feature :  PoolQC
Type    :  object
[nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  2
Feature :  Fence
Type    :  object
[nan, 'MnPrv', 'GdWo', 'GdPrv']
RangeIndex(start=0, stop=1460, step=1)
```

```
Done
Remaining feature :  1
Feature :  MiscFeature
Type    :  object
[nan, 'Shed']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  0
```

Now Lets see how much Nan column we have in full dataset

In [10]:

```
#check null in train
nans=pd.isnull(df_train_clean).sum()
nans[nans>0]
```

Out[10]:

```
Series([], dtype: int64)
```

In [11]:

```
# #check null in Test
# nans=pd.isnull(df_test_clean).sum()
# nans[nans>0]
```

check info

data types

In [12]:

```
#how many types we have df_test_clean : df_train_clean
df_train_clean.dtypes.value_counts()
```

Out[12]:

```
int64      35
object     27
float64    19
dtype: int64
```

float

In [13]:

```
#check any  df_test_clean : df_train_clean
df_train_clean.dtypes[df_train_clean.dtypes=='float64'].index
```

Out[13]:

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

In [14]:

```
#check any  df_test_clean : df_train_clean
for col in df_train_clean.dtypes[df_train_clean.dtypes=='float64'].index:
    print(col,df_train_clean[col].head(50).unique())
```

```
LotFrontage [ 65.  80.  68.  60.  84.  85.  75.  35.  51.  50.  70.  9
1.  72.  66.
 101.  57.  44. 110.  98.  47. 108. 112.  74. 115.  31.  61.  48.  33.
]
Alley [1. 0.]
MasVnrType [1. 2. 3.]
MasVnrArea [196.   0. 162. 350. 186. 240. 286. 306. 212. 180. 380. 281
. 640. 200.
 246. 132. 650. 101. 412.]
BsmtQual [2. 3. 0.]
BsmtCond [3. 1.]
BsmtExposure [3. 1. 2. 0.]
BsmtFinType1 [2. 0. 5. 4. 1. 3.]
BsmtFinType2 [5. 1. 0. 4. 3.]
Electrical [4. 1. 0. 2.]
FireplaceQu [4. 2. 1. 3. 0.]
GarageType [1. 5. 3. 4.]
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.    8. 1983. 1977.   36.]
GarageFinish [1. 2. 0.]
GarageQual [4. 1. 2.]
GarageCond [4. 1. 2.]
PoolQC [0. 1. 2.]
Fence [0. 2. 1.]
MiscFeature [2.]
```

Object

In [15]:

```
#check any  df_test_clean : df_train_clean
df_train_clean.dtypes[df_train_clean.dtypes=='object'].index
```

Out[15]:

```
Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condit
ion2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st
',
       'Exterior2nd', 'ExterQual', 'ExterCond', 'Foundation', 'Heating
',
       'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'PavedD
rive',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

In [16]:

```
#check any  df_test_clean : df_train_clean
for col in df_train_clean.dtypes[df_train_clean.dtypes=='object'].index:
    print(col,df_train_clean[col].head(50).unique())
```

```
MSZoning ['RL' 'RM' 'C (all)' 'FV']
Street ['Pave']
LotShape ['Reg' 'IR1' 'IR2']
LandContour ['Lvl' 'Bnk']
Utilities ['AllPub']
LotConfig ['Inside' 'FR2' 'Corner' 'CulDSac']
LandSlope ['Gtl']
Neighborhood ['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somer
st' 'NWAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber']
Condition1 ['Norm' 'Feedr' 'PosN' 'Artery' 'RRAe']
Condition2 ['Norm' 'Artery' 'RRNn']
BldgType ['1Fam' '2fmCon' 'Duplex' 'TwnhsE']
HouseStyle ['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer']
RoofStyle ['Gable' 'Hip' 'Gambrel']
RoofMatl ['CompShg']
Exterior1st ['VinylSd' 'MetalSd' 'Wd Sdng' 'HdBoard' 'BrkFace' 'WdShin
g' 'CemntBd'
 'Plywood' 'AsbShng']
Exterior2nd ['VinylSd' 'MetalSd' 'Wd Shng' 'HdBoard' 'Plywood' 'Wd Sdn
g' 'CmentBd'
 'BrkFace']
ExterQual ['Gd' 'TA' 'Ex']
ExterCond ['TA' 'Gd' 'Fa']
Foundation ['PConc' 'CBlock' 'BrkTil' 'Wood' 'Slab']
Heating ['GasA']
HeatingQC ['Ex' 'Gd' 'TA' 'Fa']
CentralAir ['Y' 'N']
KitchenQual ['Gd' 'TA' 'Ex' 'Fa']
Functional ['Typ' 'Min1']
PavedDrive ['Y' 'N' 'P']
SaleType ['WD' 'New' 'COD']
SaleCondition ['Normal' 'Abnorml' 'Partial' 'AdjLand']
```

Int

In [17]:

```
#check any  df_test_clean : df_train_clean
df_train_clean.dtypes[df_train_clean.dtypes=='int64'].index
```

Out[17]:

```
Index(['Id', 'MSSubClass', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtU
nfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivAr
ea',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'Bedroo
mAbvGr',
       'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars',
       'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3S
snPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'Sale
Price'],
      dtype='object')
```

In [18]:

```
#check any  df_test_clean : df_train_clean
for col in df_train_clean.dtypes[df_train_clean.dtypes=='int64'].index:
    print(col,df_train_clean[col].head(50).unique())
```

```
Id [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48
 49 50]
MSSubClass [ 60  20  70  50 190  45  90 120  30  85]
LotArea [ 8450  9600 11250  9550 14260 14115 10084 10382  6120  7420 1
1200 11924
 12968 10652 10920 11241 10791 13695  7560 14215  7449  9742  4224  82
46
 14230  7200 11478 16321  6324  8500  8544 11049 10552  7313 13418 108
59
  8532  7922  6040  8658 16905  9180  9200  7945  7658 12822 11096  44
56
  7742]
OverallQual [7 6 8 5 9 4]
OverallCond [5 8 6 7 4]
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]
YearRemodAdd [2003 1976 2002 1970 2000 1995 2005 1973 1950 1965 2006 1
962 2007 1960
 2001 1967 2004 2008 1997 1959 1990 1955 1983 1980 1966]
BsmtFinSF1 [ 706  978  486  216  655  732 1369  859    0  851  906  99
8  737  733
```

```
  578  646  504  840  188  234 1218 1277 1018 1153 1213  731  643  967
  747  280  179  456 1351   24  763]
BsmtFinSF2 [  0  32 668 486  93 491 506]
BsmtUnfSF [ 150  284  434  540  490   64  317  216  952  140  134  177
175 1494
  520  832  426    0  468  525 1158  637 1777  200  204 1566  180  486
  207  649 1228 1234  380  408 1117 1097   84  326  445  383  167  465
 1296   83 1632  736  192]
TotalBsmtSF [ 856 1262  920  756 1145  796 1686 1107  952  991 1040 11
75  912 1494
 1253  832 1004    0 1114 1029 1158  637 1777 1060 1566  900 1704 1484
  520  649 1228 1234 1398 1561 1117 1097 1297 1057 1088 1350  840  938
 1150 1752 1434 1656  736  955]
1stFlrSF [ 856 1262  920  961 1145  796 1694 1107 1022 1077 1040 1182
912 1494
 1253  854 1004 1296 1114 1339 1158 1108 1795 1060 1600  900 1704  520
  649 1228 1234 1700 1561 1132 1097 1297 1057 1152 1324 1328  884  938
 1150 1752 1518 1656  736  955]
2ndFlrSF [ 854    0  866  756 1053  566  983  752 1142 1218  668 1320
631  716]
LowQualFinSF [0]
GrLivArea [1710 1262 1786 1717 2198 1362 1694 2090 1774 1077 1040 2324
912 1494
 1253  854 1004 1296 1114 1339 2376 1108 1795 1060 1600  900 1704  520
 1317 1228 1234 1700 1561 2452 1097 1297 1057 1152 1324 1328  884  938
 1150 1752 2149 1656 1452  955]
BsmtFullBath [1 0]
BsmtHalfBath [0 1]
FullBath [2 1 3]
HalfBath [1 0]
BedroomAbvGr [3 4 1 2]
KitchenAbvGr [1 2 3]
TotRmsAbvGrd [ 8  6  7  9  5 11  4]
Fireplaces [0 1 2]
GarageCars [2 3 1 0]
GarageArea [548 460 608 642 836 480 636 484 468 205 384 736 352 840 57
6 516 294 853
 280 534 572 270 890 772 319 240 250 271 447 556 691 672 498 246   0 4
40
 308 504 300 670 826 386]
WoodDeckSF [  0 298 192  40 255 235  90 147 140 160  48 240 171 100 40
6 222 288  49
 203 113 392 145 196 168]
OpenPorchSF [ 61   0  42  35  84  30  57 204   4  21  33 213 112 102 1
54 159 110  90
  56  32  50 258  54  65  38  47  64  52 138 104  82  43 146]
EnclosedPorch [  0 272 228 205 176  87 172 102]
3SsnPorch [  0 320]
ScreenPorch [  0 176 198]
PoolArea [0]
MiscVal [  0 700 350 500]
MoSold [ 2  5  9 12 10  8 11  4  1  7  3  6]
YrSold [2008 2007 2006 2009 2010]
```

```
SalePrice [208500 181500 223500 140000 250000 143000 307000 200000 129
900 118000
 129500 345000 144000 279500 157000 132000 149000  90000 159000 139000
 325300 139400 230000 154000 256300 134800 306000 207500  68500  40000
 149350 179900 165500 277500 309000 145000 153000 109000  82000 160000
 170000 130250 141000 319900 239686 249700 113000 127000]
```

In [ ]:

In [ ]:

Transoformation None 'nan' Catagorical Feature

In [19]:

```
for col in df_train_clean.dtypes[df_train_clean.dtypes=='object'].index:
    '''All object data will be convert to float'''
    df_train_clean[col]=LabelEncoder().fit_transform(list(df_train_clean[col])).asty
    print(list(df_train_clean[col].head(10)))
```

```
[3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 4.0, 3.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 3.0, 3.0]
[3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[4.0, 2.0, 4.0, 0.0, 2.0, 4.0, 4.0, 0.0, 4.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[5.0, 24.0, 5.0, 6.0, 15.0, 11.0, 21.0, 14.0, 17.0, 3.0]
[2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 4.0, 0.0, 0.0]
[2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
[5.0, 2.0, 5.0, 5.0, 5.0, 0.0, 2.0, 5.0, 0.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[12.0, 8.0, 12.0, 13.0, 12.0, 12.0, 12.0, 6.0, 3.0, 8.0]
[13.0, 8.0, 13.0, 15.0, 13.0, 13.0, 13.0, 6.0, 15.0, 8.0]
[2.0, 3.0, 2.0, 3.0, 2.0, 3.0, 2.0, 3.0, 3.0, 3.0]
[4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0]
[2.0, 1.0, 2.0, 0.0, 2.0, 5.0, 2.0, 1.0, 0.0, 0.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[2.0, 3.0, 2.0, 2.0, 2.0, 3.0, 2.0, 3.0, 3.0, 3.0]
[6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 2.0, 6.0]
[2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
[8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0]
[4.0, 4.0, 4.0, 0.0, 4.0, 4.0, 4.0, 4.0, 0.0, 4.0]
```

