In [194]:
```
import os
import warnings
warnings.simplefilter(action = 'ignore', category=FutureWarning)
warnings.filterwarnings('ignore')
def ignore_warn(*args, **kwargs):
    pass

warnings.warn = ignore_warn #ignore annoying warning (from sklearn and seab

import numpy as np
import pandas as pd
pd.set_option('display.float_format', lambda x: '{:.3f}'.format(x)) #Limiti
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style="ticks", color_codes=True, font_scale=1.5)
color = sns.color_palette()
sns.set_style('darkgrid')
import mpl_toolkits
from mpl_toolkits.mplot3d import Axes3D
import pylab
from scipy import stats
from scipy.stats import skew, norm, probplot, boxcox
from scipy.special import boxcox1p
from patsy import dmatrices
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import RobustScaler, PolynomialFeatures, Standar
from sklearn.feature_selection import f_regression, mutual_info_regression,
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
from sklearn.feature_extraction import FeatureHasher
from sklearn.decomposition import PCA, KernelPCA
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, c
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import LinearRegression, OrthogonalMatchingPursui
from sklearn.linear_model import SGDRegressor, PassiveAggressiveRegressor,
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegress
import xgboost as xgb
from xgboost import XGBRegressor, plot_importance
#import lightgbm as lgb

from sklearn.model_selection import GridSearchCV, cross_val_score, KFold, c
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_sco
```

In [195]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
#Save the 'Id' column
train_ID = train['Id']
test_ID = test['Id']

#Now drop the  'Id' colum since it's unnecessary for  the prediction proces
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)

train.rename(columns={'3SsnPorch':'TSsnPorch'}, inplace=True)
test.rename(columns={'3SsnPorch':'TSsnPorch'}, inplace=True)

test['SalePrice'] = 0
```

In [196]:
```
def rstr(df, pred=None):
    obs = df.shape[0]
    types = df.dtypes
    counts = df.apply(lambda x: x.count())
    uniques = df.apply(lambda x: [x.unique()])
    nulls = df.apply(lambda x: x.isnull().sum())
    distincts = df.apply(lambda x: x.unique().shape[0])
    missing_ration = (df.isnull().sum()/ obs) * 100
    skewness = df.skew()
    kurtosis = df.kurt()
    print('Data shape:', df.shape)

    if pred is None:
        cols = ['types', 'counts', 'distincts', 'nulls', 'missing ration',
        str = pd.concat([types, counts, distincts, nulls, missing_ration, u

    else:
        corr = df.corr()[pred]
        str = pd.concat([types, counts, distincts, nulls, missing_ration, u
        corr_col = 'corr '  + pred
        cols = ['types', 'counts', 'distincts', 'nulls', 'missing_ration',

    str.columns = cols
    dtypes = str.types.value_counts()
    print('_____________________________\nData types:\n',str.types.value_coun
    print('_____________________________')
    return str
```

In [197]:
```
details = rstr(train, 'SalePrice')
display(details.sort_values(by='corr SalePrice', ascending=False))
```

```
Data shape: (1460, 80)

_____________________________
Data types:
 object     43
int64      34
float64     3
Name: types, dtype: int64

_____________________________
```

	types	counts	distincts	nulls	missing_ration	uniques	skewness	kurtosis	co SalePric
SalePrice	int64	1460	663	0	0.000	[[208500, 181500, 223500, 140000, 250000, 1430...	1.883	6.536	1.0(
OverallQual	int64	1460	10	0	0.000	[[7, 6, 8, 5, 9, 4, 10, 3, 1, 2]]	0.217	0.096	0.7!
GrLivArea	int64	1460	861	0	0.000	[[1710, 1262, 1786, 1717, 2198, 1362, 1694, 20...	1.367	4.895	0.7(
GarageCars	int64	1460	5	0	0.000	[[2, 3, 1, 0, 4]]	-0.343	0.221	0.6
GarageArea	int64	1460	441	0	0.000	[[548, 460, 608, 642, 836, 480, 636, 484, 468,...	0.180	0.917	0.6
...	...	...	...	...	...	...	...	...	
PoolQC	object	7	4	1453	99.521	[[nan, Ex, Fa, Gd]]	nan	nan	n
Fence	object	281	5	1179	80.753	[[nan, MnPrv, GdWo, GdPrv, MnWw]]	nan	nan	n
MiscFeature	object	54	5	1406	96.301	[[nan, Shed, Gar2, Othr, TenC]]	nan	nan	n

	types	counts	distincts	nulls	missing_ration	uniques	skewness	kurtosis	co SalePric
SaleType	object	1460	9	0	0.000	[[WD, New, COD, ConLD, ConLI, CWD, ConLw, Con,...	nan	nan	na
SaleCondition	object	1460	6	0	0.000	[[Normal, Abnorml, Partial, AdjLand, Alloca, F...	nan	nan	na

80 rows × 9 columns

In [198]:
```
display(train.describe().transpose())
```

	count	mean	std	min	25%	50%	75%	
MSSubClass	1460.000	56.897	42.301	20.000	20.000	50.000	70.000	
LotFrontage	1201.000	70.050	24.285	21.000	59.000	69.000	80.000	
LotArea	1460.000	10516.828	9981.265	1300.000	7553.500	9478.500	11601.500	21
OverallQual	1460.000	6.099	1.383	1.000	5.000	6.000	7.000	
OverallCond	1460.000	5.575	1.113	1.000	5.000	5.000	6.000	
YearBuilt	1460.000	1971.268	30.203	1872.000	1954.000	1973.000	2000.000	
YearRemodAdd	1460.000	1984.866	20.645	1950.000	1967.000	1994.000	2004.000	
MasVnrArea	1452.000	103.685	181.066	0.000	0.000	0.000	166.000	
BsmtFinSF1	1460.000	443.640	456.098	0.000	0.000	383.500	712.250	
BsmtFinSF2	1460.000	46.549	161.319	0.000	0.000	0.000	0.000	
BsmtUnfSF	1460.000	567.240	441.867	0.000	223.000	477.500	808.000	
TotalBsmtSF	1460.000	1057.429	438.705	0.000	795.750	991.500	1298.250	
1stFlrSF	1460.000	1162.627	386.588	334.000	882.000	1087.000	1391.250	
2ndFlrSF	1460.000	346.992	436.528	0.000	0.000	0.000	728.000	
LowQualFinSF	1460.000	5.845	48.623	0.000	0.000	0.000	0.000	
GrLivArea	1460.000	1515.464	525.480	334.000	1129.500	1464.000	1776.750	
BsmtFullBath	1460.000	0.425	0.519	0.000	0.000	0.000	1.000	
BsmtHalfBath	1460.000	0.058	0.239	0.000	0.000	0.000	0.000	
FullBath	1460.000	1.565	0.551	0.000	1.000	2.000	2.000	
HalfBath	1460.000	0.383	0.503	0.000	0.000	0.000	1.000	
BedroomAbvGr	1460.000	2.866	0.816	0.000	2.000	3.000	3.000	
KitchenAbvGr	1460.000	1.047	0.220	0.000	1.000	1.000	1.000	
TotRmsAbvGrd	1460.000	6.518	1.625	2.000	5.000	6.000	7.000	
Fireplaces	1460.000	0.613	0.645	0.000	0.000	1.000	1.000	
GarageYrBlt	1379.000	1978.506	24.690	1900.000	1961.000	1980.000	2002.000	
GarageCars	1460.000	1.767	0.747	0.000	1.000	2.000	2.000	
GarageArea	1460.000	472.980	213.805	0.000	334.500	480.000	576.000	
WoodDeckSF	1460.000	94.245	125.339	0.000	0.000	0.000	168.000	
OpenPorchSF	1460.000	46.660	66.256	0.000	0.000	25.000	68.000	
EnclosedPorch	1460.000	21.954	61.119	0.000	0.000	0.000	0.000	
TSsnPorch	1460.000	3.410	29.317	0.000	0.000	0.000	0.000	
ScreenPorch	1460.000	15.061	55.757	0.000	0.000	0.000	0.000	
PoolArea	1460.000	2.759	40.177	0.000	0.000	0.000	0.000	
MiscVal	1460.000	43.489	496.123	0.000	0.000	0.000	0.000	1

	count	mean	std	min	25%	50%	75%	
MoSold	1460.000	6.322	2.704	1.000	5.000	6.000	8.000	
YrSold	1460.000	2007.816	1.328	2006.000	2007.000	2008.000	2009.000	
SalePrice	1460.000	180921.196	79442.503	34900.000	129975.000	163000.000	214000.000	75

In [199]:
```
fig = plt.figure(figsize=(20, 15))
sns.set(font_scale=1.5)

# (Corr= 0.817185) Box plot overallqual/salePrice
fig1 = fig.add_subplot(221); sns.boxplot(x='OverallQual', y='SalePrice', da

# (Corr= 0.700927) GrLivArea vs SalePrice plot
fig2 = fig.add_subplot(222);
sns.scatterplot(x = train.GrLivArea, y = train.SalePrice, hue=train.Overall

# (Corr= 0.680625) GarageCars vs SalePrice plot
fig3 = fig.add_subplot(223);
sns.scatterplot(x = train.GarageCars, y = train.SalePrice, hue=train.Overal

# (Corr= 0.650888) GarageArea vs SalePrice plot
fig4 = fig.add_subplot(224);
sns.scatterplot(x = train.GarageArea, y = train.SalePrice, hue=train.Overal

fig5 = plt.figure(figsize=(16, 8))
fig6 = fig5.add_subplot(121);
sns.scatterplot(y = train.SalePrice , x = train.TotalBsmtSF, hue=train.Over
fig7 = fig5.add_subplot(122);
sns.scatterplot(y = train.SalePrice, x = train['1stFlrSF'], hue=train.Overa

plt.tight_layout(); plt.show()
```

In [200]:
```
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(121)
sns.scatterplot(x = train.GrLivArea, y = train.SalePrice, ax = ax)

#Deleting outliers
train = train.drop(train[(train.GrLivArea>4000) & (train.SalePrice<300000)]

#Check the graphic again
ax = fig.add_subplot(122)
sns.scatterplot(x =train.GrLivArea, y = train.SalePrice, ax = ax)
plt.show()
```

In [201]:
```
sns.reset_defaults()
sns.set(style="ticks", color_codes=True)

df = train[['SalePrice', 'GrLivArea', 'TotRmsAbvGrd']]
df['GrLivAreaByRms'] = train.GrLivArea/train.TotRmsAbvGrd
df['GrLivArea_x_Rms'] = train.GrLivArea*train.TotRmsAbvGrd
fig = plt.figure(figsize=(20,5))
fig1 = fig.add_subplot(121); sns.regplot(x='GrLivAreaByRms', y='SalePrice',
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.GrLivAreaByRms.co
fig2 = fig.add_subplot(122); sns.regplot(x='GrLivArea_x_Rms', y='SalePrice'
plt.text(x=30000, y=100000, s='Correlation with SalePrice: {:1.4f}'.format(

print('                                                                    Ou
df = df.loc[df.GrLivArea_x_Rms<45000]
sns.regplot(x='GrLivArea_x_Rms', y='SalePrice', data=df);
plt.title('Living Area has beter correlation ({:1.2f}) than It Multply by R
plt.text(x=30000, y=50000, s='Correlation withou Outliers: {:1.4f}'.format(
plt.show()
del df
```

```
                                                                    Outlier
s: 1
```

In [202]:
```
train = train[train.GrLivArea * train.TotRmsAbvGrd < 45000]
print('Train observations after remove outliers:',train.shape[0])
```

```
Train observations after remove outliers: 1457
```

In [203]:
```
fig = plt.figure(figsize=(20,5))
fig1 = fig.add_subplot(131); sns.boxplot(train.GarageCars)
fig2 = fig.add_subplot(132); sns.boxplot(train.GarageArea)
fig3 = fig.add_subplot(133); sns.boxplot(train.GarageCars, train.GarageArea
plt.show()
```

In [204]:
```
df = train[['SalePrice', 'GarageArea', 'GarageCars']]
df['GarageAreaByCar'] = train.GarageArea/train.GarageCars
df['GarageArea_x_Car'] = train.GarageArea*train.GarageCars

fig = plt.figure(figsize=(20,5))
fig1 = fig.add_subplot(121); sns.regplot(x='GarageAreaByCar', y='SalePrice'
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.GarageAreaByCar.c

fig2 = fig.add_subplot(122); sns.regplot(x='GarageArea_x_Car', y='SalePrice
plt.text(x=-100, y=750000, s='Correlation with SalePrice: {:6.4f}'.format(d
print('                                                                    Out
df = df.loc[df.GarageArea_x_Car<3700]
sns.regplot(x='GarageArea_x_Car', y='SalePrice', data=df); plt.title('Garag
plt.text(x=-100, y=700000, s='Correlation withou Outliers: {:6.4f}'.format(
plt.show()
del df
```

```
                                                                    Outlier
s: 4
```

In [205]:
```
train = train[train.GarageArea * train.GarageCars < 3700]
print('Total observatiosn after outliers cut:', train.shape[0])
```

```
Total observatiosn after outliers cut: 1453
```

In [206]:
```
df = train[['SalePrice', 'TotalBsmtSF', '1stFlrSF']]
df['TotalBsmtSFByBms'] = train.TotalBsmtSF/train['1stFlrSF']
df['TotalBsmtSF_x_Bsm'] = train.TotalBsmtSF*train['1stFlrSF']
fig = plt.figure(figsize=(20,5))
fig1 = fig.add_subplot(121); sns.regplot(x='TotalBsmtSFByBms', y='SalePrice
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.TotalBsmtSFByBms.
fig2 = fig.add_subplot(122); sns.regplot(x='TotalBsmtSF_x_Bsm', y='SalePric
plt.text(x=7e06, y=90000, s='Correlation with SalePrice: {:1.4f}'.format(df

print('                                                                    Outlier
df = df.loc[df.TotalBsmtSF_x_Bsm<0.9e07]
sns.regplot(x='TotalBsmtSF_x_Bsm', y='SalePrice', data=df);
plt.title('The multiplacation is better than total basement correlation ({:
plt.text(x=7e06, y=50000, s='Correlation withou Outliers: {:1.4f}'.format(d
plt.show()
del df
```

```
Outliers: 1
```

In [207]:
```
df = train[['SalePrice', 'YearBuilt', 'GarageYrBlt']]
df['YearBuilt_x_Garage'] = train.YearBuilt*train.GarageYrBlt
df['Garage_Newest'] = train.YearBuilt < train.GarageYrBlt

fig = plt.figure(figsize=(20,5))
fig1 = fig.add_subplot(121); sns.scatterplot(y = df.SalePrice, x = df.YearB
fig2 = fig.add_subplot(122); sns.regplot(x='YearBuilt_x_Garage', y='SalePri
plt.text(x=3700000, y=600000, s='YearBuilt Correlation with SalePrice: {:1.
plt.text(x=3700000, y=550000, s='Correlation with SalePrice: {:1.4f}'.forma
plt.show()
```

In [208]:
```
sns.lmplot(y = 'SalePrice', x = 'YearBuilt', data=df, markers='.',
           aspect=1.4, height=4, hue= 'Garage_Newest', palette= 'YlOrRd')
plt.show();
del df
```

In [209]:
```
fig = plt.figure(figsize=(20,10))
fig1 = fig.add_subplot(221); sns.regplot(x='FullBath', y='SalePrice', data=
plt.title('Correlation with SalePrice: {:6.4f}'.format(train.FullBath.corr(

fig2 = fig.add_subplot(222); sns.regplot(x='HalfBath', y='SalePrice', data=
plt.title('Correlation with SalePrice: {:6.4f}'.format(train.HalfBath.corr(

fig3 = fig.add_subplot(223); sns.regplot(x='BsmtFullBath', y='SalePrice', d
plt.title('Correlation with SalePrice: {:6.4f}'.format(train.BsmtFullBath.c

fig4 = fig.add_subplot(224); sns.regplot(x='BsmtHalfBath', y='SalePrice', d
plt.title('Correlation with SalePrice: {:6.4f}'.format(train.HalfBath.corr(

plt.show()
```

In [210]:
```
df = train[['SalePrice', 'FullBath', 'HalfBath', 'BsmtFullBath', 'BsmtHalfB
df['TotBathrooms'] = df.FullBath + (df.HalfBath*0.5) + df.BsmtFullBath + (d
```

In [211]:
```
fig = plt.figure(figsize=(10,5))
sns.regplot(x='TotBathrooms', y='SalePrice', data=df); plt.legend(['Outlier
plt.text(x=1, y=680000, s='Correlation with SalePrice: {:6.4f}'.format(df.T
print('                                                                    Out
df = df.loc[df.TotBathrooms<5]
sns.regplot(x='TotBathrooms', y='SalePrice', data=df); plt.title('Cut Total
plt.text(x=1, y=630000, s='Correlation withou Outliers: {:6.4f}'.format(df.
plt.show()
```