In [20]:

```
# for col in df_test_clean.dtypes[df_test_clean.dtypes=='object'].index:
#     '''All object data will be convert to float'''
#     df_test_clean[col]=LabelEncoder().fit_transform(list(df_test_clean[col])).ast
#     print(list(df_test_clean[col].head(10)))
```

Take Dadtaset back to It's Position Local Training

In [81]:

```
# df_train=df_train_clean.iloc[:1460]
# df_test=df_train_clean.iloc[1460:]

df_train=df_train_clean

# df_train_clean
```

In [82]:

```
# df_test=df_test_clean
# df_train=df_train_clean
```

In [83]:

```
# check df_trian | df_test
nans=pd.isnull(df_train).sum()
nans[nans>0]
```

Out[83]:

```
Series([], dtype: int64)
```

In [84]:

```
# check df_trian | df_test
df_train_clean.dtypes.value_counts()
```

Out[84]:

```
float64    46
int64      35
dtype: int64
```

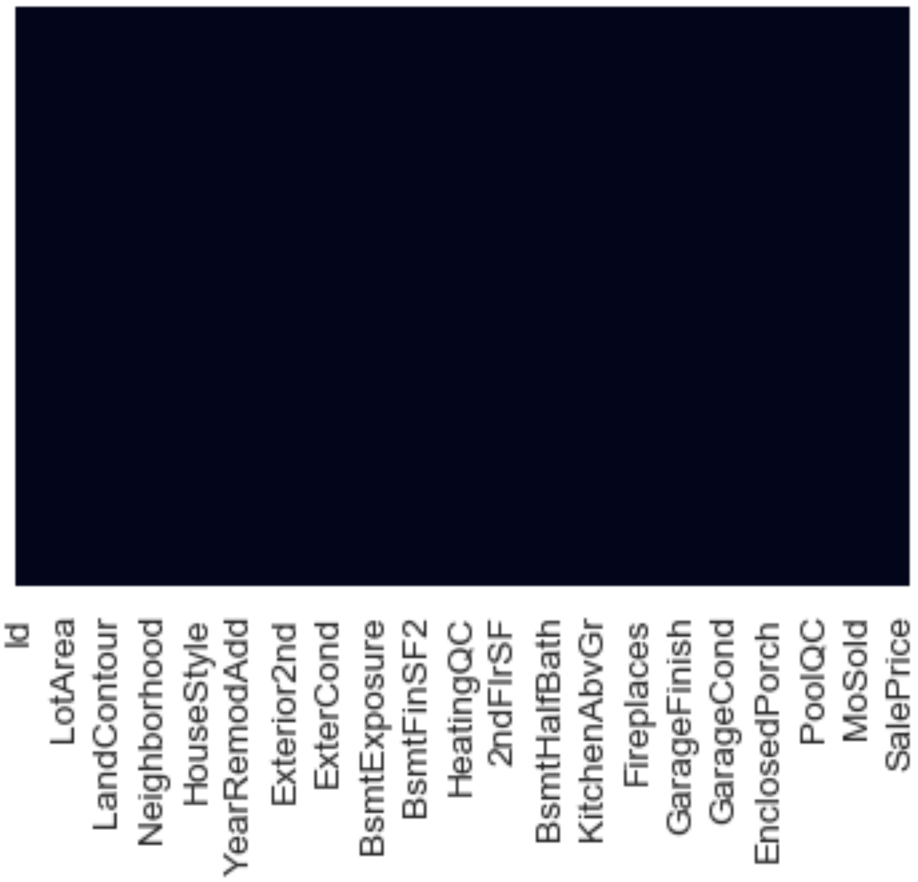
Heatmap for Missing data

In [85]:

```
#check
sns.heatmap(df_train.isnull(),yticklabels=False,cbar=False)
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c22af5190>
```



Build ML model and compare the performance of the selected feature

In [86]:

```
def run_randomForest(X_train_l,y_train_l, X_test_l, y_test_l):
    clf_gb = GradientBoostingRegressor(n_estimators=200,random_state=0)
    clf_gb.fit(X_train_l,y_train_l)
    y_pred_gb = clf_gb.predict(X_test_l)
    print("Mean Squer Error",mean_squared_error(y_test_l,y_pred_gb))
    print("sqrt of Mean Sauer Error ",np.sqrt(mean_squared_error(y_test_l,y_pred_gb
```

In [87]:

```
scaled_dataset = StandardScaler().fit_transform(df_train)
scaled_dataset=pd.DataFrame(scaled_dataset,columns=df_train.columns)


X=scaled_dataset.drop('SalePrice',axis=1)
y=scaled_dataset['SalePrice']
```

In [88]:

```

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
len(x_train.columns),x_train.columns
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[88]:

```
((1168, 80), (292, 80), (1168,), (292,))
```

Estimation of coefficients of Linear Regression

In [89]:

```
sel=SelectFromModel(GradientBoostingRegressor(n_estimators=200,random_state=0))
```

In [90]:

```
sel.fit(x_train,y_train)
```

Out[90]:

```
SelectFromModel(estimator=GradientBoostingRegressor(alpha=0.9,

criterion='friedman_mse',
                                                    init=None,
                                                    learning_rate=0.1,
                                                    loss='ls', max_dep
th=3,
                                                    max_features=None,

max_leaf_nodes=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=200,

n_iter_no_change=None,
                                                    presort='auto',
                                                    random_state=0,
                                                    subsample=1.0, tol
=0.0001,

validation_fraction=0.1,
                                                    verbose=0,
                                                    warm_start=False),
                max_features=None, norm_order=1, prefit=False, thresho
ld=None)
```

In [91]:

```
sel.get_support()
```

Out[91]:

```
array([False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False,  True, False,
       False, False,  True, False, False, False, False,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False])
```

In [92]:

```
# sel.estimator_.coef_
```

In [93]:

```
# mean = np.mean(np.abs(sel.estimator_.coef_))
```

In [94]:

```
# mean
```

In [95]:

```
# np.abs(sel.estimator_.coef_)
```

In [96]:

```
features = x_train.columns[sel.get_support()]
features
```

Out[96]:

```
Index(['Alley', 'OverallQual', 'BsmtQual', 'BsmtFinSF1', 'TotalBsmtSF'
,
       '1stFlrSF', 'GrLivArea', 'GarageCars', 'PoolQC'],
      dtype='object')
```

In [97]:

```


X_train_reg = sel.transform(x_train)
X_test_reg = sel.transform(x_test)
```

In [98]:

```
X_test_reg.shape
```

Out[98]:

```
(292, 9)
```

In [99]:

```
%%time
run_randomForest(X_train_reg, y_train,X_test_reg, y_test)
```

```
Mean Squer Error 0.09754349139496073
sqrt of Mean Sauer Error  0.31231953412324487
CPU times: user 148 ms, sys: 2.59 ms, total: 151 ms
Wall time: 150 ms
```

In [100]:

```
%%time
run_randomForest(x_train, y_train,x_test, y_test)
```

```
Mean Squer Error 0.08520220717249559
sqrt of Mean Sauer Error  0.29189417118622907
CPU times: user 571 ms, sys: 3.43 ms, total: 574 ms
Wall time: 574 ms
```

Feature selection by feature importance of random forest classifier

In [101]:

```
sel = SelectFromModel(GradientBoostingRegressor(n_estimators=200,random_state=0))
sel.fit(x_train, y_train)
sel.get_support()
```

Out[101]:

```
array([False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False,  True, False,
       False, False,  True, False, False, False, False,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False])
```

In [102]:

```
feature=x_train.columns[sel.get_support()]
```

In [103]:

```
features
```

Out[103]:

```
Index(['Alley', 'OverallQual', 'BsmtQual', 'BsmtFinSF1', 'TotalBsmtSF'
,
       '1stFlrSF', 'GrLivArea', 'GarageCars', 'PoolQC'],
      dtype='object')
```

In [104]:

```
np.mean(sel.estimator_.feature_importances_)
```

Out[104]:

```
0.0125
```

In [105]:

```
sel.estimator_.feature_importances_
```

Out[105]:

```
array([6.99053604e-04, 1.74739082e-04, 1.59981321e-03, 2.00374394e-03,
       9.04837490e-03, 0.00000000e+00, 3.41084612e-02, 9.24795432e-05,
       2.31137563e-06, 0.00000000e+00, 9.51234332e-05, 3.25017413e-04,
       1.31785755e-03, 4.40616875e-04, 1.17105694e-04, 6.70483010e-06,
       1.29089522e-04, 1.67017400e-01, 3.80087874e-03, 4.79768165e-03,
       3.55135926e-03, 6.73815747e-04, 8.68023139e-04, 6.21644815e-04,
       1.08181502e-04, 1.78969472e-04, 2.09792801e-03, 2.71241784e-04,
       9.13068303e-05, 1.05281982e-04, 1.34343206e-02, 2.26613634e-04,
       1.57999323e-03, 1.00879337e-03, 2.81043138e-02, 1.28558992e-04,
       3.18350371e-04, 1.69085513e-03, 2.50867857e-02, 0.00000000e+00,
       8.76607781e-05, 2.79615487e-03, 0.00000000e+00, 1.85963391e-02,
       6.51883311e-03, 3.81501224e-04, 7.45942630e-02, 2.21474502e-04,
       1.39918773e-04, 5.20139578e-04, 5.97247587e-05, 5.12659878e-05,
       6.21102959e-05, 2.39012278e-03, 6.37678423e-03, 2.57741173e-04,
       2.44089169e-03, 6.08827145e-04, 3.20436155e-03, 5.54950285e-03,
       6.73793255e-04, 5.48286014e-02, 3.55289468e-03, 1.67729788e-04,
       0.00000000e+00, 1.66608742e-05, 1.23135826e-03, 7.17957398e-04,
       3.81044005e-04, 8.32511337e-05, 4.87027966e-04, 4.79548379e-04,
       4.86572186e-01, 1.19892646e-02, 0.00000000e+00, 1.00718139e-05,
       5.14085012e-03, 1.71145322e-04, 3.27178311e-04, 2.38902899e-03]
)
```

In [106]:

```
x_train_rfc=sel.transform(x_train)
x_test_rfc=sel.transform(x_test)
```

In [107]:

```
%%time
run_randomForest(x_train_rfc, y_train,x_test_rfc, y_test)
```

```
Mean Squer Error 0.09754349139496073
sqrt of Mean Sauer Error  0.31231953412324487
CPU times: user 143 ms, sys: 2.49 ms, total: 146 ms
Wall time: 144 ms
```

Recursive Feature Elimination (RFE)

In [108]:

```
from sklearn.feature_selection import RFE
sel = RFE(GradientBoostingRegressor(n_estimators=200,random_state=0))
sel.fit(x_train, y_train)
```

Out[108]:

```
RFE(estimator=GradientBoostingRegressor(alpha=0.9, criterion='friedman
_mse',
                                        init=None, learning_rate=0.1,
loss='ls',
                                        max_depth=3, max_features=None
,
                                        max_leaf_nodes=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_sample
s_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=200, n_iter_no_ch
ange=None,
                                        presort='auto', random_state=0
,
                                        subsample=1.0, tol=0.0001,
                                        validation_fraction=0.1, verbo
se=0,
                                        warm_start=False),
    n_features_to_select=None, step=1, verbose=0)
```

In [ ]:

In [109]:

```
sel.get_support()
```

Out[109]:

```
array([ True, False,  True,  True,  True, False,  True, False, False,
       False, False, False,  True, False, False, False, False,  True,
        True,  True,  True, False, False,  True, False, False,  True,
       False, False, False,  True, False,  True,  True,  True, False,
        True,  True,  True, False, False,  True, False,  True,  True,
        True,  True, False, False, False, False, False, False,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
       False, False, False,  True,  True, False, False, False,  True,
        True,  True, False, False,  True, False, False,  True])
```

In [110]:

```
feature=x_train.columns[sel.get_support()]
```

In [111]:

```
feature
```

Out[111]:

```
Index(['Id', 'MSZoning', 'LotFrontage', 'LotArea', 'Alley', 'Neighborh
ood',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'Exterior1st', 'MasVnrArea', 'BsmtQual', 'BsmtExposure', 'BsmtF
inType1',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Centra
lAir',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'KitchenQu
al',
       'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'Woo
dDeckSF',
       'OpenPorchSF', 'PoolArea', 'PoolQC', 'Fence', 'MoSold',
       'SaleCondition'],
      dtype='object')
```

In [112]:

```
len(feature)
```

Out[112]:

```
40
```

In [113]:

```
x_train_rfe=sel.transform(x_train)
x_test_rfe=sel.transform(x_test)
```

In [114]:

```
%%time
run_randomForest(x_train_rfe, y_train,x_test_rfe, y_test)
```

```
Mean Squer Error 0.07624449279990299
sqrt of Mean Sauer Error  0.27612405328022943
CPU times: user 370 ms, sys: 2.93 ms, total: 373 ms
Wall time: 371 ms
```

In [115]:

```


# X=df_train.drop('SalePrice',axis=1)
# y=df_train['SalePrice']
# from sklearn.ensemble import ExtraTreesRegressor
# clf_gb = ExtraTreesRegressor(n_estimators=200)
# clf_gb.fit(x_train,y_train)
# y_pred_gb = clf_gb.predict(df_test)

```

In [116]:

```
# pred=pd.DataFrame(y_pred_gb)
# 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.csv',index=False)
```

Feature Visualization

prepare dataset

In [117]:

```
#selected feature
feature_list=[]
feature_list.extend(feature)
feature_list.append("SalePrice")
```

In [118]:

```
#selected base data triming
df_dummy=df_train[feature_list]

# corrmat=df_train[feature_list].corr()

# k = 11 #number of variables for heatmap
# best_col = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
# best_col
```

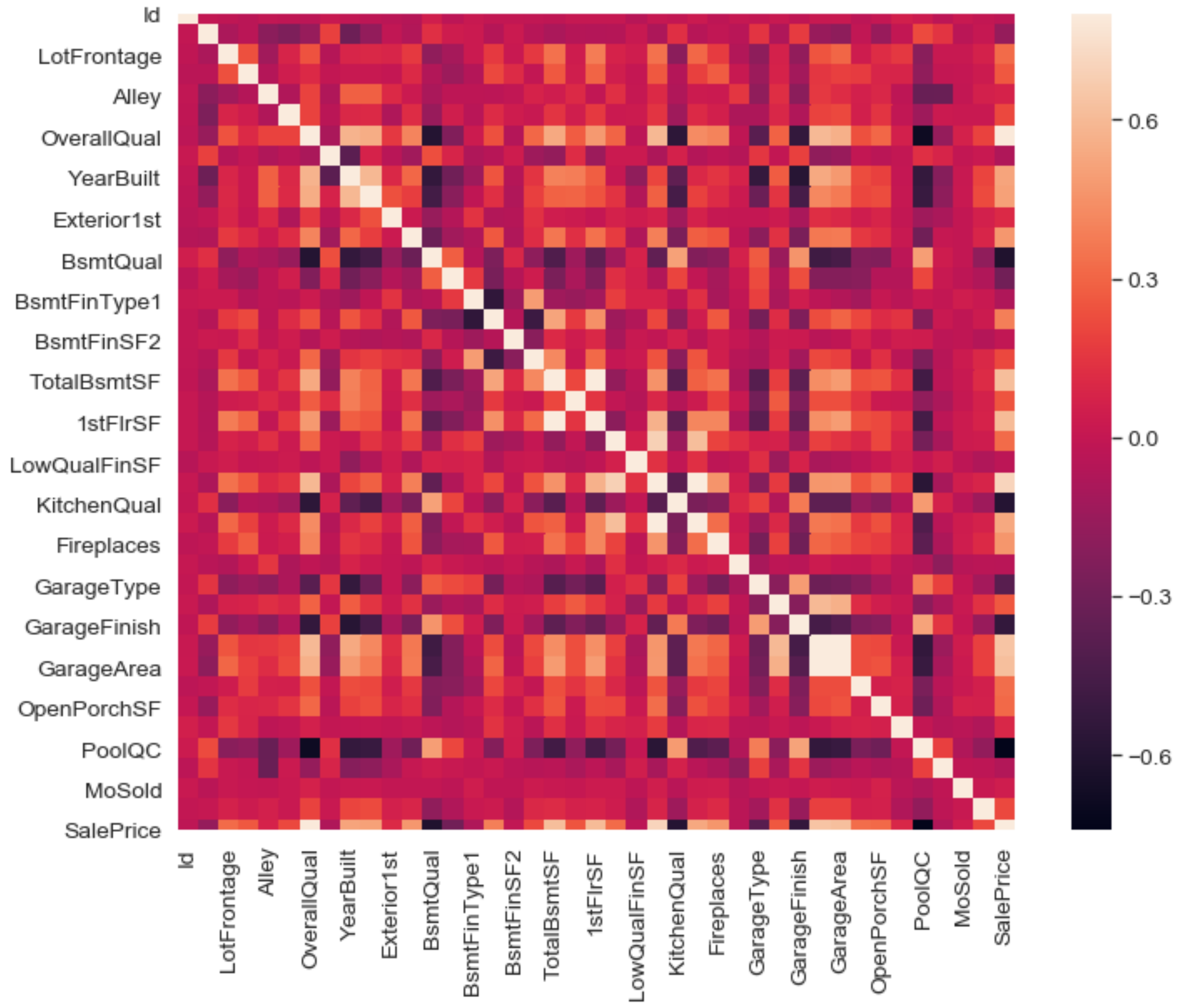
HeatMape

In [119]:

```
corrmat=df_dummy.corr()
f,a=plt.subplots(figsize=(12,9))
sns.heatmap(corrmat,vmax=.8,square=True)
```

Out[119]:

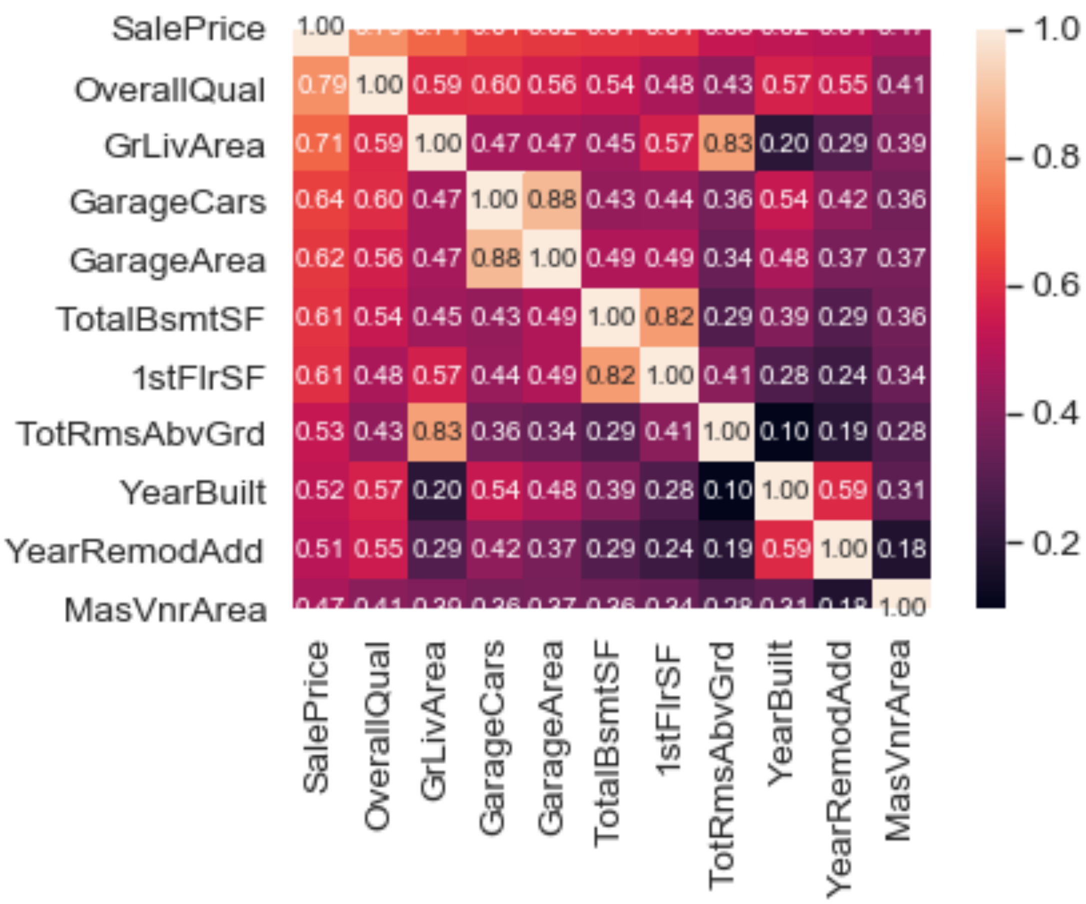
```
<matplotlib.axes._subplots.AxesSubplot at 0x1c22b85290>
```



In [120]:

```
#saleprice correlation matrix
k = 11 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index

cm = np.corrcoef(df_dummy[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size
plt.show()
```



```
According to our crystal ball, these are the variables most correlated with
'SalePrice'. My thoughts on this:

'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'S
alePrice'. Check!
'GarageCars' and 'GarageArea' are also some of the most strongly correlated
variables. However, as we discussed
in the last sub-point, the number of cars that fit into the garage is a cons
equence of the garage area.
'GarageCars' and 'GarageArea' are like twin brothers. You'll never be able t
o distinguish them. Therefore, we
just need one of these variables in our analysis (we can keep 'GarageCars' s
ince its correlation with
'SalePrice' is higher).
'TotalBsmtSF' and '1stFloor' also seem to be twin brothers. We can keep 'Tot
alBsmtSF' just to say that our
first
guess was right (re-read 'So... What can we expect?').
'FullBath'?? Really?
'TotRmsAbvGrd' and 'GrLivArea', twin brothers again. Is this dataset from Ch
ernobyl?
Ah... 'YearBuilt'... It seems that 'YearBuilt' is slightly correlated with '
SalePrice'. Honestly, it scares me to
think about 'YearBuilt' because I start feeling that we should do a little b
it of time-series analysis to get
this right. I'll leave this as a homework for you.
Let's proceed to the scatter plots.
```

Scatter plot

In [121]:

```
# k = 15 #number of variables for heatmap
# cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index
# cols
# sns.set()
# sns.pairplot(df_dummy[cols], size = 2.5)
# plt.show();
```

Dealing Missing Value

In [122]:

```
k = 11 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index


total=df_dummy[cols].isnull().sum().sort_values(ascending=False)
percent=(df_dummy[cols].isnull().sum()/df_dummy[cols].isnull().count()).sort_values(
missing_data=pd.concat([total,percent],axis=1,keys=['Total','Percent'])
missing_data.head(20)
```

Out[122]:

	Total	Percent
MasVnrArea	0	0.0
YearRemodAdd	0	0.0
YearBuilt	0	0.0
TotRmsAbvGrd	0	0.0
1stFlrSF	0	0.0
TotalBsmtSF	0	0.0
GarageArea	0	0.0
GarageCars	0	0.0
GrLivArea	0	0.0
OverallQual	0	0.0
SalePrice	0	0.0

out Liars!