```
                                                                    Outlier
s: 2
```

In [212]:
```
train = train[(train.FullBath + (train.HalfBath*0.5) + train.BsmtFullBath +
print('Data observations after outliers deletion:', train.shape[0])
```

```
Data observations after outliers deletion: 1451
```

In [213]:
```
#review porch features
def PorchPlots():
    fig = plt.figure(figsize=(20,10))
    fig1 = fig.add_subplot(231); sns.regplot(x='OpenPorchSF', y='SalePrice'
    plt.title('Correlation with SalePrice: {:6.4f}'.format(df.OpenPorchSF.c

    fig2 = fig.add_subplot(232); sns.regplot(x='EnclosedPorch', y='SalePric
    plt.title('Correlation with SalePrice: {:6.4f}'.format(df.EnclosedPorch

    fig3 = fig.add_subplot(233); sns.regplot(x='TSsnPorch', y='SalePrice',
    plt.title('Correlation with SalePrice: {:6.4f}'.format(df.TSsnPorch.cor

    fig4 = fig.add_subplot(234); sns.regplot(x='ScreenPorch', y='SalePrice'
    plt.title('Correlation with SalePrice: {:6.4f}'.format(df.ScreenPorch.c

    fig5 = fig.add_subplot(235); sns.regplot(x='WoodDeckSF', y='SalePrice',
    plt.title('Correlation with SalePrice: {:6.4f}'.format(df.WoodDeckSF.co

    fig6 = fig.add_subplot(236); sns.regplot(x='TotalPorchSF', y='SalePrice
    plt.title('Correlation with SalePrice: {:6.4f}'.format(df.TotalPorchSF.

    plt.show()

df = train[['SalePrice', 'OpenPorchSF', 'EnclosedPorch', 'TSsnPorch', 'Scre
df['TotalPorchSF'] = df.OpenPorchSF + df.EnclosedPorch + df.TSsnPorch + df.
#df = df[df.TotalPorchSF<=600] # A possible outlier cut!
PorchPlots()
```

In [214]:
```
df.OpenPorchSF = df.OpenPorchSF > 0
df.EnclosedPorch =  df.EnclosedPorch > 0
df.TSsnPorch = df.TSsnPorch > 0
df.ScreenPorch = df.ScreenPorch > 0
df.WoodDeckSF = df.WoodDeckSF > 0
df.TotalPorchSF = np.sqrt(df.TotalPorchSF) * (np.log1p(np.sqrt(df.TotalPorc

PorchPlots()
```

In [215]:
```
# LandSlope: Slope of property
LandSlope = {}
LandSlope['Gtl'] = 3 #'Gentle slope'
LandSlope['Mod'] = 2 #'Moderate Slope'
LandSlope['Sev'] = 1 #'Severe Slope'

df = train[['SalePrice', 'LandSlope', 'LotArea']]
df.LandSlope = df.LandSlope.map(LandSlope)
df['LotAreaMultSlope'] = (df.LotArea * df.LandSlope)

fig = plt.figure(figsize=(20,10))
fig1 = fig.add_subplot(231); sns.regplot(x='LandSlope', y='SalePrice', data
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.LandSlope.corr(df

fig2 = fig.add_subplot(232); sns.regplot(x='LotArea', y='SalePrice', data=d
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.LotArea.corr(df['

fig3 = fig.add_subplot(233); sns.regplot(x='LotAreaMultSlope', y='SalePrice
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.LotAreaMultSlope.
plt.show()
```

In [216]:
```
print(df[df.LotArea>155000])
df = df[df.LotArea<155000]
fig = plt.figure(figsize=(20,10))
fig1 = fig.add_subplot(234); sns.regplot(x='LandSlope', y='SalePrice', data
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.LandSlope.corr(df

fig2 = fig.add_subplot(235); sns.regplot(x='LotArea', y='SalePrice', data=d
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.LotArea.corr(df['

fig3 = fig.add_subplot(236); sns.regplot(x='LotAreaMultSlope', y='SalePrice
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.LotAreaMultSlope.
plt.show()
```

```
     SalePrice  LandSlope  LotArea  LotAreaMultSlope
249     277000          1   159000            159000
313     375000          1   215245            215245
335     228950          1   164660            164660
```

In [217]:
```
# Neighborhood analysis
figa = plt.figure(figsize=(20, 5))
g = train.Neighborhood.value_counts().plot(kind='bar', title='Number of Sal

figb = plt.figure(figsize=(20, 5))
plt.tight_layout()
df = train[['SalePrice', 'YrSold', 'Neighborhood']]

df['TotalArea'] = (train.TotalBsmtSF.fillna(0) + train.WoodDeckSF.fillna(0)
                   train.LotArea.fillna(0) + train.MasVnrArea.fillna(0) + t
                   train.OpenPorchSF.fillna(0) + train.TSsnPorch.fillna(0)
                   train.EnclosedPorch.fillna(0) + train.PoolArea.fillna(0)

df = df.groupby(by=['Neighborhood', 'YrSold'], as_index=False).sum()
Neig = df[['SalePrice', 'TotalArea', 'Neighborhood']].groupby(by='Neighborh
Neig['NeigPrice'] = Neig.SalePrice / Neig.TotalArea
Neig.drop(['TotalArea', 'SalePrice'], axis=1, inplace=True)
g = Neig.groupby('Neighborhood').NeigPrice.sum().sort_values(ascending = Fa
    plot(kind='bar', title='Mean Sales Prices per Area (Constructed + Lot)
Neig = Neig.groupby(by='Neighborhood', as_index=True).NeigPrice.sum().sort_
```

In [218]:
```
# Yearly Sales Price per Area (Constructed + Lot) by Neighborhood:
df['HistPriceByNeighborhood'] = df.SalePrice / df.TotalArea
df.drop(['TotalArea', 'SalePrice'], axis=1, inplace=True)

# Fill the gaps
df = df.append(pd.DataFrame([['NPkVill', 2006, df.HistPriceByNeighborhood[d
                             columns=df.columns))

df = df.append(pd.DataFrame([['Veenker', 2009, df.HistPriceByNeighborhood[d
                             columns=df.columns))

df = df.append(pd.DataFrame([['Veenker', 2010, df.HistPriceByNeighborhood[d
                             columns=df.columns))

df = df.append(pd.DataFrame([['Blueste', 2006, df.HistPriceByNeighborhood[d
                             columns=df.columns))

df = df.append(pd.DataFrame([['Blueste', 2007, df.HistPriceByNeighborhood[d
                             columns=df.columns))

df = df.append(pd.DataFrame([['Blueste', 2010, df.HistPriceByNeighborhood[d
                             columns=df.columns))

# Reserve data to merge with all data set of train and test data
YearlyPrice = df
YearlyPrice.columns = ['Neighborhood', 'YrSold', 'YearlyPriceByNeighborhood
print('                              Yearly Sales Prices per Area (Construc
g = sns.catplot(y= 'YearlyPriceByNeighborhood', x = 'YrSold', col='Neighbor
               kind="point", aspect=.6, col_wrap=7, height=4, col_order=Nei
```

```
                              Yearly Sales Prices per Area (Constructed +
Lot) by Neighborhood:
```

Neighborhood = Blueste
Neighborhood = Blmngtn
Neighborhood = BrDale
Neighborhood = NPkVill
Neighborhood = MeadowV
Neighborhood = StoneBr
Neighborhood = NridgHt
Neighborhood = Somerst
Neighborhood = NoRidge
Neighborhood = CollgCr
Neighborhood = SawyerW
Neighborhood = Crawfor
Neighborhood = Gilbert
Neighborhood = BrkSide
Neighborhood = SWISU
Neighborhood = Veenker
Neighborhood = NWAmes
Neighborhood = OldTown
Neighborhood = NAmes
Neighborhood = Mitchel
Neighborhood = Edwards
Neighborhood = Sawyer
Neighborhood = IDOTRR
Neighborhood = Timber
Neighborhood = ClearCr
YearlyPriceByNeighborhood
YrSold
35
30
25
20
15
10
5
2006 2007 2008 2009 2010

In [219]:
```
# Monthly Sales Prices per Area (Constructed + Lot) by Neighborhood:
df = train[['SalePrice', 'MoSold', 'Neighborhood']]

df['TotalArea'] = (train.TotalBsmtSF.fillna(0) + train.WoodDeckSF.fillna(0)
                   train.LotArea.fillna(0) + train.MasVnrArea.fillna(0) + t
                   train.OpenPorchSF.fillna(0) + train.TSsnPorch.fillna(0)
                   train.EnclosedPorch.fillna(0) + train.PoolArea.fillna(0)

df = df.groupby(by=['Neighborhood', 'MoSold'], as_index=False).sum()
df['HistPriceByNeighborhood'] = df.SalePrice / df.TotalArea
df.drop(['TotalArea', 'SalePrice'], axis=1, inplace=True)

print('                                    Monthly Sales Prices per Area (Cons
g = sns.catplot(y= 'HistPriceByNeighborhood', x = 'MoSold', col='Neighborhc
                kind="point", aspect=.6, col_wrap=7, height=4, col_order=Nei
```

```
                                    Monthly Sales Prices per Area (Construct
ed + Lot) by Neighborhood:
```

In [220]:
```
# Outliers from Crawfor Neighborhood
df = train[train.Neighborhood=='Crawfor'][['SalePrice', 'MoSold', 'Neighbor

df['TotalArea'] = (train.TotalBsmtSF.fillna(0) + train.WoodDeckSF.fillna(0)
                   train.LotArea.fillna(0) + train.MasVnrArea.fillna(0) + t
                   train.OpenPorchSF.fillna(0) + train.TSsnPorch.fillna(0)
                   train.EnclosedPorch.fillna(0) + train.PoolArea.fillna(0)

df['HistPriceByNeighborhood'] = df.SalePrice / df.TotalArea
df[df.HistPriceByNeighborhood>30]
```

Out[220]:

	SalePrice	MoSold	Neighborhood	TotalArea	HistPriceByNeighborhood
1181	392500	11	Crawfor	9875.000	39.747
1405	275000	1	Crawfor	8074.000	34.060

In [221]:
```
train = train.loc[~(train.SalePrice==392500.0)]
train = train.loc[~((train.SalePrice==275000.0) & (train.Neighborhood=='Cra
print('Data observations after outliers deletion:', train.shape[0])
```

```
Data observations after outliers deletion: 1449
```

In [222]:
```
# Bin neighborhood for trade cases with low observations on monthly sales p
Neigb = {}
Neigb['Blueste'] = 'Top'      # 32.212721
Neigb['Blmngtn'] = 'Top'      # 28.364756
Neigb['BrDale']  = 'BrDale'   # 24.903923
Neigb['NPkVill'] = 'NPkVill' # 23.681105
Neigb['MeadowV'] = 'High'     # 22.034923
Neigb['StoneBr'] = 'High'     # 20.475090
Neigb['NridgHt'] =  'High'    # 20.209245
Neigb['Somerst'] = 'Somerst' # 19.551888
Neigb['NoRidge'] = 'NoRidge' # 17.038145
Neigb['CollgCr'] = 'CollgCr' # 15.134767
Neigb['SawyerW'] = 'SawyerW' # 13.992995
Neigb['Crawfor'] = 'Crawfor' # 13.773418
Neigb['Gilbert'] = 'Gilbert' # 13.260281
Neigb['BrkSide'] = 'BrkSide' # 12.785202
Neigb['SWISU']   = 'SVN'      # 12.635171
Neigb['Veenker'] = 'SVN'      # 12.343735
Neigb['NWAmes']  = 'SVN'      # 12.066590
Neigb['OldTown'] = 'OldTown' # 11.571331
Neigb['NAmes']   = 'NAmes'    # 11.091393
Neigb['Mitchel'] = 'Mitchel' # 10.936368
Neigb['Edwards'] = 'Edwards' # 10.614919
Neigb['Sawyer']  = 'Sawyer'   #10.334445
Neigb['IDOTRR']  = 'IDOTRR'   # 9.880838
Neigb['Timber']  = 'Timber'   # 8.723326
Neigb['ClearCr'] = 'ClearCr' # 6.113654

# Preper dataset for Sales Price per Area (Constructed + Lot) by Neighborhc
df = train[['SalePrice', 'MoSold', 'Neighborhood']]

df['TotalArea'] = (train.TotalBsmtSF.fillna(0) + train.WoodDeckSF.fillna(0)
                   train.LotArea.fillna(0) + train.MasVnrArea.fillna(0) + t
                   train.OpenPorchSF.fillna(0) + train.TSsnPorch.fillna(0)
                   train.EnclosedPorch.fillna(0) + train.PoolArea.fillna(0)
df['Price'] = df.SalePrice/df.TotalArea

# Cut Outliers from Crawfor Neighborhood
df = df[(((df.Neighborhood == 'Crawfor') & (df.Price<30.)) | (df.Neighborhc
df.drop(['Price'], axis=1, inplace=True)

df.Neighborhood = train.Neighborhood.map(Neigb)

df = df.groupby(by=['Neighborhood', 'MoSold'], as_index=False).sum()
df['HistPriceByNeighborhood'] = df.SalePrice / df.TotalArea

# Get the index for order by value
Neig = df[['SalePrice', 'TotalArea', 'Neighborhood']].groupby(by='Neighborh
Neig['NeigPrice'] = Neig.SalePrice / Neig.TotalArea
Neig.drop(['TotalArea', 'SalePrice'], axis=1, inplace=True)
Neig = Neig.groupby(by='Neighborhood', as_index=True).NeigPrice.sum().sort_

df.drop(['TotalArea', 'SalePrice'], axis=1, inplace=True)

# Fill the gaps
df = df.append(pd.DataFrame([['Top', 1, df.HistPriceByNeighborhood[df.Neigh
```

```
                                columns=df.columns))
df = df.append(pd.DataFrame([['Top', 8, df.HistPriceByNeighborhood[df.Neigh
                                columns=df.columns))
df = df.append(pd.DataFrame([['Top', 11, df.HistPriceByNeighborhood[df.Neig
                                columns=df.columns))
df = df.append(pd.DataFrame([['Top', 12, df.HistPriceByNeighborhood[df.Neig
                                columns=df.columns))
df = df.append(pd.DataFrame([['BrDale', 1, df.HistPriceByNeighborhood[df.Ne
                                columns=df.columns))
df = df.append(pd.DataFrame([['BrDale', 10, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['BrDale', 12, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NPkVill', 1, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NPkVill', 3, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NPkVill', 5, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NPkVill', 8, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NPkVill', 9, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NPkVill', 11, df.HistPriceByNeighborhood[df.
df = df.append(pd.DataFrame([['NPkVill', 12, df.HistPriceByNeighborhood[df.
                                columns=df.columns))
df = df.append(pd.DataFrame([['NoRidge', 2, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NoRidge', 9, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['NoRidge', 10, df.HistPriceByNeighborhood[df.
                                columns=df.columns))
df = df.append(pd.DataFrame([['Crawfor', 1, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['Crawfor', 4, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['Timber', 10, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['ClearCr', 1, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['ClearCr', 2, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['ClearCr', 8, df.HistPriceByNeighborhood[df.N
                                columns=df.columns))
df = df.append(pd.DataFrame([['Edwards', 12, df.HistPriceByNeighborhood[df.
                                columns=df.columns))

# Reserve data to merge with all data set of train and test data
MonthlyPrice = df
MonthlyPrice.columns = ['Neigb', 'MoSold', 'MonthlyPriceByNeighborhood']

print('                            Monthly Hist Sales Prices per Area (Constru
g = sns.catplot(y= 'MonthlyPriceByNeighborhood', x = 'MoSold', col='Neigb',
                kind="point", aspect=.6, col_wrap=7, height=4, col_order=Nei
```

```
  File "<ipython-input-222-71e9418b7da1>", line 83
```

```
    df = df.append(pd.DataFrame([['NoRidge', 2, df.HistPriceByNeighborhoo
d[df.Neighborhood=='NoRidge'].mean()]],
     ^
SyntaxError: invalid syntax
```

In [223]:

```
def QQ_plot(data, measure):
    fig = plt.figure(figsize=(20,7))

    #Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(data)

    #Kernel Density plot
    fig1 = fig.add_subplot(121)
    sns.distplot(data, fit=norm)
    fig1.set_title(measure + ' Distribution ( mu = {:.2f} and sigma = {:.2f
    fig1.set_xlabel(measure)
    fig1.set_ylabel('Frequency')

    #QQ plot
    fig2 = fig.add_subplot(122)
    res = probplot(data, plot=fig2)
    fig2.set_title(measure + ' Probability Plot (skewness: {:.6f} and kurto

    plt.tight_layout()
    plt.show()
```

In [224]:

```
QQ_plot(train.SalePrice, 'Sales Price')
```

In [225]:
```
#We use the numpy fuction log1p which applies log(1+x) to all elements of t
train.SalePrice = np.log1p(train.SalePrice)

QQ_plot(train.SalePrice, 'Log1P of Sales Price')
```

In [226]:
```
#test hypothesis of better fit
df = train[['SalePrice', 'GrLivArea']]
df['ConstructArea'] = (train.TotalBsmtSF.fillna(0) + train.WoodDeckSF.filln
                       train.MasVnrArea.fillna(0) + train.GarageArea.fillna
                       train.TSsnPorch.fillna(0) + train.ScreenPorch.fillna
                       train.PoolArea.fillna(0) )

fig8 = plt.figure(figsize=(20,5))
fig9 = fig8.add_subplot(121); sns.regplot((df.ConstructArea), df.SalePrice)
plt.title('Cosntruct Area correlation {:1.2f}'.format(df.ConstructArea.corr

fig10 = fig8.add_subplot(122); sns.regplot((df.GrLivArea.fillna(0)), df.Sal
tit = 'Livig Area correlation is {:1.2f} and is {:1.2f} correlated to Const
plt.title(tit.format(df.GrLivArea.fillna(0).corr(df.SalePrice), df.GrLivAre
plt.show()
```

In [227]:
```
def print_results():
    # The coefficients
    print('Coefficients: \n', lr.coef_)
    # The mean squared error
    print("Root mean squared error: %.4f"
          % np.expm1(np.sqrt(mean_squared_error(y_test, y_pred))))
    # Explained variance score: 1 is perfect prediction
    print('Variance score: %.4f' % r2_score(y_test, y_pred))
    print('-----------------------------------------------------------------

scale = RobustScaler()
y = df.SalePrice

X = scale.fit_transform(df[['ConstructArea', 'GrLivArea']])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, r

lr = LinearRegression()

print('1. Linear regressor with only living Area:')
lr.fit(X_train[: , 1].reshape(-1, 1), y_train)

# Make predictions using the testing set
y_pred = lr.predict(X_test[: , 1].reshape(-1, 1))
print_results()

print('2. Linear regressor with bouth features:')
lr.fit(X_train, y_train)

# Makepredictions using the testing set
y_pred = lr.predict(X_test)
print_results()

print('3. Linear regressor with only Construct Area:')
lr = LinearRegression()
lr.fit(X_train[: , 0].reshape(-1, 1), y_train)

# Makepredictions using the testing set
y_pred = lr.predict(X_test[: , 0].reshape(-1, 1))
print_results()

print('4. Polinomial regressor of orden 3 with only Construction Area:')
# create polynomial features
cubic = PolynomialFeatures(degree=3, interaction_only=False, include_bias=F
X_cubic = cubic.fit_transform(X_train[: , 0].reshape(-1, 1))
# cubic fit
lr = lr.fit(X_cubic, y_train)
y_pred = lr.predict(cubic.fit_transform(X_test[: , 0].reshape(-1, 1)))
print_results()

print('5. Polinomial regressor of orden 3 with both features:')

# create polynomial features
cubic = PolynomialFeatures(degree=3, interaction_only=False, include_bias=F
X_cubic = cubic.fit_transform(X_train)
```

```
# cubic fit
lr = lr.fit(X_cubic, y_train)
y_pred = lr.predict(cubic.fit_transform(X_test))
print_results()
```

```
1. Linear regressor with only living Area:
Coefficients:
 [0.37373448]
Root mean squared error: 0.3135
Variance score: 0.5547
------------------------------------------------------------------------
-------

2. Linear regressor with bouth features:
Coefficients:
 [0.40446921 0.02013973]
Root mean squared error: 0.2275
Variance score: 0.7484
------------------------------------------------------------------------
-------

3. Linear regressor with only Construct Area:
Coefficients:
 [0.42050449]
Root mean squared error: 0.2276
Variance score: 0.7481
------------------------------------------------------------------------
-------

4. Polinomial regressor of orden 3 with only Construction Area:
Coefficients:
 [ 0.437434   -0.0420375   0.00630214]
Root mean squared error: 0.2212
Variance score: 0.7608
------------------------------------------------------------------------
-------

5. Polinomial regressor of orden 3 with both features:
Coefficients:
 [ 0.41182783  0.03846449 -0.05035786  0.06608028 -0.07134685 -0.01285254
  0.08643164 -0.11203676  0.04362431]
Root mean squared error: 0.2225
Variance score: 0.7584
------------------------------------------------------------------------
-------
```

In [228]:
```
#check quality
ntrain = train.shape[0]
ntest = test.shape[0]
y_train = train.SalePrice.values
all_data = pd.concat((train, test)).reset_index(drop=True)

print("All data observations have {0} rows and {1} columns".format(all_data
details = rstr(all_data)
print("All data have {1:2.2%} of null at {0} features".format(details[detai
                                                        details.nulls[details.nu
print('\nBelow the table with all columns with nulls oredered by missin rat
display(details.loc[(details.nulls>0), 'types':'uniques'].sort_values(by= '
```

```
All data observations have 2908 rows and 80 columns
Data shape: (2908, 80)

____________________________
Data types:
 object     43
int64      25
float64    12
Name: types, dtype: int64

____________________________
All data have 5.98% of null at 34 features

Below the table with all columns with nulls oredered by missin ration:
```

	types	counts	distincts	nulls	missing ration	uniques
PoolQC	object	9	4	2899	99.691	[[nan, Ex, Fa, Gd]]
MiscFeature	object	104	5	2804	96.424	[[nan, Shed, Gar2, Othr, TenC]]
Alley	object	197	3	2711	93.226	[[nan, Grvl, Pave]]
Fence	object	570	5	2338	80.399	[[nan, MnPrv, GdWo, GdPrv, MnWw]]
FireplaceQu	object	1493	6	1415	48.659	[[nan, TA, Gd, Fa, Ex, Po]]
LotFrontage	float64	2423	129	485	16.678	[[65.0, 80.0, 68.0, 60.0, 84.0, 85.0, 75.0, na...
GarageCond	object	2752	6	156	5.365	[[TA, Fa, nan, Gd, Po, Ex]]
GarageQual	object	2752	6	156	5.365	[[TA, Fa, Gd, nan, Ex, Po]]
GarageYrBlt	float64	2752	104	156	5.365	[[2003.0, 1976.0, 2001.0, 1998.0, 2000.0, 1993...
GarageFinish	object	2752	4	156	5.365	[[RFn, Unf, Fin, nan]]
GarageType	object	2754	7	154	5.296	[[Attchd, Detchd, BuiltIn, CarPort, nan, Basme...
BsmtExposure	object	2826	5	82	2.820	[[No, Gd, Mn, Av, nan]]
BsmtCond	object	2826	5	82	2.820	[[TA, Gd, nan, Fa, Po]]
BsmtQual	object	2827	5	81	2.785	[[Gd, TA, Ex, nan, Fa]]
BsmtFinType2	object	2828	7	80	2.751	[[Unf, BLQ, nan, ALQ, Rec, LwQ, GLQ]]
BsmtFinType1	object	2829	7	79	2.717	[[GLQ, ALQ, Unf, Rec, BLQ, nan, LwQ]]

	types	counts	distincts	nulls	missing ration	uniques
MasVnrType	object	2884	5	24	0.825	[[BrkFace, None, Stone, BrkCmn, nan]]
MasVnrArea	float64	2885	443	23	0.791	[[196.0, 0.0, 162.0, 350.0, 186.0, 240.0, 286....
MSZoning	object	2904	6	4	0.138	[[RL, RM, C (all), FV, RH, nan]]
Utilities	object	2906	3	2	0.069	[[AllPub, NoSeWa, nan]]
Functional	object	2906	8	2	0.069	[[Typ, Min1, Maj1, Min2, Mod, Maj2, Sev, nan]]
BsmtHalfBath	float64	2906	4	2	0.069	[[0.0, 1.0, 2.0, nan]]
BsmtFullBath	float64	2906	5	2	0.069	[[1.0, 0.0, 2.0, 3.0, nan]]
GarageCars	float64	2907	7	1	0.034	[[2.0, 3.0, 1.0, 0.0, 4.0, 5.0, nan]]
Exterior2nd	object	2907	17	1	0.034	[[VinylSd, MetalSd, Wd Shng, HdBoard, Plywood,...
Exterior1st	object	2907	16	1	0.034	[[VinylSd, MetalSd, Wd Sdng, HdBoard, BrkFace,...
KitchenQual	object	2907	5	1	0.034	[[Gd, TA, Ex, Fa, nan]]
Electrical	object	2907	6	1	0.034	[[SBrkr, FuseF, FuseA, FuseP, Mix, nan]]
BsmtUnfSF	float64	2907	1134	1	0.034	[[150.0, 284.0, 434.0, 540.0, 490.0, 64.0, 317...
BsmtFinSF2	float64	2907	273	1	0.034	[[0.0, 32.0, 668.0, 486.0, 93.0, 491.0, 506.0,...
BsmtFinSF1	float64	2907	989	1	0.034	[[706.0, 978.0, 486.0, 216.0, 655.0, 732.0, 13...
SaleType	object	2907	10	1	0.034	[[WD, New, COD, ConLD, ConLI, CWD, ConLw, Con,...
TotalBsmtSF	float64	2907	1055	1	0.034	[[856.0, 1262.0, 920.0, 756.0, 1145.0, 796.0, ...
GarageArea	float64	2907	600	1	0.034	[[548.0, 460.0, 608.0, 642.0, 836.0, 480.0, 63...

In [229]:
```
class DataFrameImputer(TransformerMixin):

    def __init__(self):
        """
        Impute missing values:
        - Columns of dtype object are imputed with the most frequent value
        - Columns of other types are imputed with mean of column.
        """
    def fit(self, X, y=None):

        self.fill = pd.Series([X[c].value_counts().index[0]
            if X[c].dtype == np.dtype('O') else X[c].mean() for c in X],
            index=X.columns)

        return self

    def transform(self, X, y=None):
        return X.fillna(self.fill)
```

In [230]:
```
all_data.drop('Utilities', axis=1, inplace=True)
```

In [231]:
```
#identify most common electrical
display(all_data.Electrical.value_counts())

all_data.Electrical = all_data.Electrical.fillna('SBrkr')
```

```
SBrkr    2661
FuseA     187
FuseF      50
FuseP       8
Mix         1
Name: Electrical, dtype: int64
```

In [232]:
```
#fill missing vals garage features
feat = ['GarageYrBlt', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageC
print(all_data[feat].isnull().sum())
print("GarageArea equal a 0: ", (all_data.GarageArea==0).sum())
print("GarageCars equal a 0: ", (all_data.GarageCars==0).sum())
c = all_data[~all_data.GarageType.isnull()][feat]
c[c.GarageYrBlt.isnull()]
```

```
GarageYrBlt     156
GarageType      154
GarageFinish    156
GarageQual      156
GarageCond      156
GarageArea        1
GarageCars        1
dtype: int64
GarageArea equal a 0:  154
GarageCars equal a 0:  154
```

Out[232]:

	GarageYrBlt	GarageType	GarageFinish	GarageQual	GarageCond	GarageArea	GarageCars
2115	nan	Detchd	NaN	NaN	NaN	360.000	1.000
2565	nan	Detchd	NaN	NaN	NaN	nan	nan

In [233]:
```
#group by garage
all_data.GarageType = all_data.GarageType.fillna('NA')

# Group by GarageType and fill missing value with median where GarageType==
cmedian = all_data[all_data.GarageType=='Detchd'].GarageArea.median()
print("GarageArea median of Type Detchd:", cmedian)
all_data.loc[all_data.GarageType=='Detchd', 'GarageArea'] = all_data.loc[al
                                                                          'G
all_data.GarageArea = all_data.GarageArea.fillna(0)

cmedian = all_data[all_data.GarageType=='Detchd'].GarageCars.median()
print("GarageCars median of Type Detchd:", cmedian)
all_data.loc[all_data.GarageType=='Detchd', 'GarageCars'] = all_data.loc[al
                                                                          'G
all_data.GarageCars = all_data.GarageCars.fillna(0)

cmedian = all_data[all_data.GarageType=='Detchd'].GarageYrBlt.median()
print("GarageYrBlt median of Type Detchd:", cmedian)
all_data.loc[all_data.GarageType=='Detchd', 'GarageYrBlt'] = all_data.loc[a
                                                                          '

all_data.GarageYrBlt = all_data.GarageYrBlt.fillna(0)

# Group by GarageType and fill missing value with mode where GarageType=='D
cmode = all_data[all_data.GarageType=='Detchd'].GarageFinish.mode()[0]
print("GarageFinish mode of Type Detchd:", cmode)
all_data.loc[all_data.GarageType=='Detchd', 'GarageFinish'] = all_data.loc[

all_data.GarageFinish = all_data.GarageFinish.fillna('NA')

cmode = all_data[all_data.GarageType=='Detchd'].GarageQual.mode()[0]
print("GarageQual mode of Type Detchd: %s" %cmode)
all_data.loc[all_data.GarageType=='Detchd', 'GarageQual'] = all_data.loc[al
                                                                          'G
all_data.GarageQual = all_data.GarageQual.fillna('NA')

cmode = all_data[all_data.GarageType=='Detchd'].GarageCond.mode()[0]
print("GarageCond mode of Type Detchd:", cmode)
all_data.loc[all_data.GarageType=='Detchd', 'GarageCond'] = all_data.loc[al
                                                                          'G
all_data.GarageCond = all_data.GarageCond.fillna('NA')
```

```
GarageArea median of Type Detchd: 397.5
GarageCars median of Type Detchd: 2.0
GarageYrBlt median of Type Detchd: 1962.0
GarageFinish mode of Type Detchd: Unf
GarageQual mode of Type Detchd: TA
GarageCond mode of Type Detchd: TA
```

In [234]:
```
# Check if all nulls of Garage features are inputed
print(all_data[feat].isnull().sum())
```

```
GarageYrBlt     0
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
GarageArea      0
GarageCars      0
dtype: int64
```

In [235]:
```
# Masonry Veneer
feat = ['MasVnrArea', 'MasVnrType']
c = all_data[~all_data.MasVnrArea.isnull()][feat]
print('Masonry veneer Nulls:')
print(all_data[feat].isnull().sum(), '\n')
print("Has MasVnrType but not has MasVnrArea:",all_data[~all_data.MasVnrTyp
print("Has MasVnrArea but not has MasVnrType:",c[c.MasVnrType.isnull()].Mas
print(c[c.MasVnrType.isnull()], '\n')

print("Has MasVnrType but MasVnrArea is equal a Zero:",c[c.MasVnrArea==0].M
print("MasVnrArea equal a 0: ", (all_data.MasVnrArea==0).sum(), '\n')
print("Has Type and Area == 0:")
print(c[c.MasVnrArea==0].MasVnrType.value_counts(), '\n')

print("Type None with Area > 0 ?")
print(all_data.loc[(all_data.MasVnrType=='None') & (all_data.MasVnrArea>0),

print('\n What is the most comumn MasVnrType after None?')
print(all_data.MasVnrType.value_counts())
```

```
Masonry veneer Nulls:
MasVnrArea    23
MasVnrType    24
dtype: int64

Has MasVnrType but not has MasVnrArea: 0
Has MasVnrArea but not has MasVnrType: 1
      MasVnrArea MasVnrType
2599     198.000        NaN

Has MasVnrType but MasVnrArea is equal a Zero: 1734
MasVnrArea equal a 0:  1734

Has Type and Area == 0:
None       1731
BrkFace       2
Stone         1
Name: MasVnrType, dtype: int64

Type None with Area > 0 ?
     MasVnrType  MasVnrArea
622        None     288.000
769        None       1.000
1222       None       1.000
1291       None     344.000
1325       None     312.000
1658       None     285.000
2441       None       1.000

 What is the most comumn MasVnrType after None?
None       1738
BrkFace     877
Stone       244
BrkCmn       25
Name: MasVnrType, dtype: int64
```

In [236]:
```
#Correct Masonry Veneer types
# All None Types with Are greater than 0 update to BrkFace type
all_data.loc[(all_data.MasVnrType=='None') & (all_data.MasVnrArea>0), ['Mas

# All Types null with Are greater than 0 update to BrkFace type
all_data.loc[(all_data.MasVnrType.isnull()) & (all_data.MasVnrArea>0), ['Ma

# All Types different from None with Are equal to 0 update to median Area c
all_data.loc[(all_data.MasVnrType!='None') &
             (all_data.MasVnrArea==0), ['MasVnrArea']] = all_data.loc[(all_
                                                                       (all_
# Filling 0 and None for records wheres both are nulls
all_data.MasVnrArea = all_data.MasVnrArea.fillna(0)
all_data.MasVnrType = all_data.MasVnrType.fillna('None')
```

In [237]:
```
#Check if all null types are updated
c = all_data[~all_data.MasVnrType.isnull()][feat]
print('Masonry veneer Nulls:')
print(all_data[feat].isnull().sum(), '\n')
```

```
Masonry veneer Nulls:
MasVnrArea    0
MasVnrType    0
dtype: int64
```

In [238]:
```
#Check & input basement feature nulls
feat = ['BsmtFinSF1','BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF','BsmtFullBath'
        'BsmtQual', 'BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2'
print(all_data[feat].isnull().sum())
print("BsmtFinSF1 equal a 0: ", (all_data.BsmtFinSF1==0).sum())
print("BsmtFinSF2 equal a 0: ", (all_data.BsmtFinSF2==0).sum())
print("BsmtUnfSF equal a 0: ", (all_data.BsmtUnfSF==0).sum())
print("TotalBsmtSF equal a 0: ", (all_data.TotalBsmtSF==0).sum())
print("BsmtFullBath equal a 0: ", (all_data.BsmtFullBath==0).sum())
print("BsmtHalfBath equal a 0: ", (all_data.BsmtHalfBath==0).sum())
```

```
BsmtFinSF1       1
BsmtFinSF2       1
BsmtUnfSF        1
TotalBsmtSF      1
BsmtFullBath     2
BsmtHalfBath     2
BsmtQual        81
BsmtCond        82
BsmtExposure    82
BsmtFinType1    79
BsmtFinType2    80
dtype: int64
BsmtFinSF1 equal a 0:  927
BsmtFinSF2 equal a 0:  2560
BsmtUnfSF equal a 0:  240
TotalBsmtSF equal a 0:  78
BsmtFullBath equal a 0:  1702
BsmtHalfBath equal a 0:  2734
```

In [239]:
```
# No Basement Av is the most comumn BsmtExposure.
display(all_data.BsmtExposure.value_counts())

# Update nulls Exposure to Av wheres TotalBsmntSF is grenter tham zero
all_data.loc[(~all_data.TotalBsmtSF.isnull()) &
             (all_data.BsmtExposure.isnull()) &
             (all_data.TotalBsmtSF>0), 'BsmtExposure'] = 'Av'
```

```
No    1899
Av     417
Gd     271
Mn     239
Name: BsmtExposure, dtype: int64
```

In [240]:
```
# TA is the most comumn BsmtQual.
display(all_data.BsmtQual.value_counts())

# We use TA for all cases wheres has same evidenci that the house has Basem
all_data.loc[(~all_data.TotalBsmtSF.isnull()) &
             (all_data.BsmtQual.isnull()) &
             (all_data.TotalBsmtSF>0), 'BsmtQual'] = 'TA'
```

```
TA    1278
Gd    1208
Ex     253
Fa      88
Name: BsmtQual, dtype: int64
```

In [241]:
```
# TA is the most comumn BsmtCond.
display(all_data.BsmtCond.value_counts())

# We use TA for all cases wheres has same evidenci that the house has Basem
all_data.loc[(~all_data.TotalBsmtSF.isnull()) & (all_data.BsmtCond.isnull()
```

```
TA    2597
Gd     121
Fa     103
Po       5
Name: BsmtCond, dtype: int64
```

In [242]:
```
# Unf is the most comumn BsmtFinType2.
display(all_data.BsmtFinType2.value_counts())

# We use Unf for all cases wheres BsmtFinType2 is null but BsmtFinSF2 is gr
all_data.loc[(all_data.BsmtFinSF2>0) & (all_data.BsmtFinType2.isnull()) , '
```

```
Unf    2482
Rec     105
LwQ      87
BLQ      68
ALQ      52
GLQ      34
Name: BsmtFinType2, dtype: int64
```

In [243]:
```
# See below that we have one case where BsmtFinType2 is BLQ and the Area is
display(all_data[(all_data.BsmtFinSF2==0) & (all_data.BsmtFinType2!='Unf')

# Correct BsmtFinSF2 and BsmtUnfSF:
all_data.loc[(all_data.BsmtFinSF2==0) & (all_data.BsmtFinType2!='Unf') & (~
all_data.loc[(all_data.BsmtFinSF2==0) & (all_data.BsmtFinType2!='Unf') & (~
```

	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	BsmtFullBath	BsmtHalfBath	BsmtQual	I
1459	1051.000	0.000	354.000	1405.000	1.000	0.000	Gd	

In [244]:
```
# All these cases are clear don´t have basement.
print("Rest cases where Cond is Null", (all_data[all_data.BsmtCond.isnull()
print('Others categories basement features are Null when Cond is Null:\n',
      (all_data[all_data.BsmtCond.isnull()][['BsmtQual', 'BsmtCond', 'BsmtE
                                             'BsmtFinType1' , 'BsmtFinType2
print('\nOthers numerics basement features are Null or Zero when Cond is Nu
      all_data[all_data.BsmtCond.isnull()][['BsmtFinSF1', 'BsmtFinSF2', 'Bs
                                            'BsmtFullBath', 'BsmtHalfBath']
print("\nThe particular cases where's numeric basement features see below a
display(all_data[all_data.BsmtFullBath.isnull()][feat])

# So, we update these Zero or NA according to their dictionary:
nulls_cols = {'BsmtExposure': 'NA', 'BsmtFinType2': 'NA', 'BsmtQual': 'NA',
              'BsmtFinSF1': 0, 'BsmtFinSF2': 0, 'BsmtUnfSF': 0 ,'TotalBsmtS

all_data = all_data.fillna(value=nulls_cols)

print('\nFinal Check if all nulls basement features are treated:', all_data
```

```
Rest cases where Cond is Null 79

Others categories basement features are Null when Cond is Null:
 BsmtQual         79
BsmtCond         79
BsmtExposure     79
BsmtFinType1     79
BsmtFinType2     79
dtype: int64

Others numerics basement features are Null or Zero when Cond is Null:
 BsmtFinSF1      0.000
BsmtFinSF2      0.000
BsmtUnfSF       0.000
TotalBsmtSF     0.000
BsmtFullBath    0.000
BsmtHalfBath    0.000
dtype: float64

The particular cases where's numeric basement features see below are Null
were included in the previous groups:
```

	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	BsmtFullBath	BsmtHalfBath	BsmtQual	I
2109	nan	nan	nan	nan	nan	nan	NaN	
2177	0.000	0.000	0.000	0.000	nan	nan	NaN	

```
Final Check if all nulls basement features are treated: 0
```

In [245]:
```
#Lot Frontage - Check and Fill Nulls
```

In [246]:
```
# Group by Neigborhood and fill missing value with Lot frontage median of t
NegMean = all_data.groupby('Neighborhood').LotFrontage.mean()

all_data.loc.LotFrontage = all_data[['Neighborhood', 'LotFrontage']].\
                            apply(lambda x: NegMean[x.Neighborhood] if np.is
```

In [247]:
```
#fill pool quality nulls
PoolQC = {0: 'NA', 1: 'Po', 2: 'Fa', 3: 'TA', 4: 'Gd', 5: 'Ex'}

all_data.loc[(all_data.PoolArea>0) & (all_data.PoolQC.isnull()), ['PoolQC']
        ((all_data.loc[(all_data.PoolArea>0) & (all_data.PoolQC.isnull()),
        apply(lambda x: x.map(PoolQC))

all_data.PoolQC = all_data.PoolQC.fillna('NA')
```

In [248]:
```
#Functional - Miss Values Treatment¶
all_data.Functional = all_data.Functional.fillna('Typ')
```

In [249]:
```
#Fireplace Quality - Miss Values Treatment
all_data.loc[(all_data.Fireplaces==0) & (all_data.FireplaceQu.isnull()), ['
```

In [250]:
```
#Kitchen Quality - Miss Values Treatment¶
all_data.loc[(all_data.KitchenAbvGr>0) & (all_data.KitchenQual.isnull()),
             ['KitchenQual']] = all_data.KitchenQual.mode()[0]
```

In [251]:
```
#Alley, Fence and Miscellaneous Feature - Miss Values Treatment¶
all_data.Alley = all_data.Alley.fillna('NA')
all_data.Fence = all_data.Fence.fillna('NA')
all_data.MiscFeature = all_data.MiscFeature.fillna('NA')
```

In [252]:
```
#Back to the Past! Garage Year Build from 2207¶
display(all_data.loc[all_data.GarageYrBlt==2207, ['GarageYrBlt', 'YearBuilt
all_data.loc[all_data.GarageYrBlt==2207.0, 'GarageYrBlt'] = 2007.0
```

	GarageYrBlt	YearBuilt
2581	2207.000	2006

In [253]:
```
#Final Check and Filling Nulls¶
all_data = DataFrameImputer().fit_transform(all_data)

# Final check if we have some NA
print("Data nulls:", all_data.isnull().sum().sum())
```

```
Data nulls: 0
```

In [254]:
```
#Mapping Ordinal Features
def map_ordinals(data):

    # LandSlope: Slope of property
    LandSlope = {}
    LandSlope['Gtl'] = 3 #'Gentle slope'
    LandSlope['Mod'] = 2 #'Moderate Slope'
    LandSlope['Sev'] = 1 #'Severe Slope'

    data.LandSlope = data.LandSlope.map(LandSlope)

    # ExterQual: Evaluates the quality of the material on the exterior
    ExterQual = {}
    ExterQual['Ex'] = 5 #'Excellent'
    ExterQual['Gd'] = 4 #'Good'
    ExterQual['TA'] = 3 #'Average/Typical'
    ExterQual['Fa'] = 2 #'Fair'
    ExterQual['Po'] = 1 #'Poor'
    ExterQual['NA'] = 0 #'NA'

    data.ExterQual = data.ExterQual.map(ExterQual)

    # ExterCond: Evaluates the present condition of the material on the ext
    data.ExterCond = data.ExterCond.map(ExterQual)
    #HeatingQC: Heating quality and condition
    data.HeatingQC = data.HeatingQC.map(ExterQual)

    # KitchenQual: Kitchen quality
    data.KitchenQual = data.KitchenQual.map(ExterQual)

    # FireplaceQu: Fireplace quality
    data.FireplaceQu = data.FireplaceQu.map(ExterQual)

    # GarageCond: Garage Conditionals
    data.GarageCond = data.GarageCond.map(ExterQual)

    PavedDrive = {}
    PavedDrive['Y'] = 3 #'Paved'
    PavedDrive['P'] = 2 #'Partial Pavement'
    PavedDrive['N'] = 1 #'Dirt/Gravel'

    data.PavedDrive = data.PavedDrive.map(PavedDrive)

    # LotShape: General shape of property
    LotShape = {}
    LotShape['Reg'] = 4 #'Regular'
    LotShape['IR1'] = 3 #'Slightly irregular'
    LotShape['IR2'] = 2 #'Moderately Irregular'
    LotShape['IR3'] = 1 #'Irregular'

    data.LotShape = data.LotShape.map(LotShape)

    # BsmtQual: Evaluates the height of the basement
    BsmtQual = {}
    BsmtQual['Ex'] = 5 #'Excellent (100+ inches)'
    BsmtQual['Gd'] = 4 #'Good (90-99 inches)'
```

```
    BsmtQual['TA'] = 3 #'Typical (80-89 inches)'
    BsmtQual['Fa'] = 2 #'Fair (70-79 inches)'
    BsmtQual['Po'] = 1 #'Poor (<70 inches'
    BsmtQual['NA'] = 0 #'No Basement'

    data.BsmtQual = data.BsmtQual.map(BsmtQual)

    # BsmtCond: Evaluates the general condition of the basement
    data.BsmtCond = data.BsmtCond.map(BsmtQual)

    # GarageQual: Garage quality
    data.GarageQual = data.GarageQual.map(BsmtQual)

    # PoolQC: Pool quality
    data.PoolQC = data.PoolQC.map(BsmtQual)

    # BsmtExposure: Refers to walkout or garden level walls
    BsmtExposure = {}
    BsmtExposure['Gd'] = 4 #'Good Exposure'
    BsmtExposure['Av'] = 3 #'Average Exposure (split levels or foyers typic
    BsmtExposure['Mn'] = 2 #'Mimimum Exposure'
    BsmtExposure['No'] = 1 #'No Exposure'
    BsmtExposure['NA'] = 0 #'No Basement'

    data.BsmtExposure = data.BsmtExposure.map(BsmtExposure)