```
Outliers is also something that we should be aware of. Why? Because outliers
can markedly affect our models and can be a valuable source of information,
providing us insights about specific behaviours.

Outliers is a complex subject and it deserves more attention. Here, we'll ju
st do a quick analysis through the standard deviation of 'SalePrice' and a s
et of scatter plots.
```

Univeriate

In [123]:

```
# #standardizing data
k = 11 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index

saleprice_scaled=StandardScaler().fit_transform(df_train['SalePrice'][:,np.newaxis])
# low_range=saleprice_scaled[saleprice_scaled[:0]]
low_range=saleprice_scaled[saleprice_scaled[:,0].argsort()][:10]#argsort give sorte
high_range=saleprice_scaled[saleprice_scaled[:,0].argsort()][-10:]
print('out range (low) of the distribution :')
print(low_range)
print('\nouter range (high) of the distrbution :')
print(high_range)
```

```
out range (low) of the distribution :
[[-1.83870376]
 [-1.83352844]
 [-1.80092766]
 [-1.78329881]
 [-1.77448439]
 [-1.62337999]
 [-1.61708398]
 [-1.58560389]
 [-1.58560389]
 [-1.5731    ]]

outer range (high) of the distrbution :
[[3.82897043]
 [4.04098249]
 [4.49634819]
 [4.71041276]
 [4.73032076]
 [5.06214602]
 [5.42383959]
 [5.59185509]
 [7.10289909]
 [7.22881942]]
```

Bivariate analysis

1 GrLivArea

In [124]:

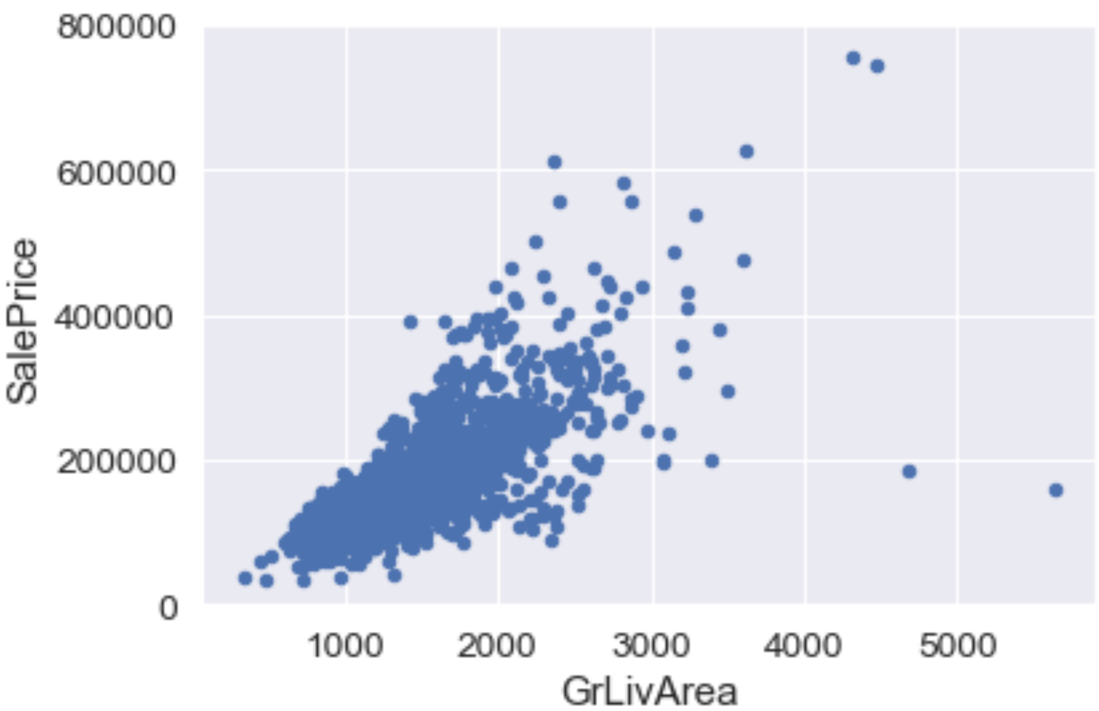
```
#bivarate analysis saleprice/grlivarea
#selecting best column for it
k = 11 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index

var='GrLivArea'
data=pd.concat([df_dummy['SalePrice'],df_dummy[var]],axis=1)
data.plot.scatter(x=var,y="SalePrice",ylim=(0,800000))
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c23713e50>
```



identify outliers

In [125]:

```
df_dummy['GrLivArea'].describe()
```

Out[125]:

```
count    1460.000000
mean     1515.463699
std       525.480383
min       334.000000
25%      1129.500000
50%      1464.000000
75%      1776.750000
max      5642.000000
Name: GrLivArea, dtype: float64
```

Remove outlier

In [126]:

```
# df_train.sort_values(by = 'GrLivArea', ascending = False)[:2]
df_dummy.sort_values(by='GrLivArea',ascending=False)['GrLivArea'][:10]
```

Out[126]:

```
1298    5642
523     4676
1182    4476
691     4316
1169    3627
185     3608
304     3493
1268    3447
635     3395
769     3279
Name: GrLivArea, dtype: int64
```

In [ ]:

```

```

In [127]:

```
# df_dummy = df_dummy.drop(df_dummy[df_dummy['Id'] == 1298].index)
# df_dummy = df_dummy.drop(df_dummy[df_dummy['Id'] == 523].index)
df_dummy=df_dummy.drop(1298,axis=0)
df_dummy=df_dummy.drop(523,axis=0)
```

In [128]:

```
df_dummy.sort_values(by='GrLivArea',ascending=False)['GrLivArea'][:10]
```

Out[128]:

```
1182    4476
691     4316
1169    3627
185     3608
304     3493
1268    3447
635     3395
769     3279
1353    3238
496     3228
Name: GrLivArea, dtype: int64
```

In [129]:

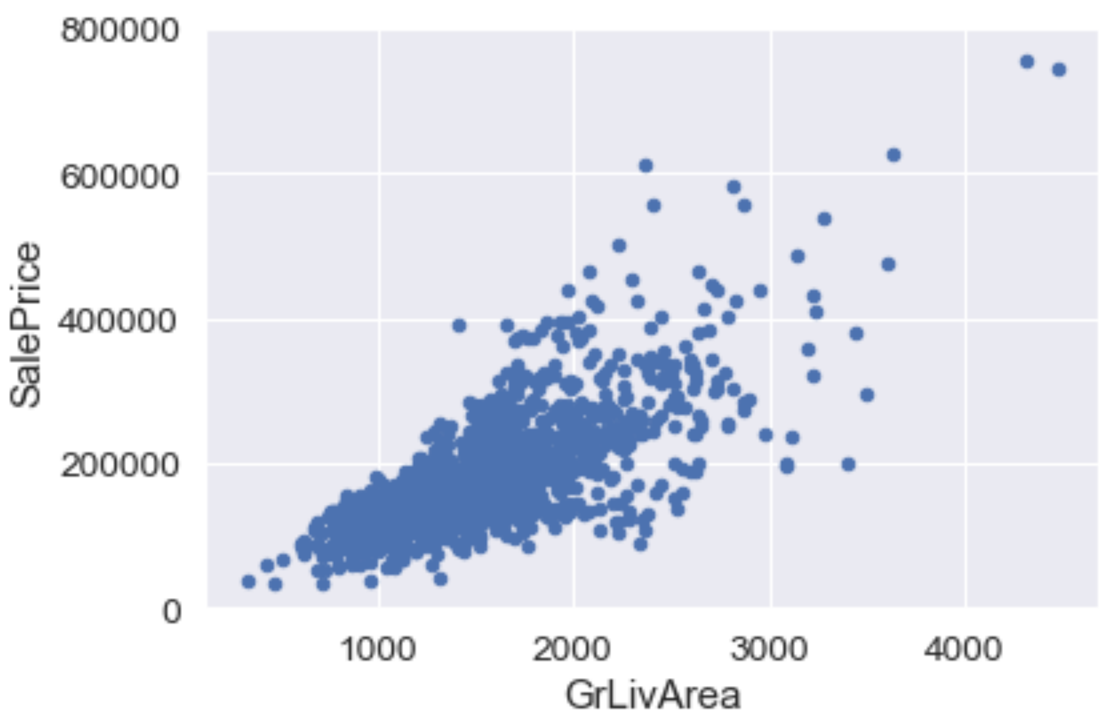
```
k = 11 #number of variables for heatmap
cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index

var='GrLivArea'
data=pd.concat([df_dummy['SalePrice'],df_dummy[var]],axis=1)
data.plot.scatter(x=var,y="SalePrice",ylim=(0,800000))
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[129]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c23784fd0>
```



2 LotArea

In [130]:

```
# df_dummy[df_dummy['Id'] == 335]
```

2 feauture not decided

In [131]:

```
df_dummy.columns
```

Out[131]:

```
Index(['Id', 'MSZoning', 'LotFrontage', 'LotArea', 'Alley', 'Neighborh
ood',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'Exterior1st', 'MasVnrArea', 'BsmtQual', 'BsmtExposure', 'BsmtF
inType1',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Centra
lAir',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'KitchenQu
al',
       'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'Woo
dDeckSF',
       'OpenPorchSF', 'PoolArea', 'PoolQC', 'Fence', 'MoSold', 'SaleCo
ndition',
       'SalePrice'],
      dtype='object')
```

In [132]:

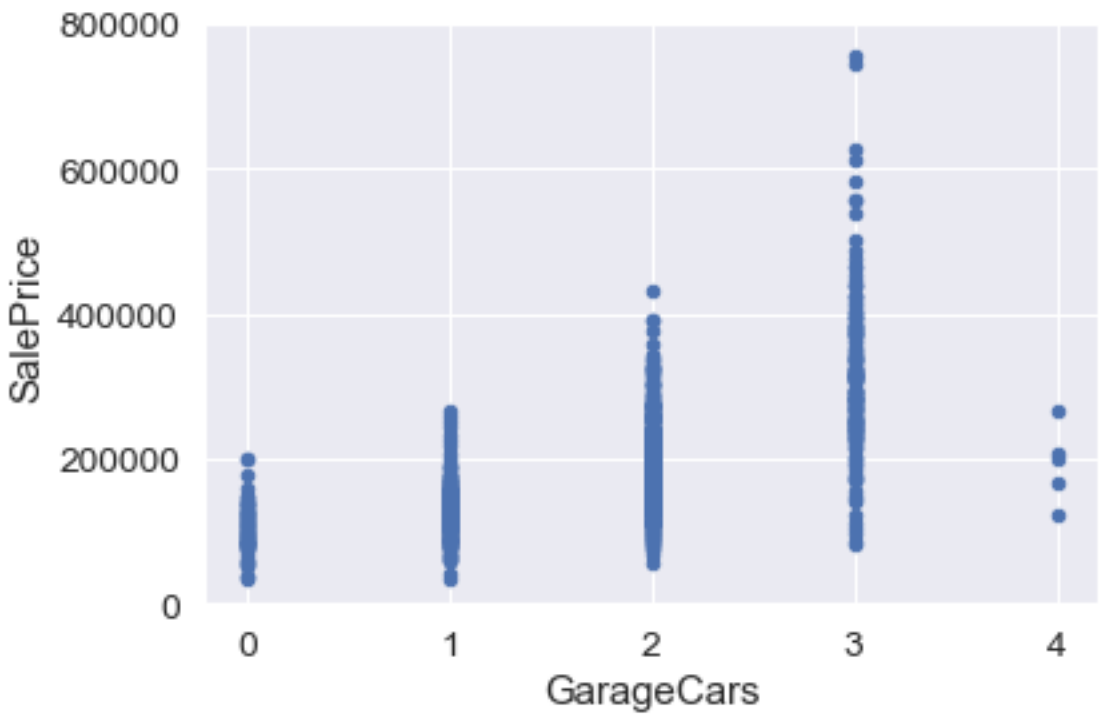
```
# k = 11 #number of variables for heatmap
# cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index

var='GarageCars'
data=pd.concat([df_dummy['SalePrice'],df_dummy[var]],axis=1)
data.plot.scatter(x=var,y="SalePrice",ylim=(0,800000))
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[132]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c237b9a50>
```



In [133]:

```
#identify outliers
#i want to remove data points greater then 600000 <800000
df_dummy.groupby(['GarageCars','SalePrice']).filter(lambda x: (x['SalePrice']<40000(
```

Out[133]:

	Id	MSZoning	LotFrontage	LotArea	Alley	Neighborhood	OverallQual	OverallCond	Ye
420	421	4.0	78.0	7060	1.0	11.0	7	5	
747	748	4.0	65.0	11700	1.0	17.0	7	7	
1190	1191	3.0	51.0	32463	0.0	11.0	4	4	
1340	1341	3.0	70.0	8294	0.0	12.0	4	5	
1350	1351	3.0	91.0	11643	1.0	12.0	5	5	

5 rows × 41 columns

In [134]:

```
# Remove outlier
# df_dummy=df_dummy.drop(420,axis=0)
# df_dummy=df_dummy.drop(747,axis=0)
# df_dummy=df_dummy.drop(1340,axis=0)
# df_dummy=df_dummy.drop(1350,axis=0)
# df_dummy=df_dummy.drop(1190,axis=0)
```

In [135]:

```
# df_dummy['OpenPorchSF'][691],df_dummy['OpenPorchSF'][1182]
```

In [136]:

```
# df_dummy['GarageArea'][1061]=np.mean(df_dummy['GarageArea'])
# df_dummy['GarageArea'][1190]=np.mean(df_dummy['GarageArea'])
# df_dummy['GarageArea'][1061],df_dummy['GarageArea'][1190]
```

In [ ]:

```

```

In [137]:

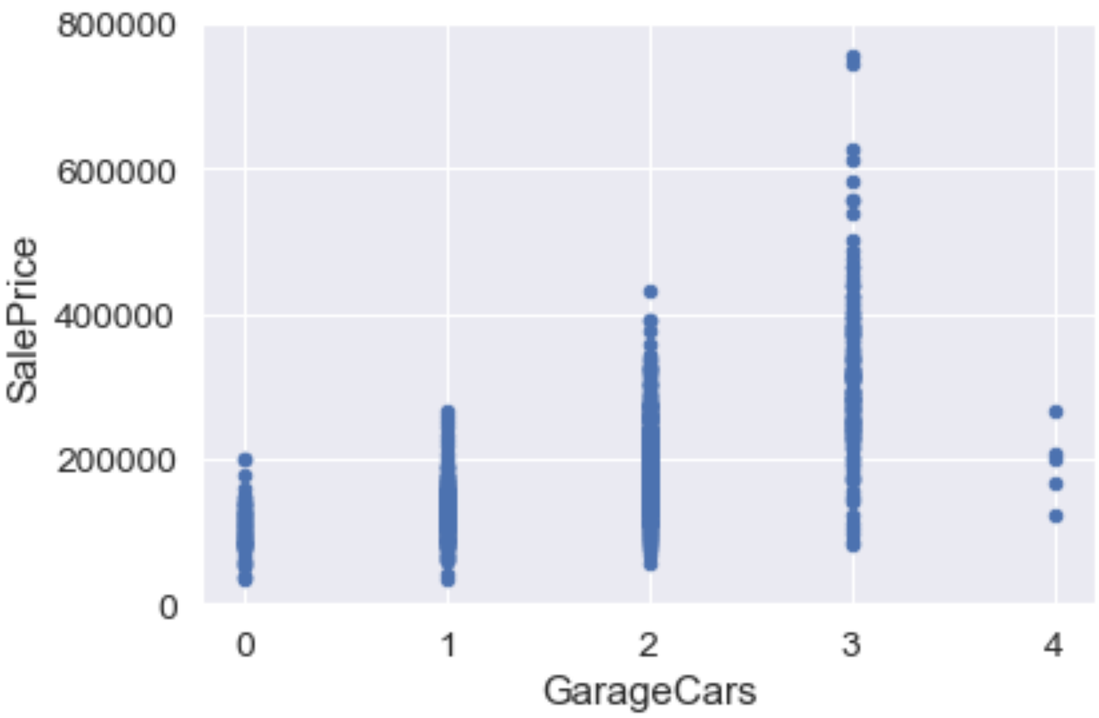
```
# k = 11 #number of variables for heatmap
# cols = corrmat.nlargest(k, 'SalePrice')['SalePrice'].index

var='GarageCars'
data=pd.concat([df_dummy['SalePrice'],df_dummy[var]],axis=1)
data.plot.scatter(x=var,y="SalePrice",ylim=(0,800000))
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[137]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2399fdd0>
```



identify outliers

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [138]:

```
scaled_dataset = StandardScaler().fit_transform(df_dummy)
scaled_dataset=pd.DataFrame(scaled_dataset,columns=df_dummy.columns)


X=scaled_dataset.drop('SalePrice',axis=1)
y=scaled_dataset['SalePrice']
```

In [139]:

```

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
len(x_train.columns),x_train.columns
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[139]:

```
((1166, 40), (292, 40), (1166,), (292,))
```

In [140]:

```
%%time
run_randomForest(x_train, y_train,x_test, y_test)
```

```
Mean Squer Error 0.06266762134413409
sqrt of Mean Sauer Error  0.25033501821386095
CPU times: user 370 ms, sys: 2.83 ms, total: 373 ms
Wall time: 372 ms
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [141]:

```

```

Introduction

```
Purpose of this Epxirment is to play with Missing data.
In cotagorical data and Numeric.
There are different ways we can deal with catagorical data as given below
1."Rmove Rows or columns" :Remove the complet row but problem is loss of inf
ormation
2."Replace with Most Frequenet":Put most frequently used value in that parti
cular colum
but problem is imblancing data
3."Apply classficiation algorithem" which is quite good then option "1" and
"2"
4."Apply clustering algorithem" which is consider a ver good solution
keep in mind all these could only apply to catagorical data
```

```
Moreover,Object dtype should be convert to numbers,explore data set of type=
=object
and Apply Label Encoding. There are two type of encoding.
Label Encoding and One hot encoding.
Label encoding workds good in case of Regression model while
one hot encoding works good incase of classification
```

Reading Train and Test File

In [2]:

```
```

checking shape

In [3]:

```
```

Out[3]:

```
((1460, 81), (1459, 80))
```

checking data types

In [4]:

```
```

```
object     43
int64      35
float64     3
dtype: int64
object     43
int64      26
float64    11
dtype: int64
```

checking null sapratly

Catagorical

In [5]:

```

```

In [6]:

```

```

In [7]:

```

```

Out[7]:

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

In [8]:

Out[8]:

```
(33, Index(['MSZoning', 'LotFrontage', 'Alley', 'Utilities', 'Exterior
1st',
        'Exterior2nd', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'BsmtCo
nd',
        'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2',
        'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'BsmtFullBath',
        'BsmtHalfBath', 'KitchenQual', 'Functional', 'FireplaceQu',
        'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'Ga
rageArea',
        'GarageQual', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeature',
        'SaleType'],
       dtype='object'))
```

In [ ]:

In [9]:

```
['LotFrontage', 'Alley', 'MasVnrType', 'MasVnrArea', 'BsmtQual', 'Bsmt
Cond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Electrical', '
FireplaceQu', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageQual
', 'GarageCond', 'PoolQC', 'Fence', 'MiscFeature']
Feature :  LotFrontage
Type    :  float64
[65.0, 80.0, 68.0, 60.0, 84.0, 85.0, 75.0, nan, 51.0, 50.0, 70.0, 91.0
, 72.0, 66.0]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  18
Feature :  Alley
Type    :  object
[nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  17
Feature :  MasVnrType
Type    :  object
['BrkFace', 'None', 'Stone']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  16
Feature :  MasVnrArea
```

```
Type    :  float64
[196.0, 0.0, 162.0, 350.0, 186.0, 240.0, 286.0, 306.0, 212.0, 180.0]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  15
Feature :  BsmtQual
Type    :  object
['Gd', 'TA', 'Ex', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  14
Feature :  BsmtCond
Type    :  object
['TA', 'Gd', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  13
Feature :  BsmtExposure
Type    :  object
['No', 'Gd', 'Mn', 'Av', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  12
Feature :  BsmtFinType1
Type    :  object
['GLQ', 'ALQ', 'Unf', 'Rec', 'BLQ', nan, 'LwQ']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  11
Feature :  BsmtFinType2
Type    :  object
['Unf', 'BLQ', nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  10
Feature :  Electrical
Type    :  object
['SBrkr', 'FuseF', 'FuseA']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  9
Feature :  FireplaceQu
Type    :  object
[nan, 'TA', 'Gd', 'Fa']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  8
Feature :  GarageType
Type    :  object
['Attchd', 'Detchd', 'BuiltIn', 'CarPort']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  7
```

```
Feature :  GarageYrBlt
Type    :  float64
[2003.0, 1976.0, 2001.0, 1998.0, 2000.0, 1993.0, 2004.0, 1973.0, 1931.
0, 1939.0, 1965.0, 2005.0, 1962.0, 2006.0, 1960.0, 1991.0, 1970.0, 196
7.0, 1958.0]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  6
Feature :  GarageFinish
Type    :  object
['RFn', 'Unf', 'Fin']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  5
Feature :  GarageQual
Type    :  object
['TA', 'Fa', 'Gd']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  4
Feature :  GarageCond
Type    :  object
['TA']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  3
Feature :  PoolQC
Type    :  object
[nan]
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  2
Feature :  Fence
Type    :  object
[nan, 'MnPrv', 'GdWo', 'GdPrv']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  1
Feature :  MiscFeature
Type    :  object
[nan, 'Shed']
RangeIndex(start=0, stop=1460, step=1)
Done
Remaining feature :  0
```

Now Lets see how much Nan column we have in full dataset

In [10]:

Out[10]:

```
Series([], dtype: int64)
```

In [11]:

check info

data types

In [12]:

Out[12]:

```
int64      35
object     27
float64    19
dtype: int64
```

float

In [13]:

Out[13]:

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

In [14]:

```
LotFrontage [ 65.   80.   68.   60.   84.   85.   75.   35.   51.   50.   70.   9
1.   72.   66.
 101.   57.   44. 110.   98.   47. 108. 112.   74. 115.   31.   61.   48.   33.
]
Alley [1. 0.]
MasVnrType [1. 2. 3.]
MasVnrArea [196.    0. 162. 350. 186. 240. 286. 306. 212. 180. 380. 281
. 640. 200.
 246. 132. 650. 101. 412.]
BsmtQual [2. 3. 0.]
BsmtCond [3. 1.]
BsmtExposure [3. 1. 2. 0.]
BsmtFinType1 [2. 0. 5. 4. 1. 3.]
BsmtFinType2 [5. 1. 0. 4. 3.]
Electrical [4. 1. 0. 2.]
FireplaceQu [4. 2. 1. 3. 0.]
GarageType [1. 5. 3. 4.]
GarageYrBlt [2003. 1976. 2001. 1998. 2000. 1993. 2004. 1973. 1931. 193
9. 1965. 2005.
```

Object

In [15]:

Out[15]:

```
Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condit
ion2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st
',
       'Exterior2nd', 'ExterQual', 'ExterCond', 'Foundation', 'Heating
',
       'HeatingQC', 'CentralAir', 'KitchenQual', 'Functional', 'PavedD
rive',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

In [16]:

```
MSZoning ['RL' 'RM' 'C (all)' 'FV']
Street ['Pave']
LotShape ['Reg' 'IR1' 'IR2']
LandContour ['Lvl' 'Bnk']
Utilities ['AllPub']
LotConfig ['Inside' 'FR2' 'Corner' 'CulDSac']
LandSlope ['Gtl']
Neighborhood ['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somer
st' 'NWAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber']
Condition1 ['Norm' 'Feedr' 'PosN' 'Artery' 'RRAe']
Condition2 ['Norm' 'Artery' 'RRNn']
BldgType ['1Fam' '2fmCon' 'Duplex' 'TwnhsE']
HouseStyle ['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer']
RoofStyle ['Gable' 'Hip' 'Gambrel']
RoofMatl ['CompShg']
Exterior1st ['VinylSd' 'MetalSd' 'Wd Sdng' 'HdBoard' 'BrkFace' 'WdShin
g' 'CemntBd'
```

Int

In [17]:

Out[17]:

```
Index(['Id', 'MSSubClass', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtU
nfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivAr
ea',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'Bedroo
mAbvGr',
       'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars',
       'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3S
snPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'Sale
Price'],
      dtype='object')
```

In [18]:

```

```

```
Id [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24
 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48
 49 50]
MSSubClass [ 60  20  70  50 190  45  90 120  30  85]
LotArea [ 8450  9600 11250  9550 14260 14115 10084 10382  6120  7420 1
1200 11924
 12968 10652 10920 11241 10791 13695  7560 14215  7449  9742  4224  82
46
 14230  7200 11478 16321  6324  8500  8544 11049 10552  7313 13418 108
59
  8532  7922  6040  8658 16905  9180  9200  7945  7658 12822 11096  44
56
  7742]
OverallQual [7 6 8 5 9 4]
OverallCond [5 8 6 7 4]
YearBuilt [2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 1965 2005
1962 2006
```

In [ ]:

```

```

In [ ]:

```

```

Transoformation None 'nan' Catagorical Feature

In [19]:

```
[3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 4.0, 3.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, 3.0, 3.0]
[3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[4.0, 2.0, 4.0, 0.0, 2.0, 4.0, 4.0, 0.0, 4.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[5.0, 24.0, 5.0, 6.0, 15.0, 11.0, 21.0, 14.0, 17.0, 3.0]
[2.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 4.0, 0.0, 0.0]
[2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
[5.0, 2.0, 5.0, 5.0, 5.0, 0.0, 2.0, 5.0, 0.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[12.0, 8.0, 12.0, 13.0, 12.0, 12.0, 12.0, 6.0, 3.0, 8.0]
[13.0, 8.0, 13.0, 15.0, 13.0, 13.0, 13.0, 6.0, 15.0, 8.0]
[2.0, 3.0, 2.0, 3.0, 2.0, 3.0, 2.0, 3.0, 3.0, 3.0]
[4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0, 4.0]
[2.0, 1.0, 2.0, 0.0, 2.0, 5.0, 2.0, 1.0, 0.0, 0.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0]
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
[2.0, 3.0, 2.0, 2.0, 2.0, 3.0, 2.0, 3.0, 3.0, 3.0]
[6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 6.0, 2.0, 6.0]
[2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
[8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0, 8.0]
[4.0, 4.0, 4.0, 0.0, 4.0, 4.0, 4.0, 4.0, 0.0, 4.0]
```

In [20]:

Take Dadtaset back to It's Position Local Training

In [81]:

In [82]:

In [83]:

Out[83]:

```
Series([], dtype: int64)
```

In [84]:

Out[84]:

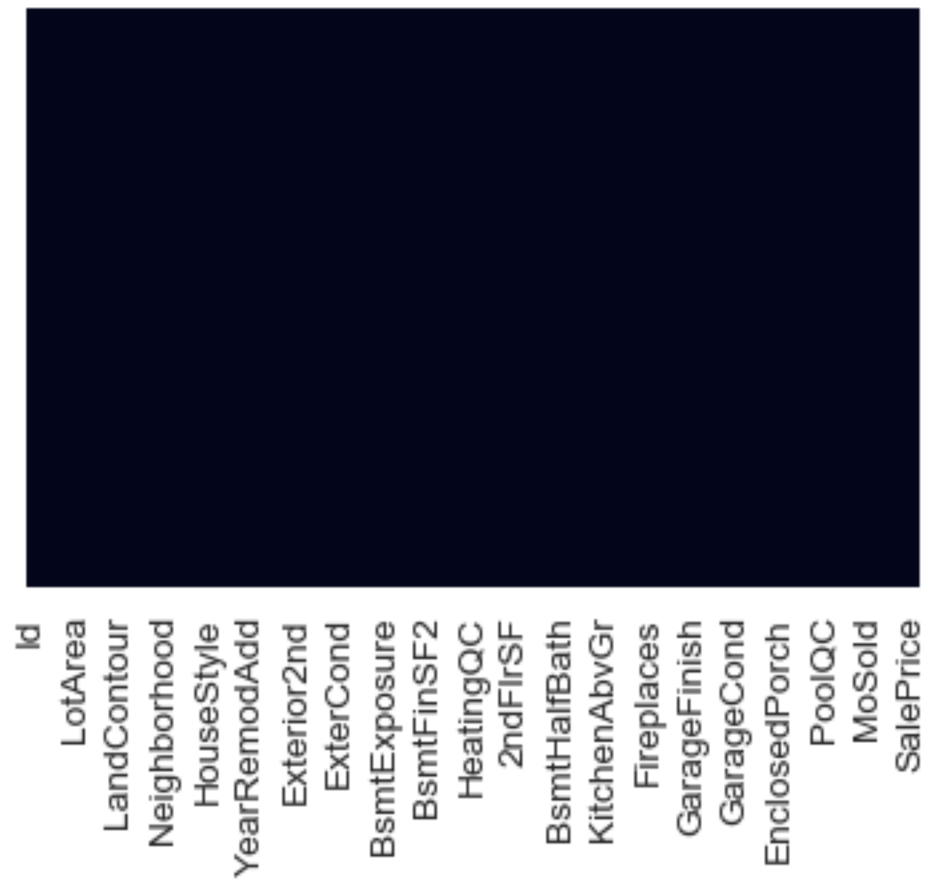
```
float64    46
int64      35
dtype: int64
```

Heatmap for Missing data

In [85]:

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c22af5190>
```



Build ML model and compare the performance of the selected feature

In [86]:

In [87]:

In [88]:

Out[88]:

```
((1168, 80), (292, 80), (1168,), (292,))
```

Estimation of coefficients of Linear Regression

In [89]:

In [90]:

Out[90]:

```
SelectFromModel(estimator=GradientBoostingRegressor(alpha=0.9,

criterion='friedman_mse',
                                                    init=None,
                                                    learning_rate=0.1,
                                                    loss='ls', max_dep
th=3,
                                                    max_features=None,

max_leaf_nodes=None,

min_impurity_decrease=0.0,

min_impurity_split=None,

min_samples_leaf=1,

min_samples_split=2,
```

In [91]:

```

```

Out[91]:

```
array([False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False,  True, False,
       False, False,  True, False, False, False, False,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False])
```

In [92]:

```

```

In [93]:

```

```

In [94]:

```

```

In [95]:

```

```

In [96]:

```

```

Out[96]:

```
Index(['Alley', 'OverallQual', 'BsmtQual', 'BsmtFinSF1', 'TotalBsmtSF'
,
       '1stFlrSF', 'GrLivArea', 'GarageCars', 'PoolQC'],
      dtype='object')
```

In [97]:

```

```

In [98]:

```

```

Out[98]:

```
(292, 9)
```

In [99]:

```
Mean Squer Error 0.09754349139496073
sqrt of Mean Sauer Error  0.31231953412324487
CPU times: user 148 ms, sys: 2.59 ms, total: 151 ms
Wall time: 150 ms
```

In [100]:

```
Mean Squer Error 0.08520220717249559
sqrt of Mean Sauer Error  0.29189417118622907
CPU times: user 571 ms, sys: 3.43 ms, total: 574 ms
Wall time: 574 ms
```

Feature selection by feature importance of random forest classifier

In [101]:

Out[101]:

```
array([False, False, False, False, False, False,  True, False, False,
       False, False, False, False, False, False, False, False,  True,
       False, False, False, False, False, False, False, False, False,
       False, False, False,  True, False, False, False,  True, False,
       False, False,  True, False, False, False, False,  True, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
       False, False, False, False, False, False, False, False, False,
        True, False, False, False, False, False, False, False])
```

In [102]:

In [103]:

Out[103]:

```
Index(['Alley', 'OverallQual', 'BsmtQual', 'BsmtFinSF1', 'TotalBsmtSF'
,
       '1stFlrSF', 'GrLivArea', 'GarageCars', 'PoolQC'],
      dtype='object')
```

In [104]:

```
```

Out[104]:

```
0.0125
```

In [105]:

```
```

Out[105]:

```
array([6.99053604e-04, 1.74739082e-04, 1.59981321e-03, 2.00374394e-03,
       9.04837490e-03, 0.00000000e+00, 3.41084612e-02, 9.24795432e-05,
       2.31137563e-06, 0.00000000e+00, 9.51234332e-05, 3.25017413e-04,
       1.31785755e-03, 4.40616875e-04, 1.17105694e-04, 6.70483010e-06,
       1.29089522e-04, 1.67017400e-01, 3.80087874e-03, 4.79768165e-03,
       3.55135926e-03, 6.73815747e-04, 8.68023139e-04, 6.21644815e-04,
       1.08181502e-04, 1.78969472e-04, 2.09792801e-03, 2.71241784e-04,
       9.13068303e-05, 1.05281982e-04, 1.34343206e-02, 2.26613634e-04,
       1.57999323e-03, 1.00879337e-03, 2.81043138e-02, 1.28558992e-04,
       3.18350371e-04, 1.69085513e-03, 2.50867857e-02, 0.00000000e+00,
       8.76607781e-05, 2.79615487e-03, 0.00000000e+00, 1.85963391e-02,
       6.51883311e-03, 3.81501224e-04, 7.45942630e-02, 2.21474502e-04,
       1.39918773e-04, 5.20139578e-04, 5.97247587e-05, 5.12659878e-05,
       6.21102959e-05, 2.39012278e-03, 6.37678423e-03, 2.57741173e-04,
       2.44089169e-03, 6.08827145e-04, 3.20436155e-03, 5.54950285e-03,
       6.73793255e-04, 5.48286014e-02, 3.55289468e-03, 1.67729788e-04,
       0.00000000e+00, 1.66608742e-05, 1.23135826e-03, 7.17957398e-04,
       3.81044005e-04, 8.32511337e-05, 4.87027966e-04, 4.79548379e-04,
```

In [106]:

```
```

In [107]:

```
Mean Squer Error 0.09754349139496073
sqrt of Mean Sauer Error  0.31231953412324487
CPU times: user 143 ms, sys: 2.49 ms, total: 146 ms
Wall time: 144 ms
```

Recursive Feature Elimination (RFE)

In [108]:

Out[108]:

```
RFE(estimator=GradientBoostingRegressor(alpha=0.9, criterion='friedman
_mse',
                                        init=None, learning_rate=0.1,
loss='ls',
                                        max_depth=3, max_features=None
,
                                        max_leaf_nodes=None,
                                        min_impurity_decrease=0.0,
                                        min_impurity_split=None,
                                        min_samples_leaf=1, min_sample
s_split=2,
                                        min_weight_fraction_leaf=0.0,
                                        n_estimators=200, n_iter_no_ch
ange=None,
                                        presort='auto', random_state=0
,
                                        subsample=1.0, tol=0.0001,
                                        validation_fraction=0.1, verbo
```

In [ ]:

In [109]:

Out[109]:

```
array([ True, False,  True,  True,  True, False,  True, False, False,
       False, False, False,  True, False, False, False, False,  True,
        True,  True,  True, False, False,  True, False, False,  True,
       False, False, False,  True, False,  True,  True,  True, False,
        True,  True,  True, False, False,  True, False,  True,  True,
        True,  True, False, False, False, False, False, False,  True,
        True, False,  True,  True,  True,  True,  True,  True,  True,
       False, False, False,  True,  True, False, False, False,  True,
        True,  True, False, False,  True, False, False,  True])
```

In [110]:

In [111]:

Out[111]:

```
Index(['Id', 'MSZoning', 'LotFrontage', 'LotArea', 'Alley', 'Neighborh
ood',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'Exterior1st', 'MasVnrArea', 'BsmtQual', 'BsmtExposure', 'BsmtF
inType1',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Centra
lAir',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'KitchenQu
al',
       'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'Woo
dDeckSF',
       'OpenPorchSF', 'PoolArea', 'PoolQC', 'Fence', 'MoSold',
       'SaleCondition'],
      dtype='object')
```

In [112]:

Out[112]:

```
40
```

In [113]:

In [114]:

```
Mean Squer Error 0.07624449279990299
sqrt of Mean Sauer Error  0.27612405328022943
CPU times: user 370 ms, sys: 2.93 ms, total: 373 ms
Wall time: 371 ms
```

In [115]:

In [116]:

Feature Visualization

prepare dataset

In [117]:

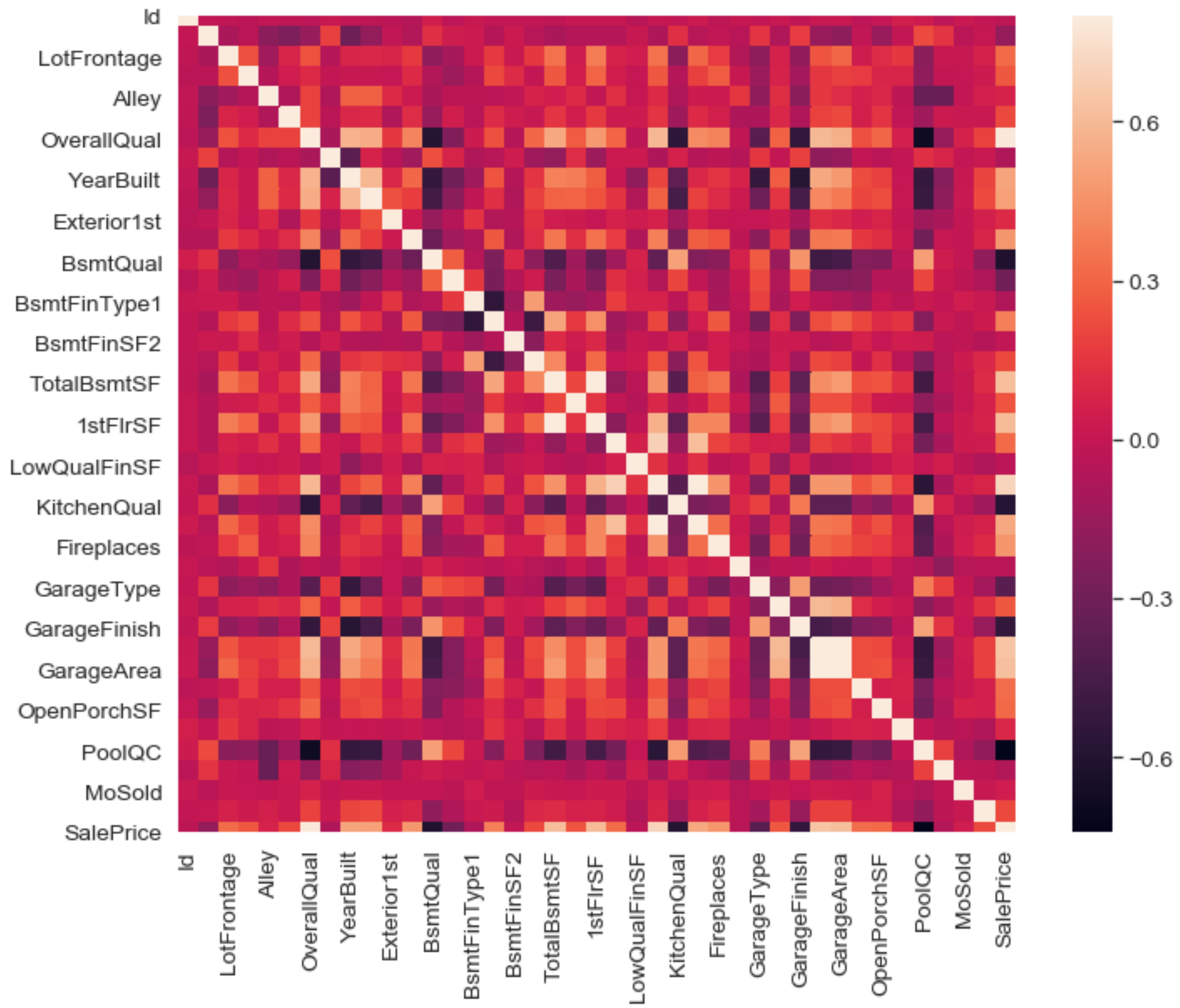
In [118]:

HeatMape

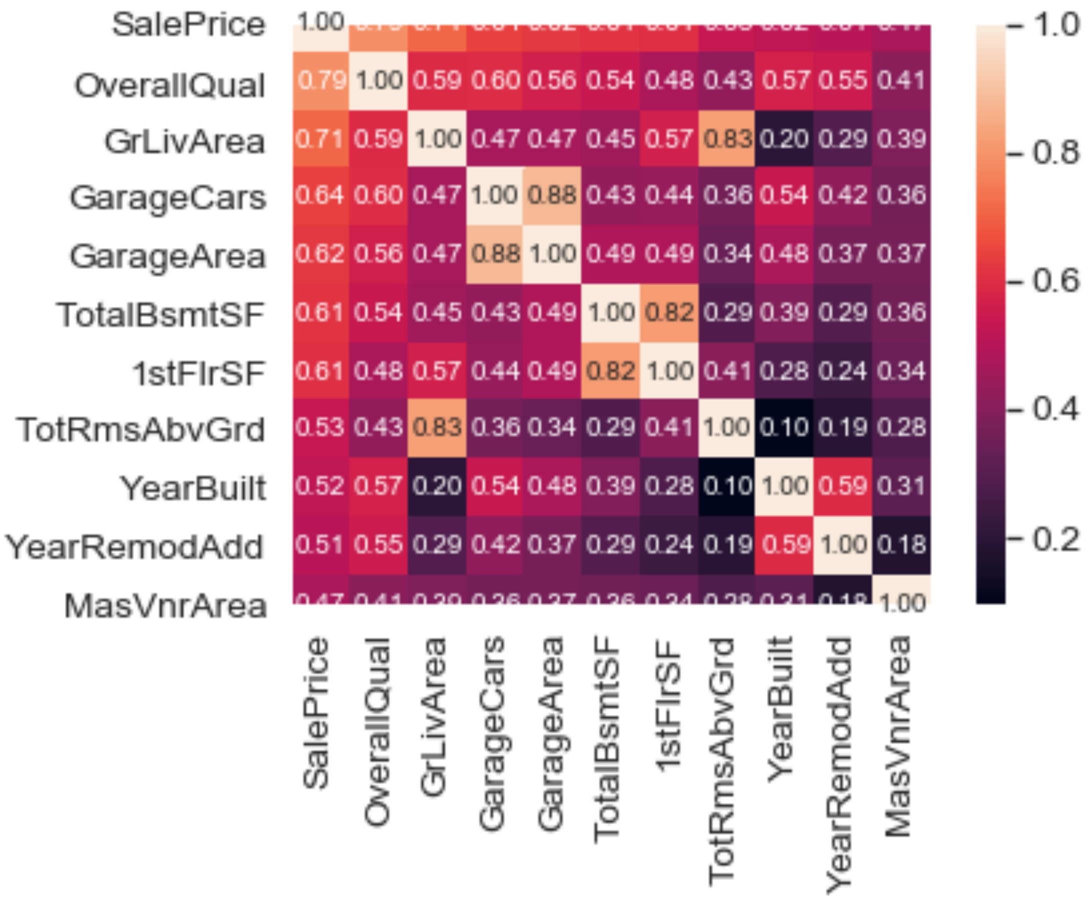
In [119]:

Out[119]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c22b85290>



In [120]:



SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
TotRmsAbvGrd
YearBuilt
YearRemodAdd
MasVnrArea
1.0
0.8
0.6
0.4
0.2

```
According to our crystal ball, these are the variables most correlated with
'SalePrice'. My thoughts on this:

'OverallQual', 'GrLivArea' and 'TotalBsmtSF' are strongly correlated with 'S
alePrice'. Check!
'GarageCars' and 'GarageArea' are also some of the most strongly correlated
variables. However, as we discussed
in the last sub-point, the number of cars that fit into the garage is a cons
equence of the garage area.
'GarageCars' and 'GarageArea' are like twin brothers. You'll never be able t
o distinguish them. Therefore, we
just need one of these variables in our analysis (we can keep 'GarageCars' s
ince its correlation with
'SalePrice' is higher).
'TotalBsmtSF' and '1stFloor' also seem to be twin brothers. We can keep 'Tot
alBsmtSF' just to say that our
first
guess was right (re-read 'So... What can we expect?').
'FullBath'?? Really?
'TotRmsAbvGrd' and 'GrLivArea', twin brothers again. Is this dataset from Ch
ernobyl?
Ah... 'YearBuilt'... It seems that 'YearBuilt' is slightly correlated with '
SalePrice'. Honestly, it scares me to
think about 'YearBuilt' because I start feeling that we should do a little b
it of time-series analysis to get
this right. I'll leave this as a homework for you.
Let's proceed to the scatter plots.
```

Scatter plot

In [121]:

```

```

Dealing Missing Value

In [122]:

Out[122]:

	Total	Percent
MasVnrArea	0	0.0
YearRemodAdd	0	0.0
YearBuilt	0	0.0
TotRmsAbvGrd	0	0.0
1stFlrSF	0	0.0
TotalBsmtSF	0	0.0
GarageArea	0	0.0
GarageCars	0	0.0
GrLivArea	0	0.0
OverallQual	0	0.0
SalePrice	0	0.0

out Liars!