    # BsmtFinType1: Rating of basement finished area
    BsmtFinType1 = {}
    BsmtFinType1['GLQ'] = 6 #'Good Living Quarters'
    BsmtFinType1['ALQ'] = 5 # 'Average Living Quarters'
    BsmtFinType1['BLQ'] = 4 # 'Below Average Living Quarters'
    BsmtFinType1['Rec'] = 3 # 'Average Rec Room'
    BsmtFinType1['LwQ'] = 2 # 'Low Quality'
    BsmtFinType1['Unf'] = 1 # 'Unfinshed'
    BsmtFinType1['NA'] = 0 #'No Basement'

    data.BsmtFinType1 = data.BsmtFinType1.map(BsmtFinType1)

    # BsmtFinType2: Rating of basement finished area (if multiple types)
    data.BsmtFinType2 = data.BsmtFinType2.map(BsmtFinType1)

    #CentralAir: Central air conditioning
    # Since with this transformatio as the same as binarize this feature
    CentralAir = {}
    CentralAir['N'] = 0
    CentralAir['Y'] = 1

    data.CentralAir = data.CentralAir.map(CentralAir)
# GarageFinish: Interior finish of the garage
    GarageFinish = {}
    GarageFinish['Fin'] = 3 #'Finished'
    GarageFinish['RFn'] = 2 #'Rough Finished'
    GarageFinish['Unf'] = 1 #'Unfinished'
    GarageFinish['NA'] = 0 #'No Garage'

    data.GarageFinish = data.GarageFinish.map(GarageFinish)
```

```
    # Functional: Home functionality
    Functional = {}
    Functional['Typ'] = 7   # Typical Functionality
    Functional['Min1'] = 6  # Minor Deductions 1
    Functional['Min2'] = 5  # Minor Deductions 2
    Functional['Mod'] = 4   # Moderate Deductions
    Functional['Maj1'] = 3  # Major Deductions 1
    Functional['Maj2'] = 2  # Major Deductions 2
    Functional['Sev'] = 1   # Severely Damaged
    Functional['Sal'] = 0   # Salvage only

    data.Functional = data.Functional.map(Functional)

    #Street: Type of road access to property
    # Since with this transformatio as the same as binarize this feature
    Street = {}
    Street['Grvl'] = 0 # Gravel
    Street['Pave'] = 1 # Paved

    data.Street = data.Street.map(Street)


    # Fence: Fence quality
    Fence = {}
    Fence['GdPrv'] = 5 #'Good Privacy'
    Fence['MnPrv'] = 4 #'Minimum Privacy'
    Fence['GdWo'] = 3 #'Good Wood'
    Fence['MnWw'] = 2 #'Minimum Wood/Wire'
    Fence['NA'] = 1 #'No Fence'

    data.Fence = data.Fence.map(Fence)
    #But No Fence has the higest median Sales Price. So I try to use it as

    return data

all_data = map_ordinals(all_data)
```

In [255]:

```
#Include pool in the Miscellaneous features
display(all_data.loc[(all_data.PoolArea>0) & (all_data.MiscVal>0), ['MiscFe
```

	MiscFeature	MiscVal	PoolArea	PoolQC
1376	TenC	2000	519	2

In [256]:

```
#Check if we had others "TenC" in the dataset
display(all_data.loc[(all_data.MiscFeature=='TenC'), ['MiscFeature', 'MiscV
```

	MiscFeature	MiscVal	PoolArea	PoolQC
1376	TenC	2000	519	2

In [257]:
```
#Since we don't have other "TenC" and others Pools don't coincide with any
all_data.loc[(all_data.PoolArea>0), ['MiscFeature']] = 'Pool'
all_data.loc[(all_data.PoolArea>0), ['MiscVal']] = all_data.loc[(all_data.P
                                                                ['MiscVal',
                                                                 apply(lambd
```

In [258]:
```
#points review
all_data['TotalExtraPoints'] = all_data.HeatingQC + all_data.PoolQC + all_d
all_data['TotalPoints'] =  (all_data.ExterQual + all_data.FireplaceQu + all
                            all_data.BsmtQual + all_data.BsmtExposure + all
                            all_data.ExterCond + all_data.BsmtCond + all_da
                            all_data.BsmtFinType2 + all_data.HeatingQC ) +

df = all_data.loc[(all_data.SalePrice>0), ['TotalPoints', 'TotalExtraPoints
               'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
               'FireplaceQu', 'GarageQual', 'GarageCond', 'SalePrice']]
```

In [259]:
```
# plot
fig = plt.figure(figsize=(20,10))
fig1 = fig.add_subplot(231); sns.regplot(x='BsmtExposure', y='SalePrice', d
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.BsmtExposure.corr

fig2 = fig.add_subplot(232); sns.regplot(x='BsmtFinType1', y='SalePrice', d
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.BsmtFinType1.corr

fig3 = fig.add_subplot(233); sns.regplot(x='BsmtFinType2', y='SalePrice', d
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.BsmtFinType2.corr

fig4 = fig.add_subplot(234); sns.regplot(x='GarageQual', y='SalePrice', dat
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.GarageQual.corr(d

fig5 = fig.add_subplot(235); sns.regplot(x='GarageCond', y='SalePrice', dat
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.GarageCond.corr(d

fig6 = fig.add_subplot(236); sns.regplot(x='TotalPoints', y='SalePrice', da
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.TotalPoints.corr(
plt.show()
```

In [260]:
```
fig = plt.figure(figsize=(20,10))
fig1 = fig.add_subplot(231); sns.regplot(x='OverallQual', y='SalePrice', da
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.OverallQual.corr(

fig2 = fig.add_subplot(232); sns.regplot(x='OverallCond', y='SalePrice', da
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.OverallCond.corr(

fig3 = fig.add_subplot(233); sns.regplot(x='ExterQual', y='SalePrice', data
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.ExterQual.corr(df

fig4 = fig.add_subplot(234); sns.regplot(x='ExterCond', y='SalePrice', data
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.ExterCond.corr(df

fig5 = fig.add_subplot(235); sns.regplot(x='BsmtQual', y='SalePrice', data=
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.BsmtQual.corr(df[

fig6 = fig.add_subplot(236); sns.regplot(x='BsmtCond', y='SalePrice', data=
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.BsmtCond.corr(df[
plt.show()
```

In [261]:
```
fig = plt.figure(figsize=(20,10))
fig1 = fig.add_subplot(231); sns.regplot(x='HeatingQC', y='SalePrice', data
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.HeatingQC.corr(df

fig2 = fig.add_subplot(232); sns.regplot(x='PoolQC', y='SalePrice', data=df
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.PoolQC.corr(df['S

fig3 = fig.add_subplot(233); sns.regplot(x='KitchenQual', y='SalePrice', da
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.KitchenQual.corr(

fig4 = fig.add_subplot(234); sns.regplot(x='FireplaceQu', y='SalePrice', da
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.FireplaceQu.corr(

fig5 = fig.add_subplot(235); sns.regplot(x='TotalExtraPoints', y='SalePrice
plt.title('Correlation with SalePrice: {:6.4f}'.format(df.TotalExtraPoints.

plt.show()
```

In [262]:
```
all_data['GarageArea_x_Car'] = all_data.GarageArea * all_data.GarageCars

all_data['TotalBsmtSF_x_Bsm'] = all_data.TotalBsmtSF * all_data['1stFlrSF']

# We don´t have a feature with all construct area, maybe it is an interesti
all_data['ConstructArea'] = (all_data.TotalBsmtSF + all_data.WoodDeckSF + a
                             all_data.OpenPorchSF + all_data.TSsnPorch + al
                             all_data.MasVnrArea + all_data.GarageArea + al

#all_data['TotalArea'] = all_data.ConstructArea + all_data.LotArea

all_data['Garage_Newest'] = all_data.YearBuilt > all_data.GarageYrBlt
all_data.Garage_Newest =  all_data.Garage_Newest.apply(lambda x: 1 if x els

all_data['TotalPorchSF'] = all_data.OpenPorchSF + all_data.EnclosedPorch +
all_data.EnclosedPorch = all_data.EnclosedPorch.apply(lambda x: 1 if x else

all_data['LotAreaMultSlope'] = all_data.LotArea * all_data.LandSlope


all_data['BsmtSFPoints'] = (all_data.BsmtQual**2 + all_data.BsmtCond + all_
                            all_data.BsmtFinType1 + all_data.BsmtFinType2)


all_data['BsmtSFMultPoints'] = all_data.TotalBsmtSF * (all_data.BsmtQual**2
                                                       all_data.BsmtFinType
all_data['TotBathrooms'] = all_data.FullBath + (all_data.HalfBath*0.5) + al
all_data.FullBath = all_data.FullBath.apply(lambda x: 1 if x else 0)
all_data.HalfBath = all_data.HalfBath.apply(lambda x: 1 if x else 0)
all_data.BsmtFullBath = all_data.BsmtFullBath.apply(lambda x: 1 if x else 0
all_data.BsmtHalfBath = all_data.BsmtHalfBath.apply(lambda x: 1 if x else 0
```

In [263]:
```
#One Hot Encode Categorical Features
def one_hot_encode(df):
    categorical_cols = df.select_dtypes(include=['object']).columns

    print(len(categorical_cols), "categorical columns")
    print(categorical_cols)
    # Remove special charactres and withe spaces.
    for col in categorical_cols:
        df[col] = df[col].str.replace('\W', '').str.replace(' ', '_') #.str

    dummies = pd.get_dummies(df[categorical_cols], columns = categorical_cc
    df = pd.get_dummies(df, columns = categorical_cols)

    print("Total Columns:",len(df.columns))
    print(df.info())

    return df, dummies

# Correct Categorical from int to str types
all_data.MSSubClass = all_data.MSSubClass.astype('str')
all_data.MoSold = all_data.MoSold.astype('str')

all_data, dummies = one_hot_encode(all_data)
```

```
23 categorical columns
Index(['MSSubClass', 'MSZoning', 'Alley', 'LandContour', 'LotConfig',
       'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyl
e',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrTyp
e',
       'Foundation', 'Heating', 'Electrical', 'GarageType', 'MiscFeatur
e',
       'MoSold', 'SaleType', 'SaleCondition'],
      dtype='object')
Total Columns: 259
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2908 entries, 0 to 2907
Columns: 259 entries, LotFrontage to SaleCondition_Partial
dtypes: float64(15), int64(52), uint8(192)
memory usage: 2.0 MB
None
```

In [264]:
```
#Removing Dummies with none observations in train or test datasets
# Find Dummies with all test observatiosn are equal to 0
ZeroTest = all_data[dummies][ntrain:].sum()==0
all_data.drop(dummies[ZeroTest], axis=1, inplace=True)
print('Dummies in test dataset with all observatios equal to 0:',len(dummie
dummies = dummies.drop(dummies[ZeroTest])

# Find dummies with all training observatiosn are equal to 0
ZeroTest = all_data[dummies][:ntrain].sum()==0
all_data.drop(dummies[ZeroTest], axis=1, inplace=True)
print('Dummies in trainig dataset with all observatios equal to 0:',len(dum
dummies = dummies.drop(dummies[ZeroTest])

del ZeroTest
```

```
Dummies in test dataset with all observatios equal to 0: 13 of
 Index(['Condition2_RRAe', 'Condition2_RRAn', 'Condition2_RRNn',
       'HouseStyle_25Fin', 'RoofMatl_Membran', 'RoofMatl_Metal',
       'RoofMatl_Roll', 'Exterior1st_ImStucc', 'Exterior1st_Stone',
       'Exterior2nd_Other', 'Heating_Floor', 'Heating_OthW', 'Electrical_
Mix'],
      dtype='object')

Dummies in trainig dataset with all observatios equal to 0: 1 of
 Index(['MSSubClass_150'], dtype='object')
```

In [265]:
```
#Transform Years to Ages and Create Flags to New and Remod
def AgeYears(feature):
    return feature.apply(lambda x: 0 if x==0 else (2011 - x))

all_data.YearBuilt = AgeYears(all_data.YearBuilt)
all_data.YearRemodAdd = AgeYears(all_data.YearRemodAdd)
all_data.GarageYrBlt = AgeYears(all_data.GarageYrBlt)
all_data.YrSold =  AgeYears(all_data.YrSold)
```

In [266]:
```
all_data['Remod'] = 2
all_data.loc[(all_data.YearBuilt==all_data.YearRemodAdd), ['Remod']] = 0
all_data.loc[(all_data.YearBuilt!=all_data.YearRemodAdd), ['Remod']] = 1

all_data.Age = all_data.YearRemodAdd - all_data.YrSold # sice I convert bot

all_data["IsNew"] = 2
all_data.loc[(all_data.YearBuilt==all_data.YrSold), ['IsNew']] = 1
all_data.loc[(all_data.YearBuilt!=all_data.YrSold), ['IsNew']] = 0
```

In [267]:
```
#Check for any correlations between features
```

In [268]:
```
corr = all_data[all_data.SalePrice>1].corr()
top_corr_cols = corr[abs((corr.SalePrice)>=.26)].SalePrice.sort_values(asce
top_corr = corr.loc[top_corr_cols, top_corr_cols]
dropSelf = np.zeros_like(top_corr)
dropSelf[np.triu_indices_from(dropSelf)] = True
plt.figure(figsize=(20, 20))
sns.heatmap(top_corr, cmap=sns.diverging_palette(220, 10, as_cmap=True), an
sns.set(font_scale=0.5)
plt.show()
del corr, dropSelf, top_corr
```

In [269]:
```
#Drop the features with highest correlations to other Features
all_data.drop(['FireplaceQu', 'BsmtSFPoints', 'TotalBsmtSF', 'GarageArea',
               'TotalBsmtSF_x_Bsm', '1stFlrSF', 'PoolArea', 'LotArea', 'Sal
               'GarageCond', 'HouseStyle_2Story', 'BsmtSFMultPoints', 'Scre
               'TSsnPorch'], axis=1, inplace=True)
```

In [270]:
```
corr = all_data[all_data.SalePrice>1].corr()
top_corr_cols = corr[abs((corr.SalePrice)>=.26)].SalePrice.sort_values(asce
top_corr = corr.loc[top_corr_cols, top_corr_cols]
dropSelf = np.zeros_like(top_corr)
dropSelf[np.triu_indices_from(dropSelf)] = True
plt.figure(figsize=(20, 20))
sns.heatmap(top_corr, cmap=sns.diverging_palette(220, 10, as_cmap=True), an
sns.set(font_scale=0.5)
plt.show()
del corr, dropSelf, top_corr
```

In [271]:
```
sns.set(font_scale=1.0)
g = sns.pairplot(all_data.loc[all_data.SalePrice>0, top_corr_cols[:12]])
```

In [272]:
```
#Identify and treat multicollinearity
```

In [273]:
```
all_data.rename(columns={'2ndFlrSF':'SndFlrSF'}, inplace=True)

def VRF(predict, data, y):

    scale = StandardScaler(with_std=False)
    df = pd.DataFrame(scale.fit_transform(data), columns= cols)
    features = "+".join(cols)
    df['SalePrice'] = y.values

    # get y and X dataframes based on this regression:
    y, X = dmatrices(predict + ' ~' + features, data = df, return_type='dat

   # Calculate VIF Factors
    # For each X, calculate VIF and save in dataframe
    vif = pd.DataFrame()
    vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in ra
    vif["features"] = X.columns

    # Inspect VIF Factors
    display(vif.sort_values('VIF Factor'))
    return vif

# Remove the higest correlations and run a multiple regression
cols = all_data.columns
cols = cols.drop(['SalePrice'])
vif = VRF('SalePrice', all_data.loc[all_data.SalePrice>0, cols], all_data.S
```

	VIF Factor	features
0	1.000	Intercept
41	1.267	YrSold
39	1.462	Fence
22	1.529	BsmtHalfBath
10	1.530	ExterCond
...	...	...
102	inf	Neighborhood_SawyerW
103	inf	Neighborhood_Somerst
104	inf	Neighborhood_StoneBr
97	inf	Neighborhood_NoRidge
113	inf	Condition1_RRAn

227 rows × 2 columns

In [274]:
```
# Remove one feature with VIF on Inf from the same category and run a multi
cols = cols.drop(['Condition1_PosN', 'Neighborhood_NWAmes', 'Exterior1st_CB
                  'MSZoning_Call', 'Alley_Grvl', 'LandContour_Bnk', 'LotCon
                  'MasVnrType_BrkCmn', 'Foundation_CBlock', 'MiscFeature_Ga

vif = VRF('SalePrice', all_data.loc[all_data.SalePrice>0, cols], all_data.S
```

	VIF Factor	features
0	1.000	Intercept
165	1.191	Foundation_Wood
197	1.196	SaleType_Con
202	1.216	SaleType_Oth
75	1.222	LotConfig_FR3
...	...	...
193	inf	MoSold_7
192	inf	MoSold_6
191	inf	MoSold_5
189	inf	MoSold_3
190	inf	MoSold_4

211 rows × 2 columns

In [275]:
```
# Remove one feature with highest VIF from the same category and run a mult
cols = cols.drop(['PoolQC', 'BldgType_TwnhsE', 'BsmtFinSF1', 'BsmtUnfSF', '
                  'Exterior2nd_VinylSd', 'GarageQual', 'GarageType_Attchd',
                  'MiscFeature_NA', 'MSZoning_RL', 'RoofStyle_Gable', 'Sale
                  'SaleType_New', 'SndFlrSF', 'TotalPorchSF', 'WoodDeckSF',

vif = VRF('SalePrice', all_data.loc[all_data.SalePrice>0, cols], all_data.S
```

	VIF Factor	features
0	1.000	Intercept
186	1.134	SaleCondition_Family
177	1.151	SaleType_Con
181	1.162	SaleType_Oth
179	1.167	SaleType_ConLI
...	...	...
134	25.316	Exterior2nd_CmentBd
50	25.578	MSSubClass_50
47	31.834	MSSubClass_20
105	33.733	BldgType_2fmCon
46	35.043	MSSubClass_190

189 rows × 2 columns

In [276]:
```
#Defining Categorical and Boolean Data as unit8 types
# Reserve a copy for futer analysis:
df_copy = all_data[all_data.SalePrice>0].copy()

all_data.CentralAir = all_data.CentralAir.astype('uint8')
all_data.Garage_Newest = all_data.Garage_Newest.astype('uint8')
all_data.EnclosedPorch = all_data.EnclosedPorch.astype('uint8')
all_data.FullBath = all_data.FullBath.astype('uint8')
all_data.HalfBath = all_data.HalfBath.astype('uint8')
all_data.BsmtFullBath = all_data.BsmtFullBath.astype('uint8')
all_data.BsmtHalfBath = all_data.BsmtHalfBath.astype('uint8')
all_data.Remod = all_data.Remod.astype('uint8')
all_data.IsNew = all_data.IsNew.astype('uint8')
all_data.Street = all_data.Street.astype('uint8') # ordinal
all_data.PavedDrive = all_data.PavedDrive.astype('uint8') # ordinal
all_data.Functional = all_data.Functional.astype('uint8') # ordinal
all_data.LandSlope = all_data.LandSlope.astype('uint8') # ordinal

'''
for feat in cols:
    if all_data[feat].dtype=='uint8':
        all_data[feat] = all_data[feat].astype('category')'''
```

Out[276]:
```
"\nfor feat in cols:\n    if all_data[feat].dtype=='uint8':\n        all_
data[feat] = all_data[feat].astype('category')"
```

In [277]:
```
#Box cox transformation of highly skewed features
numeric_features = list(all_data.loc[:, cols].dtypes[(all_data.dtypes != "c

'''
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=RuntimeWarning)
'''
skewed_features = all_data[numeric_features].apply(lambda x : skew (x.dropn

#compute skewness
skewness = pd.DataFrame({'Skew' :skewed_features})

# Get only higest skewed features
skewness = skewness[abs(skewness) > 0.7]
skewness = skewness.dropna()
print ("There are {} higest skewed numerical features to box cox transform"

l_opt = {}

for feat in skewness.index:
    all_data[feat], l_opt[feat] = boxcox((all_data[feat]+1))

skewed_features2 = all_data[skewness.index].apply(lambda x : skew (x.dropna

#compute skewness
skewness2 = pd.DataFrame({'New Skew' :skewed_features2})
display(pd.concat([skewness, skewness2], axis=1).sort_values(by=['Skew'], a
```

```
There are 18 higest skewed numerical features to box cox transform
```

	Skew	New Skew
MiscVal	21.750	4.772
KitchenAbvGr	4.349	-2.059
LotAreaMultSlope	3.494	0.136
BsmtFinType2	3.147	-0.434
MasVnrArea	2.622	0.457
OpenPorchSF	2.529	-0.025
Fence	1.751	1.533
GarageArea_x_Car	1.354	-0.132
ExterCond	1.313	-0.036
LotFrontage	1.231	0.171
BsmtExposure	1.120	-0.030
ConstructArea	1.088	0.022
ExterQual	0.788	-0.023
Fireplaces	0.725	0.112
TotRmsAbvGrd	0.717	-0.000

	Skew	New Skew
LotShape	-1.247	-0.597
BsmtQual	-1.265	0.046
BsmtCond	-3.633	0.657

In [278]:
```
def QQ_plot(data, measure):
    fig = plt.figure(figsize=(12,4))

    #Get the fitted parameters used by the function
    (mu, sigma) = norm.fit(data)

    #Kernel Density plot
    fig1 = fig.add_subplot(121)
    sns.distplot(data, fit=norm)
    fig1.set_title(measure + ' Distribution ( mu = {:.2f} and sigma = {:.2f
    fig1.set_xlabel(measure)
    fig1.set_ylabel('Frequency')

    #QQ plot
    fig2 = fig.add_subplot(122)
    res = probplot(data, plot=fig2)
    fig2.set_title(measure + ' Probability Plot (skewness: {:.6f} and kurto
                   format(data.skew(), data.kurt()), loc='center')

    plt.tight_layout()
    plt.show()

for feat in skewness.index:
    QQ_plot(all_data[feat], ('Boxcox1p of {}'.format(feat)))
```

In [279]:
```
#Evaluate Apply Polynomials by Region Plots on the more Correlated Features
fig = plt.figure(figsize=(20,15))
ax = fig.add_subplot(331); g = sns.regplot(y='SalePrice', x='ConstructArea'
ax = fig.add_subplot(332); g = sns.regplot(y='SalePrice', x='GarageArea_x_C
ax = fig.add_subplot(333); g = sns.regplot(y='SalePrice', x='LotAreaMultSlo
ax = fig.add_subplot(334); g = sns.regplot(y='SalePrice', x='TotalExtraPoin
ax = fig.add_subplot(335); g = sns.regplot(y='SalePrice', x='TotalPoints',

plt.tight_layout()
```

In [280]:
```
#Evaluating Polynomials Options Performance
def poly(X, y, feat=''):

    # Initializatin of regression models
    regr = LinearRegression()
    regr = regr.fit(X, y)
    y_lin_fit = regr.predict(X)
    linear_r2 = r2_score(y, regr.predict(X))

    # create polynomial features
    quadratic = PolynomialFeatures(degree=2, interaction_only=False, includ
    cubic = PolynomialFeatures(degree=3, interaction_only=False, include_bi
    fourth = PolynomialFeatures(degree=4, interaction_only=False, include_b
    fifth = PolynomialFeatures(degree=5, interaction_only=False, include_bi
    X_quad = quadratic.fit_transform(X)
    X_cubic = cubic.fit_transform(X)
    X_fourth = fourth.fit_transform(X)
    X_fifth = fifth.fit_transform(X)

    # quadratic fit
    regr = regr.fit(X_quad, y)
    y_quad_fit = regr.predict(quadratic.fit_transform(X))
    quadratic_r2 = r2_score(y, y_quad_fit)
    # cubic fit
    regr = regr.fit(X_cubic, y)
    y_cubic_fit = regr.predict(cubic.fit_transform(X))
    cubic_r2 = r2_score(y, y_cubic_fit)

    # Fourth fit
    regr = regr.fit(X_fourth, y)
    y_fourth_fit = regr.predict(fourth.fit_transform(X))
    four_r2 = r2_score(y, y_fourth_fit)
    # Fifth fit
    regr = regr.fit(X_fifth, y)
    y_fifth_fit = regr.predict(fifth.fit_transform(X))
    five_r2 = r2_score(y, y_fifth_fit)

    if len(feat)>0:
        fig = plt.figure(figsize=(20,5))
        # Plot lowest Polynomials
        fig1 = fig.add_subplot(121)
        plt.scatter(X[feat], y, label='training points', color='lightgray')
        plt.plot(X[feat], y_lin_fit, label='linear (d=1), $R^2=%.3f$' % lin
        plt.plot(X[feat], y_quad_fit, label='quadratic (d=2), $R^2=%.3f$' %
        plt.plot(X[feat], y_cubic_fit, label='cubic (d=3), $R^2=%.3f$' % cu

        plt.xlabel(feat)
        plt.ylabel('Sale Price')
        plt.legend(loc='upper left')
        # Plot higest Polynomials
        fig2 = fig.add_subplot(122)
        plt.scatter(X[feat], y, label='training points', color='lightgray')
        plt.plot(X[feat], y_lin_fit, label='linear (d=1), $R^2=%.3f$' % lin
        plt.plot(X[feat], y_fifth_fit, label='Fifth (d=5), $R^2=%.3f$' % fi
        plt.plot(X[feat], y_fifth_fit, label='Fourth (d=4), $R^2=%.3f$' % f
```

```
        plt.xlabel(feat)
        plt.ylabel('Sale Price')
        plt.legend(loc='upper left')
    else:
        # Plot initialisation
        fig = plt.figure(figsize=(20,10))
        ax = fig.add_subplot(121, projection='3d')
        ax.scatter(X.iloc[:, 0], X.iloc[:, 1], y, s=40)

        # make lines of the regressors:
        plt.plot(X.iloc[:, 0], X.iloc[:, 1], y_lin_fit, label='linear (d=1)
                 color='blue', lw=2, linestyle=':')
        plt.plot(X.iloc[:, 0], X.iloc[:, 1], y_quad_fit, label='quadratic (
                 color='red', lw=0.5, linestyle='-')
        plt.plot(X.iloc[:, 0], X.iloc[:, 1], y_cubic_fit, label='cubic (d=3
                 color='green', lw=0.5, linestyle='--')
         # label the axes
        ax.set_xlabel(X.columns[0])
        ax.set_ylabel(X.columns[1])
        ax.set_zlabel('Sales Price')
        ax.set_title("Poly up to 3 degree")
        plt.legend(loc='upper left')

    plt.tight_layout()
    plt.show()
```

In [281]:

```
y = all_data.SalePrice[all_data.SalePrice>0]
X = all_data.loc[all_data.SalePrice>0, ['ConstructArea']]
poly(X, y, 'ConstructArea')

X = all_data.loc[all_data.SalePrice>0, ['ConstructArea', 'TotalPoints']]
poly(X, y)

X = all_data.loc[all_data.SalePrice>0, ['ConstructArea', 'TotalPoints', 'Lc
poly(X, y)
```

In [282]:
```
poly_cols = ['ConstructArea', 'TotalPoints', 'LotAreaMultSlope',  'GarageAr

pf = PolynomialFeatures(degree=3, interaction_only=False, include_bias=Fals
res = pf.fit_transform(all_data.loc[:, poly_cols])

target_feature_names = [feat.replace(' ','_') for feat in pf.get_feature_na
output_df = pd.DataFrame(res, columns = target_feature_names,  index=all_da
print('Polynomial Features included:', output_df.shape[1])
display(output_df.head())
all_data = pd.concat([all_data, output_df], axis=1)
print('Total Features after Polynomial Features included:', all_data.shape[
colsP = output_df.columns

del output_df, target_feature_names, res, pf
```

```
Polynomial Features included: 30
```

	ConstructArea^2	ConstructArea_TotalPoints	ConstructArea_LotAreaMultSlope	ConstructArea_Gara
0	644.889	2310.914	1862.103	
1	635.244	2117.141	1926.778	
2	660.557	2441.624	2069.062	
3	650.361	2346.200	1946.249	
4	786.236	3112.428	2438.275	

5 rows × 30 columns

```
Total Features after Polynomial Features included: 257
```

In [283]:
```
#Separate Train, Test Datasets, identifiers and Dependent Variable
y_train = (all_data.SalePrice[all_data.SalePrice>0].reset_index(drop=True,

# Data with Polynomials
train = all_data.loc[(all_data.SalePrice>0), cols].reset_index(drop=True, i
test = all_data.loc[(all_data.SalePrice==0), cols].reset_index(drop=True, i
```

In [284]:
```
#Select Features
scale = RobustScaler()
# Data without Polynomials
df = pd.DataFrame(scale.fit_transform(train[cols]), columns= cols)
```

In [285]:
```
#Wrapper Methods
ln_model=sm.OLS(y_train,df)
result=ln_model.fit()
print(result.summary2())
```

```
                        Results: Ordinary least squares
=========================================================================
======
Model:                    OLS              Adj. R-squared (uncentered): 0.9
94
Dependent Variable:       SalePrice        AIC:                         396
1.2009
Date:                     2022-01-16 20:45 BIC:                         495
3.5832
No. Observations:         1449             Log-Likelihood:              -17
92.6
Df Model:                 188              F-statistic:                 139
0.
Df Residuals:             1261             Prob (F-statistic):          0.0
0
R-squared (uncentered):   0.995            Scale:                       0.7
9880
-------------------------------------------------------------------------
-------
```

In [286]:
```
pv_cols = cols.values

def backwardElimination(x, Y, sl, columns):
    ini = len(columns)
    numVars = x.shape[1]
    for i in range(0, numVars):
        regressor = sm.OLS(Y, x).fit()
        maxVar = max(regressor.pvalues) #.astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor.pvalues[j].astype(float) == maxVar):
                    columns = np.delete(columns, j)
                    x = x.loc[:, columns]

    print('\nSelect {:d} features from {:d} by best p-values.'.format(len(c
    print('The max p-value from the features selecte is {:.3f}.'.format(max
    print(regressor.summary())

    # odds ratios and 95% CI
    conf = np.exp(regressor.conf_int())
    conf['Odds Ratios'] = np.exp(regressor.params)
    conf.columns = ['2.5%', '97.5%', 'Odds Ratios']
    display(conf)

    return columns, regressor

SL = 0.051

pv_cols, LR = backwardElimination(df, y_train, SL, pv_cols)
```

```
Select 107 features from 188 by best p-values.
The max p-value from the features selecte is 0.050.
                                 OLS Regression Results
=======================================================================
==============
Dep. Variable:              SalePrice   R-squared (uncentered):
0.995
Model:                            OLS   Adj. R-squared (uncentered):
0.995
Method:                 Least Squares   F-statistic:
2454.
Date:                Sun, 16 Jan 2022   Prob (F-statistic):
0.00
Time:                        20:45:21   Log-Likelihood:
-1833.9
No. Observations:                1449   AIC:
3882.
Df Residuals:                    1342   BIC:
```

In [287]:
```
#Let's take a look at the graphs of some of the interactions of the selecte
pred = LR.predict(df[pv_cols])

df_copy['proba'] = pred

y_pred = pred.apply(lambda x: 1 if x > 0.5 else 0)

print('Fvalue: {:.6f}'.format(LR.fvalue))
print('MSE total on the train data: {:.4f}'.format(LR.mse_total))
def plot_proba(continous, predict, discret, data):
    grouped = pd.pivot_table(data, values=[predict], index=[continous, disc
    colors = 'rbgyrbgy'
    for col in data[discret].unique():
        plt_data = grouped.ix[grouped.index.get_level_values(1)==col]
        plt.plot(plt_data.index.get_level_values(0), plt_data[predict], col
    plt.xlabel(continous)
    plt.ylabel("Probabilities")
    plt.legend(np.sort(data[discret].unique()), loc='upper left', title=dis
    plt.title("Probabilities with " + continous + " and " + discret)

fig = plt.figure(figsize=(30, 20))
ax = fig.add_subplot(241); plot_proba('ConstructArea', 'SalePrice', 'Remod'
ax = fig.add_subplot(242); plot_proba('TotalPoints', 'SalePrice', 'ExterCon
ax = fig.add_subplot(243); plot_proba('BsmtFinType1', 'SalePrice', 'BsmtCon
ax = fig.add_subplot(245); plot_proba('TotalExtraPoints', 'SalePrice', 'Lot

plt.show()
del df_copy
```

```
Fvalue: 2453.619955
MSE total on the train data: 144.7138
```

Probabilities with ConstructArea and Remod
Remod
0
1
Probabilities
ConstructArea

Probabilities with TotalPoints and ExterCond
ExterCond
1
2
3
4
5
Probabilities
TotalPoints

Probabilities with BsmtFinType1 and BsmtCond
BsmtCond
0
1
2
3
4
Probabilities
BsmtFinType1

Probabilities with TotalExtraPoints and LotShape
LotShape
1
2
3
4
Probabilities
TotalExtraPoints

In [288]:
```
ls = Lasso(alpha = 0.0005, max_iter = 161, selection = 'cyclic', tol = 0.00
rfecv = RFECV(estimator=ls, n_jobs = -1, step=1, scoring = 'neg_mean_square
rfecv.fit(df, y_train)

select_features_rfecv = rfecv.get_support()
RFEcv = cols[select_features_rfecv]
print('{:d} Features Select by RFEcv:\n{:}'.format(rfecv.n_features_, RFEcv
```

```
79 Features Select by RFEcv:
['LotFrontage' 'LandSlope' 'OverallCond' 'YearBuilt' 'YearRemodAdd'
 'MasVnrArea' 'BsmtQual' 'BsmtCond' 'BsmtFinType1' 'BsmtFinType2'
 'HeatingQC' 'CentralAir' 'BsmtHalfBath' 'HalfBath' 'BedroomAbvGr'
 'KitchenQual' 'TotRmsAbvGrd' 'Functional' 'Fireplaces' 'GarageFinish'
 'PavedDrive' 'OpenPorchSF' 'EnclosedPorch' 'YrSold' 'TotalPoints'
 'GarageArea_x_Car' 'ConstructArea' 'LotAreaMultSlope' 'TotBathrooms'
 'MSSubClass_160' 'MSSubClass_20' 'MSSubClass_30' 'MSSubClass_50'
 'MSSubClass_70' 'MSZoning_FV' 'MSZoning_RM' 'Alley_Pave'
 'LandContour_Lvl' 'LotConfig_CulDSac' 'LotConfig_FR2'
 'Neighborhood_BrkSide' 'Neighborhood_ClearCr' 'Neighborhood_CollgCr'
 'Neighborhood_Crawfor' 'Neighborhood_Edwards' 'Neighborhood_IDOTRR'
 'Neighborhood_MeadowV' 'Neighborhood_Mitchel' 'Neighborhood_NoRidge'
 'Neighborhood_NridgHt' 'Neighborhood_OldTown' 'Neighborhood_Somerst'
 'Neighborhood_StoneBr' 'Condition1_Artery' 'Condition1_Norm'
 'Condition1_RRAe' 'RoofStyle_Hip' 'Exterior1st_BrkFace'
 'Exterior1st_HdBoard' 'Exterior1st_Plywood' 'Exterior1st_WdSdng'
 'Exterior2nd_HdBoard' 'Exterior2nd_MetalSd' 'Exterior2nd_Plywood'
 'MasVnrType_Stone' 'Foundation_BrkTil' 'Foundation_PConc'
 'Foundation_Slab' 'Heating_GasW' 'GarageType_BuiltIn' 'GarageType_NA'
 'MiscFeature_Shed' 'MoSold_11' 'MoSold_3' 'MoSold_5' 'MoSold_7'
 'SaleType_WD' 'SaleCondition_Abnorml' 'SaleCondition_Family']
```

In [289]:
```
from itertools import combinations
class SBS():
    def __init__(self, estimator, k_features, scoring=r2_score, test_size=0
        self.scoring = scoring
        self.estimator = clone(estimator)
        self.k_features = k_features
        self.test_size = test_size
        self.random_state = random_state

    def fit(self, X, y):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
        dim = X_train.shape[1]
        self.indices_ = list(range(dim))
        self.subsets_ = [self.indices_]
        score = self._calc_score(X_train, y_train, X_test, y_test, self.ind
        self.scores_ = [score]

        while dim > self.k_features:
            scores = []
            subsets = []
            for p in combinations(self.indices_, r=dim-1):
                score = self._calc_score(X_train, y_train, X_test, y_test,
                scores.append(score)
                subsets.append(list(p))

            best = np.argmax(scores)
            self.indices_ = subsets[best]
            self.subsets_.append(self.indices_)
            dim -= 1
            self.scores_.append(scores[best])

        self.k_score_ = self.scores_[-1]
        return self

    def transform(self, X):
        return X.iloc[:, self.indices_]

    def _calc_score(self, X_train, y_train, X_test, y_test, indices):
        self.estimator.fit(X_train.iloc[:, indices], y_train)
        y_pred = self.estimator.predict(X_test.iloc[:, indices])
        score = self.scoring(y_test, y_pred)
        return score

score = r2_score
ls = Lasso(alpha = 0.0005, max_iter = 161, selection = 'cyclic', tol = 0.00
sbs = SBS(ls, k_features=1, scoring= score)
sbs.fit(df, y_train)

k_feat = [len(k) for k in sbs.subsets_]
fig = plt.figure(figsize=(10,5))
plt.plot(k_feat, sbs.scores_, marker='o')
plt.xlim([1, len(sbs.subsets_)])
plt.xticks(np.arange(1, len(sbs.subsets_)+1))
plt.ylabel('R2 Score')
plt.xlabel('Number of features')
plt.grid(b=1)
```

```
plt.show()
print('Best Score: {:2.2%}\n'.format(max(sbs.scores_)))
print('Best score with:{0:2d}.\n'.\
      format(len(list(df.columns[sbs.subsets_[np.argmax(sbs.scores_)]]))))
SBS = list(df.columns[list(sbs.subsets_[max(np.arange(0, len(sbs.scores_))[
print('\nBest score with {0:2d} features:\n{1:}'.format(len(SBS), SBS))
```

```
Best Score: 93.18%

Best score with:52.


Best score with 52 features:
['OverallCond', 'MasVnrArea', 'BsmtCond', 'BsmtFinType2', 'CentralAir',
'HalfBath', 'KitchenQual', 'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'G
arageFinish', 'PavedDrive', 'EnclosedPorch', 'YrSold', 'TotalPoints', 'Co
nstructArea', 'LotAreaMultSlope', 'TotBathrooms', 'MSSubClass_160', 'MSSu
bClass_20', 'MSSubClass_60', 'LandContour_Lvl', 'LotConfig_CulDSac', 'Lot
Config_FR2', 'Neighborhood_BrkSide', 'Neighborhood_Crawfor', 'Neighborhoo
d_Edwards', 'Neighborhood_IDOTRR', 'Neighborhood_MeadowV', 'Neighborhood_
NoRidge', 'Neighborhood_NridgHt', 'Neighborhood_OldTown', 'Neighborhood_S
omerst', 'Neighborhood_StoneBr', 'Condition1_Norm', 'Condition1_RRAe', 'H
ouseStyle_SFoyer', 'HouseStyle_SLvl', 'RoofMatl_CompShg', 'Exterior1st_Wd
Sdng', 'Exterior2nd_HdBoard', 'Exterior2nd_Plywood', 'Foundation_BrkTil',
'Foundation_PConc', 'Foundation_Slab', 'Heating_GasW', 'GarageType_CarPor
t', 'MoSold_11', 'MoSold_3', 'MoSold_5', 'SaleCondition_Family', 'IsNew']
```

In [290]:
```
skb = SelectKBest(score_func=f_regression, k=80)
skb.fit(df, y_train)
select_features_kbest = skb.get_support()
kbest_FR = cols[select_features_kbest]
scores = skb.scores_[select_features_kbest]
feature_scores = pd.DataFrame([(item, score) for item, score in zip(kbest_F
fig = plt.figure(figsize=(40,20))
f1 = fig.add_subplot(121)
feature_scores.sort_values(by='score', ascending=True).plot(y = 'score', x
                                                            ax = f1, fontsi

skb = SelectKBest(score_func=mutual_info_regression, k=80)
skb.fit(df, y_train)
select_features_kbest = skb.get_support()
kbest_MIR = cols[select_features_kbest]
scores = skb.scores_[select_features_kbest]
feature_scores = pd.DataFrame([(item, score) for item, score in zip(kbest_F
f2 = fig.add_subplot(122)
feature_scores.sort_values(by='score', ascending=True).plot(y = 'score', x
                                                            ax = f2, fontsi
plt.show()
```

In [291]:
```
warnings.filterwarnings(action='ignore', category=DeprecationWarning)

# split data into train and test sets
X_train, X_test, y, y_test = train_test_split(df, y_train, test_size=0.30,

# fit model on all training data
#importance_type='gain'
model =  XGBRegressor(base_score=0.5, colsample_bylevel=1, colsample_bytree
                      random_state=101, min_child_weight=1, missing=None, n
                      scale_pos_weight=1, seed=None, silent=True, subsample


model.fit(X_train, y)
fig=plt.figure(figsize=(20,20))
ax = fig.add_subplot(121)
g = plot_importance(model, height=0.5, ax=ax)

# Using each unique importance as a threshold
thresholds = np.sort(np.unique(model.feature_importances_))
best = 1e36
colsbest = 31
my_model = model
threshold = 0

for thresh in thresholds:
    # select features using threshold
    selection = SelectFromModel(model, threshold=thresh, prefit=True)
    select_X_train = selection.transform(X_train)
    # train model
    selection_model =  XGBRegressor(base_score=0.5, colsample_bylevel=1, co
                                    random_state=101, min_child_weight=1, m
                                    scale_pos_weight=1, seed=None, silent=T
    selection_model.fit(select_X_train, y)
    # eval model
    select_X_test = selection.transform(X_test)
    y_pred = selection_model.predict(select_X_test)
    predictions = [round(value) for value in y_pred]
    r2 = r2_score(y_test, predictions)
    mse = mean_squared_error(y_test, predictions)
    print("Thresh={:1.3f}, n={:d}, R2: {:2.2%} with MSE: {:.4f}".format(thr
    if (best >= mse):
        best = mse
        colsbest = select_X_train.shape[1]
        my_model = selection_model
        threshold = thresh

ax = fig.add_subplot(122)
g = plot_importance(my_model,height=0.5, ax=ax,
                    title='The best MSE: {:1.4f} with {:d} features'.\
                    format(best, colsbest))

feature_importances = [(score, feature) for score, feature in zip(model.fea
XGBest = pd.DataFrame(sorted(sorted(feature_importances, reverse=True)[:col
g = XGBest.plot(x='Feature', kind='barh', figsize=(20,10), fontsize=14, gri
     title='Original feature importance from selected features')
plt.tight_layout(); plt.show()
```

```
XGBestCols = XGBest.iloc[:, 1].tolist()
```

```
Thresh=0.000, n=188, R2: 50.18% with MSE: 0.0832
Thresh=0.000, n=77, R2: 50.18% with MSE: 0.0832
Thresh=0.001, n=76, R2: 50.49% with MSE: 0.0827
Thresh=0.001, n=75, R2: 50.49% with MSE: 0.0827
Thresh=0.002, n=74, R2: 49.88% with MSE: 0.0837
Thresh=0.002, n=73, R2: 49.88% with MSE: 0.0837
Thresh=0.002, n=72, R2: 50.09% with MSE: 0.0833
Thresh=0.002, n=71, R2: 49.78% with MSE: 0.0839
Thresh=0.002, n=70, R2: 50.78% with MSE: 0.0822
Thresh=0.002, n=69, R2: 50.78% with MSE: 0.0822
Thresh=0.002, n=68, R2: 51.11% with MSE: 0.0816
Thresh=0.002, n=67, R2: 50.52% with MSE: 0.0826
Thresh=0.003, n=66, R2: 50.52% with MSE: 0.0826
Thresh=0.003, n=65, R2: 50.49% with MSE: 0.0827
Thresh=0.003, n=64, R2: 49.78% with MSE: 0.0838
Thresh=0.003, n=63, R2: 50.25% with MSE: 0.0831
Thresh=0.003, n=62, R2: 50.07% with MSE: 0.0834
Thresh=0.003, n=61, R2: 49.71% with MSE: 0.0840
Thresh=0.003, n=60, R2: 51.10% with MSE: 0.0817
```

In [292]:
```
bcols = set(pv_cols).union(set(RFEcv)).union(set(kbest_FR)).union(set(kbest
print('Features Selected by Filter Methods:\n')
print("Extra features select by Kbest_FR:", set(kbest_FR).\
     difference(set(pv_cols).union(set(RFEcv)).union(set(kbest_MIR)).union(
print("Extra features select by Kbest_MIR:", set(kbest_MIR).\
      difference(set(pv_cols).union(set(RFEcv)).union(set(kbest_FR)).union(
print('_'*75,'\nFeatures Selected by Wrappers Methods:\n')
print("Extra features select by pv_cols:", set(pv_cols).\
      difference(set(SBS).union(set(RFEcv)).union(set(kbest_MIR)).union(set
print("Extra features select by RFEcv:", set(RFEcv).\
      difference(set(pv_cols).union(set(kbest_FR)).union(set(kbest_MIR)).un
print("Extra features select by SBS:", set(SBS).\
      difference(set(pv_cols).union(set(RFEcv)).union(set(kbest_MIR)).union
print('_'*75,'\nFeatures Selected by Embedded Methods:\n')
print("Extra features select by XGBestCols:", set(XGBestCols).\
      difference(set(pv_cols).union(set(RFEcv)).union(set(kbest_MIR)).union
print('_'*75,'\nIntersection Features Selected:')
intersection = set(SBS).intersection(set(kbest_MIR)).intersection(set(RFEcv
               intersection(set(kbest_FR)).intersection(set(XGBestCols))
print(intersection, '\n')
print('_'*75,'\nUnion All Features Selected:')
print('Total number of features selected:', len(bcols))
print('\n{0:2d} features removed if use the union of selections: {1:}'.\
      format(len(cols.difference(bcols)), cols.difference(bcols)))
```

```
Features Selected by Filter Methods:

Extra features select by Kbest_FR: {'Heating_GasA', 'Heating_Grav', 'Cond
ition1_Feedr', 'Alley_NA'}

Extra features select by Kbest_MIR: {'ExterCond', 'Exterior1st_CemntBd'}

___________________________________________________________________________
__
Features Selected by Wrappers Methods:

Extra features select by pv_cols: {'RoofStyle_Gambrel', 'LotConfig_Insid
e', 'RoofStyle_Mansard', 'MoSold_8', 'MSSubClass_190', 'MSSubClass_180',
'HouseStyle_15Unf', 'Neighborhood_NPkVill', 'MiscFeature_Pool', 'Conditio
n1_RRAn', 'MoSold_6', 'Condition2_Artery', 'SaleCondition_AdjLand', 'Neig
hborhood_Blueste', 'MoSold_4', 'MiscFeature_Othr', 'MoSold_1', 'RoofMatl_
TarGrv', 'Neighborhood_SWISU', 'Exterior1st_AsphShn', 'Exterior2nd_BrkFac
e', 'Exterior1st_WdShing', 'MSSubClass_75', 'RoofStyle_Shed', 'MSSubClass
_40', 'MoSold_9', 'Neighborhood_Veenker', 'MSSubClass_85', 'Street', 'Ext
erior2nd_Stone', 'RoofMatl_WdShake', 'Exterior1st_Stucco'}

Extra features select by RFEcv: {'MiscFeature_Shed', 'Alley_Pave'}

Extra features select by SBS: set()

___________________________________________________________________________
__
Features Selected by Embedded Methods:

Extra features select by XGBestCols: {'FullBath', 'Exterior1st_BrkComm',
'GarageType_Basment'}
```

```
--------------------------------------------------------------------------------
__
Intersection Features Selected:
{'KitchenQual', 'TotRmsAbvGrd', 'GarageFinish', 'Neighborhood_IDOTRR', 'T
otBathrooms', 'ConstructArea', 'Neighborhood_Edwards'}


--------------------------------------------------------------------------------
__
Union All Features Selected:
Total number of features selected: 157

31 features removed if use the union of selections: Index(['BldgType_2fmC
on', 'BldgType_Twnhs', 'Condition1_PosA',
       'Condition1_RRNe', 'Condition1_RRNn', 'Condition2_Feedr',
       'Condition2_Norm', 'Condition2_PosA', 'Condition2_PosN',
       'Electrical_FuseP', 'Exterior2nd_AsphShn', 'Exterior2nd_BrkCmn',
       'Exterior2nd_CmentBd', 'Exterior2nd_ImStucc', 'Exterior2nd_Stucc
o',
       'Exterior2nd_WdShng', 'Foundation_Stone', 'Foundation_Wood',
       'Heating_Wall', 'HouseStyle_25Unf', 'LandContour_HLS',
       'LandContour_Low', 'LotConfig_FR3', 'MSZoning_RH', 'MiscVal',
       'SaleType_CWD', 'SaleType_Con', 'SaleType_ConLD', 'SaleType_ConL
I',
       'SaleType_ConLw', 'SaleType_Oth'],
      dtype='object')
```

In [293]:

```
totalCols = list(bcols.union(set(colsP)))
train = all_data.loc[all_data.SalePrice>0 , list(totalCols)].reset_index(dr
y_train = all_data.SalePrice[all_data.SalePrice>0].reset_index(drop=True, i
test = all_data.loc[all_data.SalePrice==0 , list(totalCols)].reset_index(dr
```

In [294]:

```
class select_fetaures(object): # BaseEstimator, TransformerMixin,
    def __init__(self, select_cols):
        self.select_cols_ = select_cols

    def fit(self, X, Y ):
        print('Recive {0:2d} features...'.format(X.shape[1]))
        return self

    def transform(self, X):
        print('Select {0:2d} features'.format(X.loc[:, self.select_cols_].s
        return X.loc[:, self.select_cols_]

    def fit_transform(self, X, Y):
        self.fit(X, Y)
        df = self.transform(X)
        return df
        #X.loc[:, self.select_cols_]

    def __getitem__(self, x):
        return self.X[x], self.Y[x]
```

In [295]:
```
scale = RobustScaler()
df = scale.fit_transform(train)

pca = PCA().fit(df) # whiten=True
print('With only 120 features: {:6.4%}'.format(sum(pca.explained_variance_r

print('After PCA, {:3} features only not explained {:6.4%} of variance rati


del df,all_data
```

```
With only 120 features: 99.8795% %

After PCA, 120 features only not explained 0.1205% of variance ratio from
the original 187
```

In [297]:
```
def get_results(model, name='NAN', log=False):

    rcols = ['Name','Model', 'BestParameters', 'Scorer', 'Index', 'BestScor
             'MeanScoreStd', 'Best']
    res = pd.DataFrame(columns=rcols)

    results = gs.cv_results_
    modelo = gs.best_estimator_

    scoring = {'MEA': 'neg_mean_absolute_error', 'R2': 'r2', 'RMSE': 'neg_m

    for scorer in sorted(scoring):
        best_index = np.nonzero(results['rank_test_%s' % scoring[scorer]] =
        if scorer == 'RMSE':
            best = np.sqrt(-results['mean_test_%s' % scoring[scorer]][best_
            best_std = np.sqrt(results['std_test_%s' % scoring[scorer]][bes
            scormean = np.sqrt(-results['mean_test_%s' % scoring[scorer]].m
            stdmean = np.sqrt(results['std_test_%s' % scoring[scorer]].mean
            if log:
                best = np.expm1(best)
                best_std = np.expm1(best_std)
                scormean = np.expm1(scormean)
                stdmean = np.expm1(stdmean)
        elif scorer == 'MEA':
            best = (-results['mean_test_%s' % scoring[scorer]][best_index])
            best_std = results['std_test_%s' % scoring[scorer]][best_index]
            scormean =(-results['mean_test_%s' % scoring[scorer]].mean())
            stdmean = results['std_test_%s' % scoring[scorer]].mean()
            if log:
                best = np.expm1(best)
                best_std = np.expm1(best_std)
                scormean = np.expm1(scormean)
                stdmean = np.expm1(stdmean)
        else:
            best = results['mean_test_%s' % scoring[scorer]][best_index]*10
            best_std = results['std_test_%s' % scoring[scorer]][best_index]
            scormean = results['mean_test_%s' % scoring[scorer]].mean()*100
            stdmean = results['std_test_%s' % scoring[scorer]].mean()*100

        r1 = pd.DataFrame([(name, modelo, gs.best_params_, scorer, best_inc
                           stdmean, gs.best_score_)],
                          columns = rcols)
        res = res.append(r1)

    if log:
        bestscore = np.expm1(np.sqrt(-gs.best_score_))
    else:
        bestscore = np.sqrt(-gs.best_score_)
    print("Best Score: {:.6f}".format(bestscore))
    print('----------------------------------------')
    print('Best Parameters:')
    print(gs.best_params_)

    return res
```

In [298]:
```
#Residuals Plots

def resilduals_plots(lr, X, Y, log=False):
    y_pred = lr.predict(X)
    residual = pd.DataFrame()
    residual['Predict'] = y_pred
    residual['Residual'] = Y - y_pred
    residual['Predicted'] = np.expm1(residual.Predict)
    residual['StdResidual'] = np.expm1(Y) - residual.Predicted
    residual.StdResidual = residual.StdResidual / residual.StdResidual.std(
    residual['IDX'] = X.index

    if log:
        fig = plt.figure(figsize=(20,10))
        ax = fig.add_subplot(121)
        g = sns.regplot(y='Residual', x='Predict', data = residual, order=1
        plt.xlabel('Log Predicted Values')
        plt.ylabel('Log Residuals')
        plt.hlines(y=0, xmin=min(Y)-1, xmax=max(Y)+1, lw=2, color='red')
        plt.xlim([min(Y)-1, max(Y)+1])

        ax = fig.add_subplot(122)
        g = sns.regplot(y='StdResidual', x='Predicted', data = residual, or
        plt.xlabel('Predicted Values')
        plt.ylabel('Standardized Residuals')
        plt.hlines(y=0, xmin=np.expm1(min(Y))-1, xmax=np.expm1(max(Y))+1, l
        plt.xlim([np.expm1(min(Y))-1, np.expm1(max(Y))+1])
    else:
        residual.StdResidual = residual.Residual / residual.Residual.std()
        residual.drop(['Residual', 'Predicted'], axis = 1, inplace=True)
        g = sns.regplot(y='StdResidual', x='Predict', data = residual, orde
        plt.xlabel('Predicted Values')
        plt.ylabel('Standardized Residuals')
        plt.hlines(y=0, xmin=min(Y)-1, xmax=max(Y)+1, lw=2, color='red')
        plt.xlim([min(Y)-1, max(Y)+1])

    plt.show()

    return residual
```

In [299]:
```
#Hyperparametrization
model = Pipeline([
        ('pca', PCA(random_state = 101)),
        ('model', Lasso(random_state = 101))])

SEL = list(set(RFEcv).union(set(colsP)))
n_components = [len(SEL)-5, len(SEL)-3, len(SEL)]
whiten = [False, True]
max_iter = [5] #, 10, 100, 200, 300, 400, 500, 600]
alpha = [0.0003, 0.0007, 0.0005, 0.05, 0.5, 1.0]
selection = ['random', 'cyclic']
tol = [2e-03, 0.003, 0.001, 0.0005]
param_grid =\
            dict(
                  model__alpha = alpha
                  ,model__max_iter = max_iter
                  ,model__selection = selection
                  ,model__tol = tol
                  ,pca__n_components = n_components
                  ,pca__whiten = whiten
                )

gs = GridSearchCV(estimator = model, param_grid = param_grid, refit = 'neg_
                   , scoring=list(['neg_mean_squared_error' , 'neg_mean_abs
                   ,cv=5, verbose=1, n_jobs=4)

lasso = Pipeline([
        ('sel', select_fetaures(select_cols=SEL)),
        ('scl', RobustScaler()),
        ('gs', gs)
 ])

lasso.fit(train,y_train)

results = get_results(lasso, 'lasso Lg1', log=True)
display(results.loc[:, 'Scorer' : 'MeanScoreStd'])
r = resilduals_plots(lasso, train, y_train, log=True)
```

```
Recive 187 features...
Select 109 features
Fitting 5 folds for each of 288 candidates, totalling 1440 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    1.9s

Best Score: 0.116418
----------------------------------------
Best Parameters:
{'model__alpha': 0.0007, 'model__max_iter': 5, 'model__selection': 'cycli
c', 'model__tol': 0.002, 'pca__n_components': 106, 'pca__whiten': True}

[Parallel(n_jobs=4)]: Done 1440 out of 1440 | elapsed:    8.1s finished
```

Scorer	Index	BestScore	BestScoreStd	MeanScore	MeanScoreStd

	Scorer	Index	BestScore	BestScoreStd	MeanScore	MeanScoreStd
0	MEA	75	0.080	0.003	0.153	0.006
0	R2	51	92.364	0.223	69.284	0.743
0	RMSE	75	0.116	0.026	0.247	0.065

```
Select 109 features
```

In [300]:
```
fica =  list(r.IDX[abs(r.Residual)<=0.3])
print('Outliers removed:', r.shape[0]-len(fica))
t = train.iloc[fica, :].reset_index(drop=True, inplace=False)
y_t = y_train.iloc[fica].reset_index(drop=True, inplace=False)

lasso.fit(t, y_t)
results = get_results(lasso, 'lasso Lg2', log=True)
display(results.loc[:, 'Scorer' : 'MeanScoreStd'])
r = resilduals_plots(lasso, t, y_t, log=True)
del  t, y_t, fica
```

```
Outliers removed: 18
Recive 187 features...
Select 109 features
Fitting 5 folds for each of 288 candidates, totalling 1440 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=4)]: Done 368 tasks      | elapsed:    2.0s

Best Score: 0.094187
----------------------------------------
Best Parameters:
{'model__alpha': 0.0007, 'model__max_iter': 5, 'model__selection': 'cycli
c', 'model__tol': 0.002, 'pca__n_components': 106, 'pca__whiten': True}

[Parallel(n_jobs=4)]: Done 1440 out of 1440 | elapsed:    7.6s finished
```

	Scorer	Index	BestScore	BestScoreStd	MeanScore	MeanScoreStd
0	MEA	51	0.072	0.004	0.146	0.006
0	R2	51	94.577	0.440	70.894	0.611
0	RMSE	75	0.094	0.023	0.232	0.060

```
Select 109 features
```

In [301]:
```
y = np.expm1(y_train)
lasso.fit(train, y)
results = get_results(lasso, 'lasso N1')
display(results.loc[:, 'Scorer' : 'MeanScoreStd'])
r = resilduals_plots(lasso, train, y)
```

```
Recive 187 features...
Select 109 features
Fitting 5 folds for each of 288 candidates, totalling 1440 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=4)]: Done 420 tasks      | elapsed:    2.3s

Best Score: 21504.641433
----------------------------------------
Best Parameters:
{'model__alpha': 1.0, 'model__max_iter': 5, 'model__selection': 'cyclic',
'model__tol': 0.002, 'pca__n_components': 106, 'pca__whiten': False}

[Parallel(n_jobs=4)]: Done 1440 out of 1440 | elapsed:    8.1s finished
```

	Scorer	Index	BestScore	BestScoreStd	MeanScore	MeanScoreStd
0	MEA	242	14582.133	854.804	14647.674	858.145
0	R2	242	92.600	0.579	92.537	0.526
0	RMSE	266	21504.641	7071.982	21600.052	7013.087

```
Select 109 features
```

In [302]:
```
fica =  list(r.IDX[abs(r.StdResidual)<3]) # =2.7
print('Outliers removed:', r.shape[0]-len(fica))
t = train.iloc[fica, :].reset_index(drop=True, inplace=False)
y_l = y_train.iloc[fica].reset_index(drop=True, inplace=False)
y_n = np.expm1(y_l)

lasso.fit(t, y_n)
results = get_results(lasso, 'lasso N2')
display(results.loc[:, 'Scorer' : 'MeanScoreStd'])
r2 = resilduals_plots(lasso, t, y_n)
del fica, r2
```

```
Outliers removed: 23
Recive 187 features...
Select 109 features
Fitting 5 folds for each of 288 candidates, totalling 1440 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=4)]: Done 248 tasks      | elapsed:    1.6s

Best Score: 17597.954430
----------------------------------------
Best Parameters:
{'model__alpha': 1.0, 'model__max_iter': 5, 'model__selection': 'random',
'model__tol': 0.002, 'pca__n_components': 106, 'pca__whiten': False}

[Parallel(n_jobs=4)]: Done 1440 out of 1440 | elapsed:    8.2s finished
```

	Scorer	Index	BestScore	BestScoreStd	MeanScore	MeanScoreStd
0	MEA	242	12966.314	1004.908	13033.307	980.243
0	R2	242	94.283	0.795	94.212	0.792
0	RMSE	242	17597.954	6826.881	17714.918	6968.298

```
Select 109 features
```

4
3
2
1
0
−1
−2
−3
Standardized Residuals
100000
200000
300000
400000
500000
600000
700000
Predicted Values

In [303]:
```
y_log = y_train.copy()
y_train = np.expm1(y_train)

lasso.fit(train, y_train)

results = get_results(lasso, 'lasso', log=False)
display(results.loc[:, 'Scorer' : 'MeanScoreStd'])
r = resilduals_plots(lasso, train, y_train, log=False)
```

```
Recive 187 features...
Select 109 features
Fitting 5 folds for each of 288 candidates, totalling 1440 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent worker
s.
[Parallel(n_jobs=4)]: Done 472 tasks      | elapsed:    3.0s

Best Score: 21504.641433
----------------------------------------
Best Parameters:
{'model__alpha': 1.0, 'model__max_iter': 5, 'model__selection': 'cyclic',
'model__tol': 0.002, 'pca__n_components': 106, 'pca__whiten': False}

[Parallel(n_jobs=4)]: Done 1440 out of 1440 | elapsed:    8.6s finished
```

	Scorer	Index	BestScore	BestScoreStd	MeanScore	MeanScoreStd
0	MEA	242	14582.133	854.804	14647.674	858.145
0	R2	242	92.600	0.579	92.537	0.526
0	RMSE	266	21504.641	7071.982	21600.052	7013.087

```
Select 109 features
```

8
6
4
2
0
−2
−4
−6
Standardized Residuals
100000
200000
300000
400000
500000
600000
700000
Predicted Values

```
In [ ]:
```