```
Outliers is also something that we should be aware of. Why? Because outliers
can markedly affect our models and can be a valuable source of information,
providing us insights about specific behaviours.

Outliers is a complex subject and it deserves more attention. Here, we'll ju
st do a quick analysis through the standard deviation of 'SalePrice' and a s
et of scatter plots.
```

Univeriate

In [123]:

```
out range (low) of the distribution :
[[-1.83870376]
 [-1.83352844]
 [-1.80092766]
 [-1.78329881]
 [-1.77448439]
 [-1.62337999]
 [-1.61708398]
 [-1.58560389]
 [-1.58560389]
 [-1.5731     ]]

outer range (high) of the distrbution :
[[3.82897043]
 [4.04098249]
 [4.49634819]
 [4.71041276]
 [4.73032076]
 [5.06214602]
```

Bivariate analysis

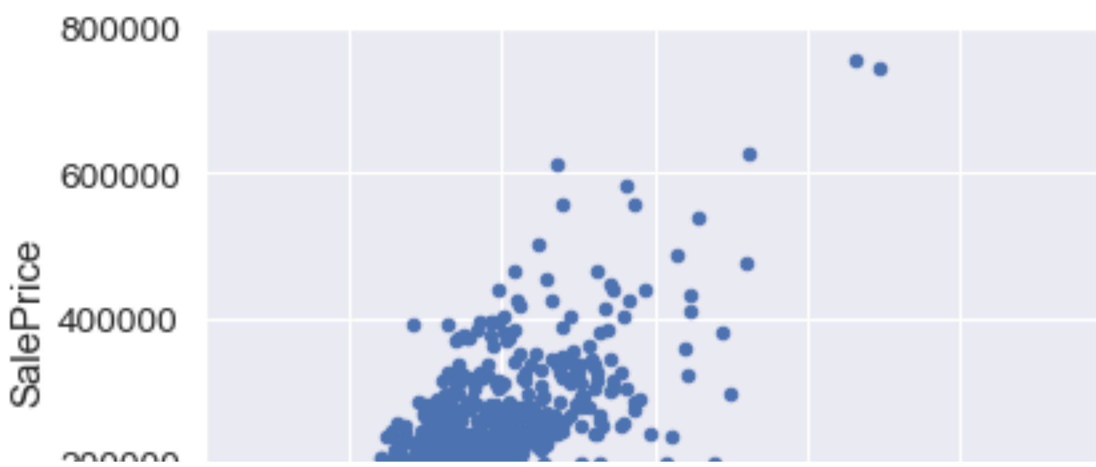
1 GrLivArea

In [124]:

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[124]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c23713e50>
```



identify outliers

In [125]:

Out[125]:

```
count    1460.000000
mean     1515.463699
std       525.480383
min       334.000000
25%      1129.500000
50%      1464.000000
75%      1776.750000
max      5642.000000
Name: GrLivArea, dtype: float64
```

Remove outlier

In [126]:

Out[126]:

```
1298    5642
523     4676
1182    4476
691     4316
1169    3627
185     3608
304     3493
1268    3447
635     3395
769     3279
Name: GrLivArea, dtype: int64
```

In [ ]:

In [127]:

In [128]:

Out[128]:

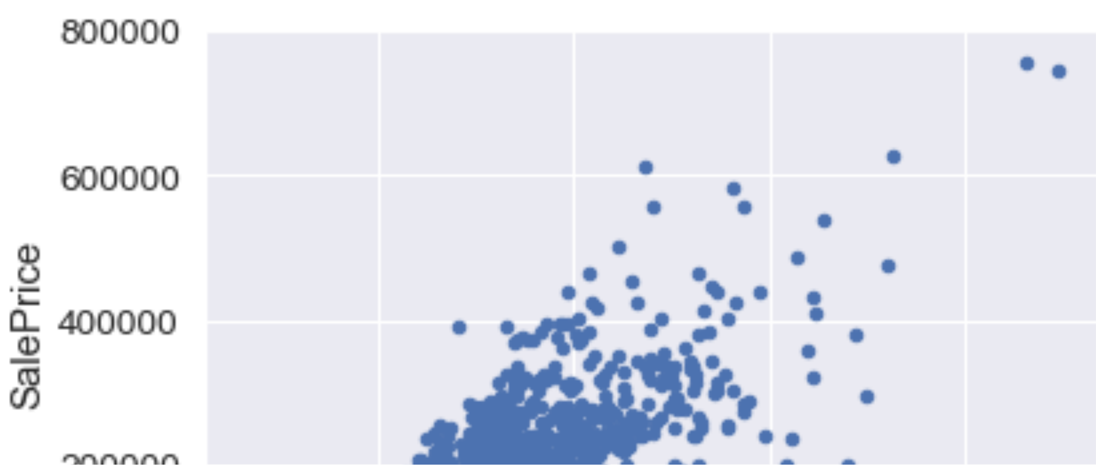
```
1182    4476
691     4316
1169    3627
185     3608
304     3493
1268    3447
635     3395
769     3279
1353    3238
496     3228
Name: GrLivArea, dtype: int64
```

In [129]:

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[129]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c23784fd0>
```



2 LotArea

In [130]:

2 feauture not decided

In [131]:

Out[131]:

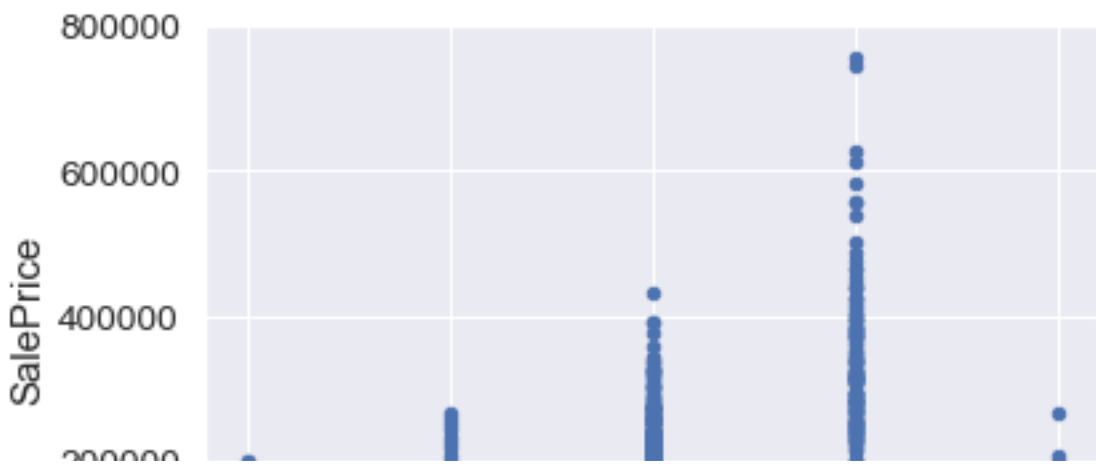
```
Index(['Id', 'MSZoning', 'LotFrontage', 'LotArea', 'Alley', 'Neighborh
ood',
       'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'Exterior1st', 'MasVnrArea', 'BsmtQual', 'BsmtExposure', 'BsmtF
inType1',
       'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Centra
lAir',
       '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'KitchenQu
al',
       'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'Woo
dDeckSF',
       'OpenPorchSF', 'PoolArea', 'PoolQC', 'Fence', 'MoSold', 'SaleCo
ndition',
       'SalePrice'],
      dtype='object')
```

In [132]:

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[132]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c237b9a50>
```



In [133]:

Out[133]:

	Id	MSZoning	LotFrontage	LotArea	Alley	Neighborhood	OverallQual	OverallCond	Ye
420	421	4.0	78.0	7060	1.0	11.0	7	5	
747	748	4.0	65.0	11700	1.0	17.0	7	7	
1190	1191	3.0	51.0	32463	0.0	11.0	4	4	
1340	1341	3.0	70.0	8294	0.0	12.0	4	5	
1350	1351	3.0	91.0	11643	1.0	12.0	5	5	

5 rows × 41 columns

In [134]:

In [135]:

In [136]:

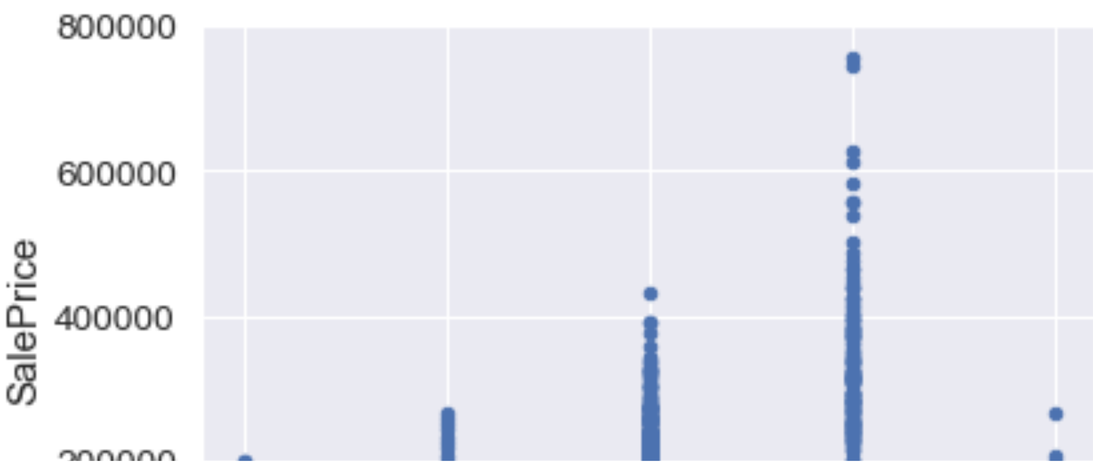
In [ ]:

In [137]:

```
'c' argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its len
gth matches with 'x' & 'y'.  Please use a 2-D array with a single row
if you really want to specify the same RGB or RGBA value for all point
s.
```

Out[137]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c2399fdd0>
```



identify outliers

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [138]:

In [139]:

Out[139]:

```
((1166, 40), (292, 40), (1166,), (292,))
```

In [140]:

```
Mean Squer Error 0.06266762134413409
sqrt of Mean Sauer Error  0.25033501821386095
CPU times: user 370 ms, sys: 2.83 ms, total: 373 ms
Wall time: 372 ms
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: