Ref.

[https://www.kaggle.com/juliencs/a-study-on-regression-applied-to-the-ames-dataset (https://www.kaggle.com/juliencs/a-study-on-regression-applied-to-the-ames-dataset)](https://www.kaggle.com/juliencs/a-study-on-regression-applied-to-the-ames-dataset)

In [1]:
```
# Imports
import pandas as pd
import numpy as np

from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV, ElasticNetCV
from sklearn.metrics import mean_squared_error, make_scorer
from scipy.stats import skew
from IPython.display import display

import matplotlib.pyplot as plt
import seaborn as sns

# Definitions
pd.set_option('display.float_format', lambda x: '%.3f' % x)
%matplotlib inline
#njobs = 4
```

In [2]:
```
# Get data
train = pd.read_csv("/home/hduser/jupyter/Comprehensive_data_exploration_with_Pyt
print("train : " + str(train.shape))
```

```
train : (1460, 81)
```

In [3]:
```
# Check for duplicates
idsUnique = len(set(train['Id']))
idsUnique
```

Out[3]:
```
1460
```

In [4]:
```
idsTotal = train.shape[0]
idsTotal
```

Out[4]:
```
1460
```

In [5]:
```
idsDupli = idsTotal - idsUnique
```

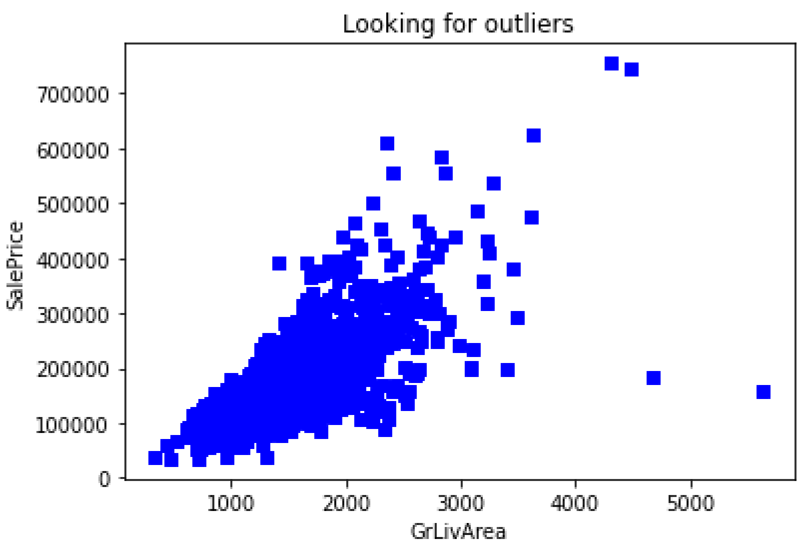
In [6]:
```
print("There are " + str(idsDupli) + " duplicate IDs for " + str(idsTotal) + " to
```

```
There are 0 duplicate IDs for 1460 total entries
```

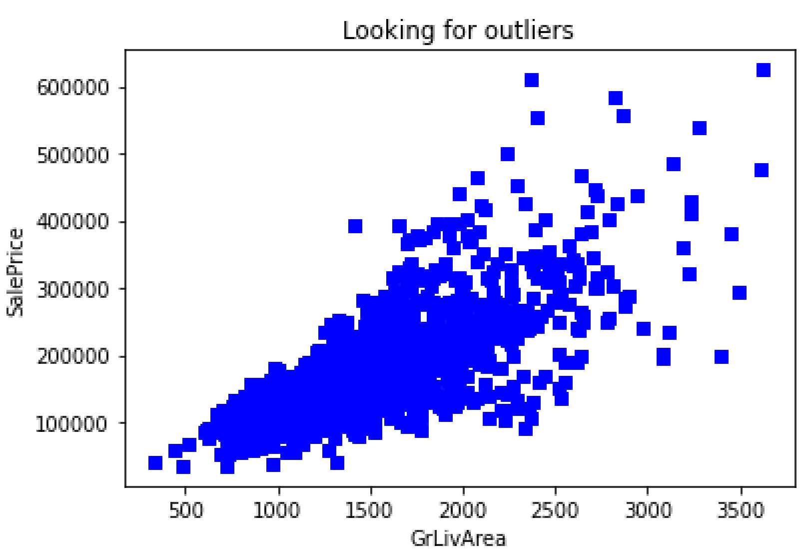
In [7]:
```
# Drop Id column
train.drop("Id", axis = 1, inplace = True)
```

Preprocessing

In [8]:
```
# Looking for outliers, as indicated in https://ww2.amstat.org/publications/jse/v
plt.scatter(train['GrLivArea'], train['SalePrice'], c = 'blue', marker = 's')
plt.title('Looking for outliers')
plt.xlabel('GrLivArea')
plt.ylabel('SalePrice')
plt.show()

train = train[train['GrLivArea'] < 4000]
```



In [9]:
```
plt.scatter(train.GrLivArea, train.SalePrice, c = "blue", marker = "s")
plt.title("Looking for outliers")
plt.xlabel("GrLivArea")
plt.ylabel("SalePrice")
plt.show()
```



In [10]:
```
train['SalePrice']
```

Out[10]:
```
0       208500
1       181500
2       223500
3       140000
4       250000
         ...
1455    175000
1456    210000
1457    266500
1458    142125
1459    147500
Name: SalePrice, Length: 1456, dtype: int64
```

In [11]:
```
# Log transform the target for official scoring
train['SalePrice'] = np.log1p(train['SalePrice'])
y = train['SalePrice']
```

In [12]:
```
train['SalePrice']
```

Out[12]:
```
0       12.248
1       12.109
2       12.317
3       11.849
4       12.429
         ...
1455    12.073
1456    12.255
1457    12.493
1458    11.864
1459    11.902
Name: SalePrice, Length: 1456, dtype: float64
```

Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.

In [13]:
```
# Handle missing values for features where median/mean or most common value doesn

# Alley : data description says NA means "no alley access"
train.loc[:, "Alley"] = train.loc[:, "Alley"].fillna("None")

# BedroomAbvGr : NA most likely means 0
train.loc[:, 'BedroomAbvGr'] = train.loc[:, 'BedroomAbvGr'].fillna(0)

# BsmtQual etc : data description says NA for basement features is "no basement"
train.loc[:, 'BsmtQual'] = train.loc[:, 'BsmtQual'].fillna('No')
train.loc[:, "BsmtCond"] = train.loc[:, "BsmtCond"].fillna("No")
train.loc[:, "BsmtExposure"] = train.loc[:, "BsmtExposure"].fillna("No")
train.loc[:, "BsmtFinType1"] = train.loc[:, "BsmtFinType1"].fillna("No")
train.loc[:, "BsmtFinType2"] = train.loc[:, "BsmtFinType2"].fillna("No")
train.loc[:, "BsmtFullBath"] = train.loc[:, "BsmtFullBath"].fillna(0)
train.loc[:, "BsmtHalfBath"] = train.loc[:, "BsmtHalfBath"].fillna(0)
train.loc[:, "BsmtUnfSF"] = train.loc[:, "BsmtUnfSF"].fillna(0)
# CentralAir : NA most likely means No
train.loc[:, "CentralAir"] = train.loc[:, "CentralAir"].fillna("N")
# Condition : NA most likely means Normal
train.loc[:, "Condition1"] = train.loc[:, "Condition1"].fillna("Norm")
train.loc[:, "Condition2"] = train.loc[:, "Condition2"].fillna("Norm")
# EnclosedPorch : NA most likely means no enclosed porch
train.loc[:, "EnclosedPorch"] = train.loc[:, "EnclosedPorch"].fillna(0)
# External stuff : NA most likely means average
train.loc[:, "ExterCond"] = train.loc[:, "ExterCond"].fillna("TA")
train.loc[:, "ExterQual"] = train.loc[:, "ExterQual"].fillna("TA")
# Fence : data description says NA means "no fence"
train.loc[:, "Fence"] = train.loc[:, "Fence"].fillna("No")
# FireplaceQu : data description says NA means "no fireplace"
train.loc[:, "FireplaceQu"] = train.loc[:, "FireplaceQu"].fillna("No")
train.loc[:, "Fireplaces"] = train.loc[:, "Fireplaces"].fillna(0)
# Functional : data description says NA means typical
train.loc[:, "Functional"] = train.loc[:, "Functional"].fillna("Typ")
# GarageType etc : data description says NA for garage features is "no garage"
train.loc[:, "GarageType"] = train.loc[:, "GarageType"].fillna("No")
train.loc[:, "GarageFinish"] = train.loc[:, "GarageFinish"].fillna("No")
train.loc[:, "GarageQual"] = train.loc[:, "GarageQual"].fillna("No")
train.loc[:, "GarageCond"] = train.loc[:, "GarageCond"].fillna("No")
train.loc[:, "GarageArea"] = train.loc[:, "GarageArea"].fillna(0)
train.loc[:, "GarageCars"] = train.loc[:, "GarageCars"].fillna(0)
# HalfBath : NA most likely means no half baths above grade
train.loc[:, "HalfBath"] = train.loc[:, "HalfBath"].fillna(0)
# HeatingQC : NA most likely means typical
train.loc[:, "HeatingQC"] = train.loc[:, "HeatingQC"].fillna("TA")
# KitchenAbvGr : NA most likely means 0
train.loc[:, "KitchenAbvGr"] = train.loc[:, "KitchenAbvGr"].fillna(0)
# KitchenQual : NA most likely means typical
train.loc[:, "KitchenQual"] = train.loc[:, "KitchenQual"].fillna("TA")
# LotFrontage : NA most likely means no lot frontage
train.loc[:, "LotFrontage"] = train.loc[:, "LotFrontage"].fillna(0)
# LotShape : NA most likely means regular
train.loc[:, "LotShape"] = train.loc[:, "LotShape"].fillna("Reg")
# MasVnrType : NA most likely means no veneer
train.loc[:, "MasVnrType"] = train.loc[:, "MasVnrType"].fillna("None")
train.loc[:, "MasVnrArea"] = train.loc[:, "MasVnrArea"].fillna(0)
```

```
# MiscFeature : data description says NA means "no misc feature"
train.loc[:, "MiscFeature"] = train.loc[:, "MiscFeature"].fillna("No")
train.loc[:, "MiscVal"] = train.loc[:, "MiscVal"].fillna(0)
# OpenPorchSF : NA most likely means no open porch
train.loc[:, "OpenPorchSF"] = train.loc[:, "OpenPorchSF"].fillna(0)
# PavedDrive : NA most likely means not paved
train.loc[:, "PavedDrive"] = train.loc[:, "PavedDrive"].fillna("N")
# PoolQC : data description says NA means "no pool"
train.loc[:, "PoolQC"] = train.loc[:, "PoolQC"].fillna("No")
train.loc[:, "PoolArea"] = train.loc[:, "PoolArea"].fillna(0)
# SaleCondition : NA most likely means normal sale
train.loc[:, "SaleCondition"] = train.loc[:, "SaleCondition"].fillna("Normal")
# ScreenPorch : NA most likely means no screen porch
train.loc[:, "ScreenPorch"] = train.loc[:, "ScreenPorch"].fillna(0)
# TotRmsAbvGrd : NA most likely means 0
train.loc[:, "TotRmsAbvGrd"] = train.loc[:, "TotRmsAbvGrd"].fillna(0)
# Utilities : NA most likely means all public utilities
train.loc[:, "Utilities"] = train.loc[:, "Utilities"].fillna("AllPub")
# WoodDeckSF : NA most likely means no wood deck
train.loc[:, "WoodDeckSF"] = train.loc[:, "WoodDeckSF"].fillna(0)
```

In [14]:

```
# Some numerical features are actually really categories
train = train.replace({'MSSubClass' : {20 : "SC20", 30 : "SC30", 40 : "SC40", 45
                                       50 : "SC50", 60 : "SC60", 70 : "SC70", 75
                                       80 : "SC80", 85 : "SC85", 90 : "SC90", 12(
                                       150 : "SC150", 160 : "SC160", 180 : "SC18(
                      'MoSold' : {1 : "Jan", 2 : "Feb", 3 : "Mar", 4 : "Apr", 5 :
                                  7 : "Jul", 8 : "Aug", 9 : "Sep", 10 : "Oct", 1
                      })
```

In [15]:
```
# Encode some categorical features as ordered numbers when there is information i

train = train.replace({"Alley" : {"Grvl" : 1, "Pave" : 2},
                      "BsmtCond" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd" 
                      "BsmtExposure" : {"No" : 0, "Mn" : 1, "Av": 2, "Gd" : 3},
                      "BsmtFinType1" : {"No" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3,
                                        "ALQ" : 5, "GLQ" : 6},
                      "BsmtFinType2" : {"No" : 0, "Unf" : 1, "LwQ": 2, "Rec" : 3,
                                        "ALQ" : 5, "GLQ" : 6},
                      "BsmtQual" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA": 3, "Gd" :
                      "ExterCond" : {"Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" :
                      "ExterQual" : {"Po" : 1, "Fa" : 2, "TA": 3, "Gd": 4, "Ex" :
                      "FireplaceQu" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "G
                      "Functional" : {"Sal" : 1, "Sev" : 2, "Maj2" : 3, "Maj1" :
                                      "Min2" : 6, "Min1" : 7, "Typ" : 8},
                      "GarageCond" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd
                      "GarageQual" : {"No" : 0, "Po" : 1, "Fa" : 2, "TA" : 3, "Gd
                      "HeatingQC" : {"Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "Ex"
                      "KitchenQual" : {"Po" : 1, "Fa" : 2, "TA" : 3, "Gd" : 4, "E
                      "LandSlope" : {"Sev" : 1, "Mod" : 2, "Gtl" : 3},
                      "LotShape" : {"IR3" : 1, "IR2" : 2, "IR1" : 3, "Reg" : 4},
                      "PavedDrive" : {"N" : 0, "P" : 1, "Y" : 2},
                      "PoolQC" : {"No" : 0, "Fa" : 1, "TA" : 2, "Gd" : 3, "Ex" :
                      "Street" : {"Grvl" : 1, "Pave" : 2},
                      "Utilities" : {"ELO" : 1, "NoSeWa" : 2, "NoSewr" : 3, "AllP
                      })
```

In [16]:
```
# Create new features
# 1* Simplifications of existing features
train["SimplOverallQual"] = train.OverallQual.replace({1 : 1, 2 : 1, 3 : 1, # bad
                                                       4 : 2, 5 : 2, 6 : 2, # ave
                                                       7 : 3, 8 : 3, 9 : 3, 10 :
                                                      })
train["SimplOverallCond"] = train.OverallCond.replace({1 : 1, 2 : 1, 3 : 1, # bad
                                                       4 : 2, 5 : 2, 6 : 2, # ave
                                                       7 : 3, 8 : 3, 9 : 3, 10 :
                                                      })
train["SimplPoolQC"] = train.PoolQC.replace({1 : 1, 2 : 1, # average
                                             3 : 2, 4 : 2 # good
                                            })
train["SimplGarageCond"] = train.GarageCond.replace({1 : 1, # bad
                                                     2 : 1, 3 : 1, # average
                                                     4 : 2, 5 : 2 # good
                                                    })
train["SimplGarageQual"] = train.GarageQual.replace({1 : 1, # bad
                                                     2 : 1, 3 : 1, # average
                                                     4 : 2, 5 : 2 # good
                                                    })
train["SimplFireplaceQu"] = train.FireplaceQu.replace({1 : 1, # bad
                                                       2 : 1, 3 : 1, # average
                                                       4 : 2, 5 : 2 # good
                                                      })
train["SimplFireplaceQu"] = train.FireplaceQu.replace({1 : 1, # bad
                                                       2 : 1, 3 : 1, # average
                                                       4 : 2, 5 : 2 # good
                                                      })
train["SimplFunctional"] = train.Functional.replace({1 : 1, 2 : 1, # bad
                                                     3 : 2, 4 : 2, # major
                                                     5 : 3, 6 : 3, 7 : 3, # minor
                                                     8 : 4 # typical
                                                    })
train["SimplKitchenQual"] = train.KitchenQual.replace({1 : 1, # bad
                                                       2 : 1, 3 : 1, # average
                                                       4 : 2, 5 : 2 # good
                                                      })
train["SimplHeatingQC"] = train.HeatingQC.replace({1 : 1, # bad
                                                   2 : 1, 3 : 1, # average
                                                   4 : 2, 5 : 2 # good
                                                  })
train["SimplBsmtFinType1"] = train.BsmtFinType1.replace({1 : 1, # unfinished
                                                         2 : 1, 3 : 1, # rec room
                                                         4 : 2, 5 : 2, 6 : 2 # li
                                                        })
train["SimplBsmtFinType2"] = train.BsmtFinType2.replace({1 : 1, # unfinished
                                                         2 : 1, 3 : 1, # rec room
                                                         4 : 2, 5 : 2, 6 : 2 # li
                                                        })
train["SimplBsmtCond"] = train.BsmtCond.replace({1 : 1, # bad
                                                 2 : 1, 3 : 1, # average
                                                 4 : 2, 5 : 2 # good
                                                })
train["SimplBsmtQual"] = train.BsmtQual.replace({1 : 1, # bad
                                                 2 : 1, 3 : 1, # average
```

```
                                                    4 : 2, 5 : 2 # good
                                                   })
train["SimplExterCond"] = train.ExterCond.replace({1 : 1, # bad
                                                     2 : 1, 3 : 1, # average
                                                     4 : 2, 5 : 2 # good
                                                    })
train["SimplExterQual"] = train.ExterQual.replace({1 : 1, # bad
                                                     2 : 1, 3 : 1, # average
                                                     4 : 2, 5 : 2 # good
                                                    })

# 2* Combinations of existing features

# Overall quality of the house
train["OverallGrade"] = train["OverallQual"] * train["OverallCond"]
# Overall quality of the garage
train["GarageGrade"] = train["GarageQual"] * train["GarageCond"]
# Overall quality of the exterior
train["ExterGrade"] = train["ExterQual"] * train["ExterCond"]
# Overall kitchen score
train["KitchenScore"] = train["KitchenAbvGr"] * train["KitchenQual"]
# Overall fireplace score
train["FireplaceScore"] = train["Fireplaces"] * train["FireplaceQu"]
# Overall garage score
train["GarageScore"] = train["GarageArea"] * train["GarageQual"]
# Overall pool score
train["PoolScore"] = train["PoolArea"] * train["PoolQC"]
# Simplified overall quality of the house
train["SimplOverallGrade"] = train["SimplOverallQual"] * train["SimplOverallCond'
# Simplified overall quality of the exterior
train["SimplExterGrade"] = train["SimplExterQual"] * train["SimplExterCond"]
# Simplified overall pool score
train["SimplPoolScore"] = train["PoolArea"] * train["SimplPoolQC"]
# Simplified overall garage score
train["SimplGarageScore"] = train["GarageArea"] * train["SimplGarageQual"]
# Simplified overall fireplace score
train["SimplFireplaceScore"] = train["Fireplaces"] * train["SimplFireplaceQu"]
# Simplified overall kitchen score
train["SimplKitchenScore"] = train["KitchenAbvGr"] * train["SimplKitchenQual"]
# Total number of bathrooms
train["TotalBath"] = train["BsmtFullBath"] + (0.5 * train["BsmtHalfBath"]) + \
train["FullBath"] + (0.5 * train["HalfBath"])
# Total SF for house (incl. basement)
train["AllSF"] = train["GrLivArea"] + train["TotalBsmtSF"]
# Total SF for 1st + 2nd floors
train["AllFlrsSF"] = train["1stFlrSF"] + train["2ndFlrSF"]
# Total SF for porch
train["AllPorchSF"] = train["OpenPorchSF"] + train["EnclosedPorch"] + \
train["3SsnPorch"] + train["ScreenPorch"]
# Has masonry veneer or not
train["HasMasVnr"] = train.MasVnrType.replace({"BrkCmn" : 1, "BrkFace" : 1, "CBlc
                                               "Stone" : 1, "None" : 0})
# House completed before sale or not
train["BoughtOffPlan"] = train.SaleCondition.replace({"Abnorml" : 0, "Alloca" : 0
                                                      "Family" : 0, "Normal" : 0,
```

In [17]:
```
# Find most important features relative to target
print("Find most important features relative to target")
corr = train.corr()
corr.sort_values(['SalePrice'], ascending = False, inplace = True)
print(corr['SalePrice'])
```

```
Find most important features relative to target
SalePrice         1.000
OverallQual       0.819
AllSF             0.817
AllFlrsSF         0.729
GrLivArea         0.719
                  ...
LandSlope        -0.040
SimplExterCond   -0.042
KitchenAbvGr     -0.148
EnclosedPorch    -0.149
LotShape         -0.286
Name: SalePrice, Length: 87, dtype: float64
```

In [18]:
```
# Create new features
# 3* Polynomials on the top 10 existing features
train['OverallQual-s2'] = train['OverallQual'] ** 2
train['OverallQual-s3'] = train['OverallQual'] ** 3
train['OverallQual-Sq'] = np.sqrt(train['OverallQual'])
train["AllSF-2"] = train["AllSF"] ** 2
train["AllSF-3"] = train["AllSF"] ** 3
train["AllSF-Sq"] = np.sqrt(train["AllSF"])
train["AllFlrsSF-2"] = train["AllFlrsSF"] ** 2
train["AllFlrsSF-3"] = train["AllFlrsSF"] ** 3
train["AllFlrsSF-Sq"] = np.sqrt(train["AllFlrsSF"])
train["GrLivArea-2"] = train["GrLivArea"] ** 2
train["GrLivArea-3"] = train["GrLivArea"] ** 3
train["GrLivArea-Sq"] = np.sqrt(train["GrLivArea"])
train["SimplOverallQual-s2"] = train["SimplOverallQual"] ** 2
train["SimplOverallQual-s3"] = train["SimplOverallQual"] ** 3
train["SimplOverallQual-Sq"] = np.sqrt(train["SimplOverallQual"])
train["ExterQual-2"] = train["ExterQual"] ** 2
train["ExterQual-3"] = train["ExterQual"] ** 3
train["ExterQual-Sq"] = np.sqrt(train["ExterQual"])
train["GarageCars-2"] = train["GarageCars"] ** 2
train["GarageCars-3"] = train["GarageCars"] ** 3
train["GarageCars-Sq"] = np.sqrt(train["GarageCars"])
train["TotalBath-2"] = train["TotalBath"] ** 2
train["TotalBath-3"] = train["TotalBath"] ** 3
train["TotalBath-Sq"] = np.sqrt(train["TotalBath"])
train["KitchenQual-2"] = train["KitchenQual"] ** 2
train["KitchenQual-3"] = train["KitchenQual"] ** 3
train["KitchenQual-Sq"] = np.sqrt(train["KitchenQual"])
train["GarageScore-2"] = train["GarageScore"] ** 2
train["GarageScore-3"] = train["GarageScore"] ** 3
train["GarageScore-Sq"] = np.sqrt(train["GarageScore"])
```

In [19]:
```
# Differentiate numerical features (minus the target) and categorical features
categorical_features = train.select_dtypes(include = ['object']).columns
numerical_features = train.select_dtypes(exclude = ['object']).columns
numerical_features = numerical_features.drop('SalePrice')
print("Numerical features : " + str(len(numerical_features)))
print("Categorical features : " + str(len(categorical_features)))
train_num = train[numerical_features]
train_cat = train[categorical_features]
```

```
Numerical features : 116
Categorical features : 27
```

In [20]:
```
# Handle remaining missing values for numerical features by using median as repla
print("NAs for numerical features in train : " + str(train_num.isnull().values.su
train_num = train_num.fillna(train_num.median())
print("Remaining NAs for numerical features in train : " + str(train_num.isnull()
```

```
NAs for numerical features in train : 81
Remaining NAs for numerical features in train : 0
```

In [21]:
```
# Log transform of the skewed numerical features to lessen impact of outliers
# Inspired by Alexandru Papiu's script : https://www.kaggle.com/apapiu/house-pric
# As a general rule of thumb, a skewness with an absolute value > 0.5 is consider

skewness = train_num.apply(lambda x: skew(x))
skewness
```

Out[21]:
```
LotFrontage       -0.006
LotArea           12.575
Street           -15.481
LotShape          -1.290
Utilities        -38.118
                    ...
KitchenQual-3      1.229
KitchenQual-Sq     0.140
GarageScore-2      2.403
GarageScore-3      5.268
GarageScore-Sq    -1.494
Length: 116, dtype: float64
```

In [22]:
```
skewness = skewness[abs(skewness) > 0.5]
skewness
```

Out[22]:
```
LotArea           12.575
Street           -15.481
LotShape          -1.290
Utilities        -38.118
LandSlope         -4.801
                   ...
KitchenQual-2      0.812
KitchenQual-3      1.229
GarageScore-2      2.403
GarageScore-3      5.268
GarageScore-Sq    -1.494
Length: 85, dtype: float64
```

In [23]:
```
print(str(skewness.shape[0]) + " skewed numerical features to log transform")
skewed_features = skewness.index
skewed_features
```

```
85 skewed numerical features to log transform
```

Out[23]:
```
Index(['LotArea', 'Street', 'LotShape', 'Utilities', 'LandSlope',
       'OverallCond', 'YearBuilt', 'MasVnrArea', 'ExterQual', 'ExterCond',
       'BsmtQual', 'BsmtExposure', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2',
       'BsmtUnfSF', 'HeatingQC', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'HalfBath', 'KitchenAbvGr',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'GarageYrBlt', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'MiscVal', 'SimplOverallCond', 'SimplPoolQC', 'SimplGarageCond',
       'SimplGarageQual', 'SimplFunctional', 'SimplHeatingQC',
       'SimplBsmtFinType1', 'SimplBsmtFinType2', 'SimplBsmtCond',
       'SimplExterCond', 'SimplExterQual', 'GarageGrade', 'ExterGrade',
       'KitchenScore', 'FireplaceScore', 'PoolScore', 'SimplExterGrade',
       'SimplPoolScore', 'SimplGarageScore', 'SimplFireplaceScore', 'AllSF',
       'AllFlrsSF', 'AllPorchSF', 'BoughtOffPlan', 'OverallQual-s2',
       'OverallQual-s3', 'AllSF-2', 'AllSF-3', 'AllFlrsSF-2', 'AllFlrsSF-3',
       'GrLivArea-2', 'GrLivArea-3', 'ExterQual-2', 'ExterQual-3',
       'ExterQual-Sq', 'GarageCars-2', 'GarageCars-3', 'GarageCars-Sq',
       'TotalBath-2', 'TotalBath-3', 'KitchenQual-2', 'KitchenQual-3',
       'GarageScore-2', 'GarageScore-3', 'GarageScore-Sq'],
      dtype='object')
```

In [24]:
```
train_num[skewed_features] = np.log1p(train_num[skewed_features])
train_num[skewed_features]
```

Out[24]:

	LotArea	Street	LotShape	Utilities	LandSlope	OverallCond	YearBuilt	MasVnrArea	ExterQ
0	9.042	1.099	1.609	1.609	1.386	1.792	7.603	5.283	1.6
1	9.170	1.099	1.609	1.609	1.386	2.197	7.589	0.000	1.3
2	9.328	1.099	1.386	1.609	1.386	1.792	7.602	5.094	1.6
3	9.164	1.099	1.386	1.609	1.386	1.792	7.558	0.000	1.3
4	9.565	1.099	1.386	1.609	1.386	1.792	7.601	5.861	1.6
...	...	...	...	...	...	...	...	...	
1455	8.977	1.099	1.609	1.609	1.386	1.792	7.601	0.000	1.3
1456	9.486	1.099	1.609	1.609	1.386	1.946	7.590	4.787	1.3
1457	9.110	1.099	1.609	1.609	1.386	2.303	7.571	0.000	1.7
1458	9.182	1.099	1.609	1.609	1.386	1.946	7.576	0.000	1.3
1459	9.204	1.099	1.609	1.609	1.386	1.946	7.584	0.000	1.6

1456 rows × 85 columns

In [25]:
```
# Create dummy features for categorical values via one-hot encoding
print("NAs for categorical features in train : " + str(train_cat.isnull().values.
train_cat = pd.get_dummies(train_cat)
print("Remaining NAs for categorical features in train : " + str(train_cat.isnull
```

```
NAs for categorical features in train : 1
Remaining NAs for categorical features in train : 0
```

Modeling

In [26]:
```
# Join categorical and numerical features
train = pd.concat([train_num, train_cat], axis = 1)
print("New number of features : " + str(train.shape[1]))

# Partition the dataset in train + validation sets
X_train, X_test, y_train, y_test = train_test_split(train, y, test_size = 0.3, ra
print("X_train : " + str(X_train.shape))
print("X_test : " + str(X_test.shape))
print("y_train : " + str(y_train.shape))
print("y_test : " + str(y_test.shape))
```

```
New number of features : 323
X_train : (1019, 323)
X_test : (437, 323)
y_train : (1019,)
y_test : (437,)
```

In [27]:
```
X_train
```

Out[27]:

	LotFrontage	LotArea	Street	LotShape	Utilities	LandSlope	OverallQual	OverallCond	Year
328	0.000	9.383	1.099	1.386	1.609	1.386	6	1.946	7
1026	73.000	9.138	1.099	1.609	1.609	1.386	5	1.792	7
843	80.000	8.987	1.099	1.609	1.609	1.386	5	1.609	7
994	96.000	9.430	1.099	1.609	1.609	1.386	10	1.792	7
1226	86.000	9.589	1.099	1.386	1.609	1.386	6	1.792	7
...	...	...	...	...	...	...	...	...	
765	75.000	9.588	1.099	1.386	1.609	1.386	9	1.792	7
837	21.000	7.427	1.099	1.609	1.609	1.386	6	1.792	7
1219	21.000	7.427	1.099	1.609	1.609	1.386	6	1.792	7
560	0.000	9.336	1.099	1.386	1.609	1.386	5	1.946	7
685	0.000	8.530	1.099	1.386	1.609	1.386	7	1.792	7

1019 rows × 323 columns

In [28]:
```
# Standardize numerical features
stdSc = StandardScaler()
X_train.loc[:, numerical_features] = stdSc.fit_transform(X_train.loc[:, numerica
X_test.loc[:, numerical_features] = stdSc.transform(X_test.loc[:, numerical_featu
```

```
/usr/local/lib64/python3.6/site-packages/pandas/core/indexing.py:494: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  self.obj[item] = s
/usr/local/lib64/python3.6/site-packages/pandas/core/indexing.py:494: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  self.obj[item] = s
```

Standardization cannot be done before the partitioning, as we don't want to fit the StandardScaler on some observations that will later be used in the test set.

In [29]:
```
X_train
```

Out[29]:

	LotFrontage	LotArea	Street	LotShape	Utilities	LandSlope	OverallQual	OverallCond	Year
328	-1.720	0.533	0.063	-0.930	0.031	0.227	-0.058	0.475	-1
1026	0.467	0.031	0.063	0.671	0.031	0.227	-0.794	-0.421	-(
843	0.677	-0.277	0.063	0.671	0.031	0.227	-0.794	-1.480	-(
994	1.156	0.629	0.063	0.671	0.031	0.227	2.886	-0.421	1
1226	0.856	0.953	0.063	-0.930	0.031	0.227	-0.058	-0.421	1
...	...	...	...	...	...	...	...	...	
765	0.527	0.951	0.063	-0.930	0.031	0.227	2.150	-0.421	1
837	-1.091	-3.467	0.063	0.671	0.031	0.227	-0.058	-0.421	(
1219	-1.091	-3.467	0.063	0.671	0.031	0.227	-0.058	-0.421	(
560	-1.720	0.437	0.063	-0.930	0.031	0.227	-0.794	0.475	-(
685	-1.720	-1.212	0.063	-0.930	0.031	0.227	0.678	-0.421	(

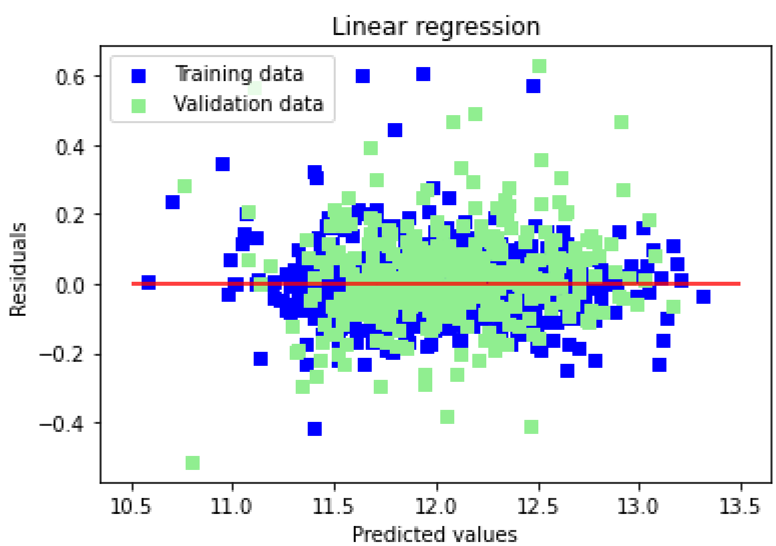
1019 rows × 323 columns

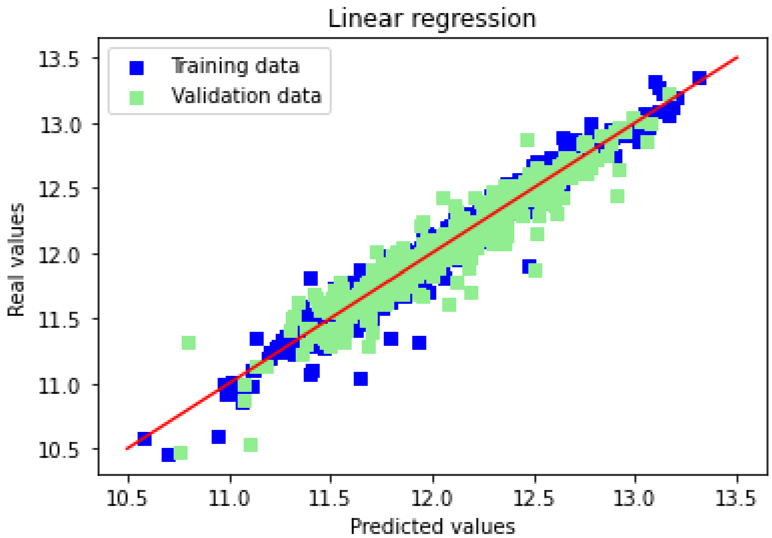
In [30]:
```python
# Define error measure for official scoring : RMSE
scorer = make_scorer(mean_squared_error, greater_is_better = False)

def rmse_cv_train(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y_train, scoring = scorer, cv
    return(rmse)

def rmse_cv_test(model):
    rmse= np.sqrt(-cross_val_score(model, X_test, y_test, scoring = scorer, cv =
    return(rmse)
```

1* Linear Regression without regularization

In [31]:
```
# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)

# Look at predictions on training and validation set
print("RMSE on Training set :", rmse_cv_train(lr).mean())
print("RMSE on Test set :", rmse_cv_test(lr).mean())
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)

# Plot residuals(a quantity remaining after other things have been subtracted or
plt.scatter(y_train_pred, y_train_pred - y_train, c = 'blue', marker = 's', label
plt.scatter(y_test_pred, y_test_pred - y_test, c = "lightgreen", marker = "s", la
plt.title('Linear regression')
plt.xlabel('Predicted values')
plt.ylabel('Residuals')
plt.legend(loc = 'upper left')
plt.hlines(y = 0, xmin = 10.5, xmax = 13.5, color = 'red')
plt.show()

# Plot predictions
plt.scatter(y_train_pred, y_train, c = "blue", marker = "s", label = "Training da
plt.scatter(y_test_pred, y_test, c = "lightgreen", marker = "s", label = "Validat
plt.title("Linear regression")
plt.xlabel("Predicted values")
plt.ylabel("Real values")
plt.legend(loc = "upper left")
plt.plot([10.5, 13.5], [10.5, 13.5], c = "red")
plt.show()
```

```
RMSE on Training set : 0.42107702693188453
RMSE on Test set : 0.4095155319280443
```





2* Linear Regression with Ridge regularization (L2 penalty)

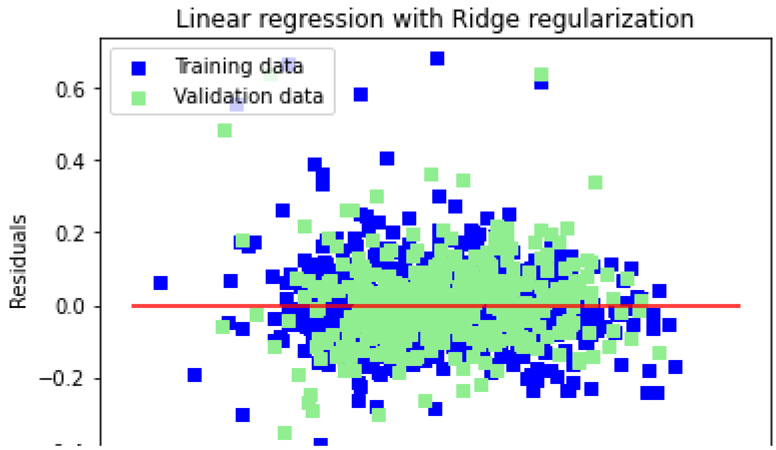
From the Python Machine Learning book by Sebastian Raschka : **Regularization** is a very useful method to handle **collinearity, filter out noise from data, and eventually prevent overfitting.** The concept behind regularization is to introduce **additional information (bias) to penalize extreme parameter weights.**

Ridge regression is an L2 penalized model where **we simply add the squared sum of the weights to our cost function.**

In [32]:

```
# 2* Ridge
ridge = RidgeCV(alphas = [0.01, 0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6, 10, 30, 60])
ridge.fit(X_train, y_train)
alpha = ridge.alpha_
print("Best alpha :", alpha)

print("Try again for more precision with alphas centered around " + str(alpha))
ridge = RidgeCV(alphas = [alpha * .6, alpha * .65, alpha * .7, alpha * .75, alpha
                          alpha * .9, alpha * .95, alpha, alpha * 1.05, alpha * 1
                          alpha * 1.25, alpha * 1.3, alpha * 1.35, alpha * 1.4],
                cv = 10)
ridge.fit(X_train, y_train)
alpha = ridge.alpha_
print("Best alpha :", alpha)

print("Ridge RMSE on Training set :", rmse_cv_train(ridge).mean())
print("Ridge RMSE on Test set :", rmse_cv_test(ridge).mean())
y_train_rdg = ridge.predict(X_train)
y_test_rdg = ridge.predict(X_test)

# Plot residuals
plt.scatter(y_train_rdg, y_train_rdg - y_train, c = "blue", marker = "s", label =
plt.scatter(y_test_rdg, y_test_rdg - y_test, c = "lightgreen", marker = "s", labe
plt.title("Linear regression with Ridge regularization")
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.legend(loc = "upper left")
plt.hlines(y = 0, xmin = 10.5, xmax = 13.5, color = "red")
plt.show()

# Plot predictions
plt.scatter(y_train_rdg, y_train, c = "blue", marker = "s", label = "Training dat
plt.scatter(y_test_rdg, y_test, c = "lightgreen", marker = "s", label = "Validati
plt.title("Linear regression with Ridge regularization")
plt.xlabel("Predicted values")
plt.ylabel("Real values")
plt.legend(loc = "upper left")
plt.plot([10.5, 13.5], [10.5, 13.5], c = "red")
plt.show()

# Plot important coefficients
coefs = pd.Series(ridge.coef_, index = X_train.columns)
print("Ridge picked " + str(sum(coefs != 0)) + " features and eliminated the othe
     str(sum(coefs == 0)) + " features")

imp_coefs = pd.concat([coefs.sort_values().head(10),
                      coefs.sort_values().tail(10)])
imp_coefs.plot(kind = 'barh')
plt.title('Coefficients in the Ridge Model')
plt.show()
```

```
Best alpha : 30.0
Try again for more precision with alphas centered around 30.0
Best alpha : 24.0
Ridge RMSE on Training set : 0.11527633660632723
Ridge RMSE on Test set : 0.1164325338035731
```



We're getting a much better RMSE result now that we've added regularization. **The very small difference between training and test results indicate that we eliminated most of the overfitting.** Visually, the graphs seem to confirm that idea.

Ridge used almost all of the existing features.

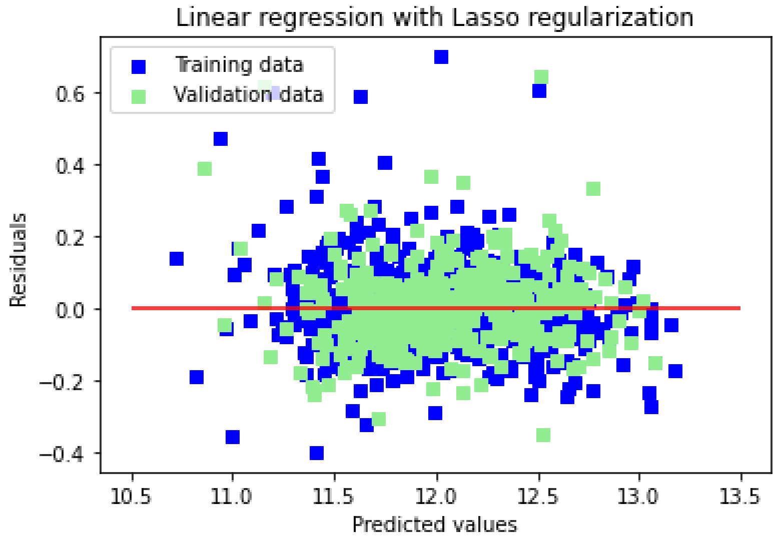
3* Linear Regression with Lasso regularization (L1 penalty)

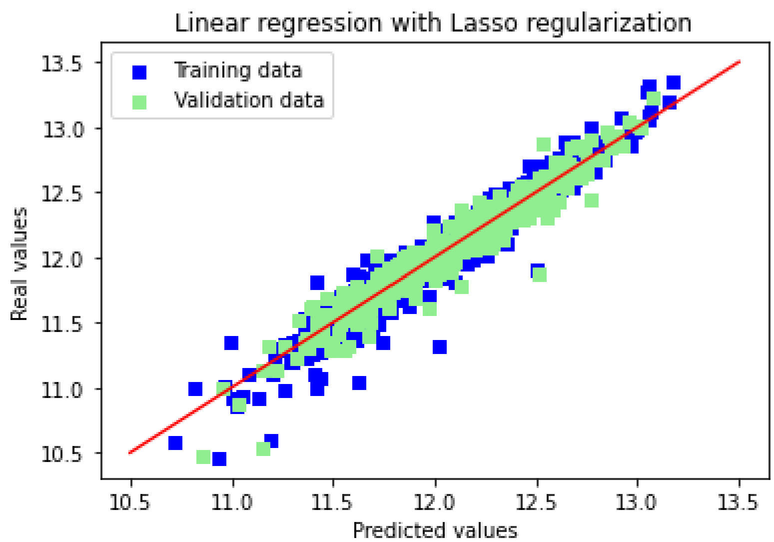
LASSO stands for **Least Absolute Shrinkage and Selection Operator.** It is an alternative regularization method, where we **simply replace the square of the weights by the sum of the absolute value of the weights**. In contrast to L2 regularization, L1 regularization yields sparse feature vectors : most feature weights will be zero. **Sparsity can be useful in practice if we have a high dimensional dataset with many features that are irrelevant.**

In [33]:
```
# 3* Lasso
lasso = LassoCV(alphas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006, 0.01, 0.03
                          0.3, 0.6, 1],
                max_iter = 50000, cv = 10)
lasso.fit(X_train, y_train)
alpha = lasso.alpha_
print("Best alpha :", alpha)

print("Try again for more precision with alphas centered around " + str(alpha))
lasso = LassoCV(alphas = [alpha * .6, alpha * .65, alpha * .7, alpha * .75, alpha
                          alpha * .85, alpha * .9, alpha * .95, alpha, alpha * 1.
                          alpha * 1.1, alpha * 1.15, alpha * 1.25, alpha * 1.3, a
                          alpha * 1.4],
                max_iter = 50000, cv = 10)
lasso.fit(X_train, y_train)
alpha = lasso.alpha_
print("Best alpha :", alpha)

print("Lasso RMSE on Training set :", rmse_cv_train(lasso).mean())
print("Lasso RMSE on Test set :", rmse_cv_test(lasso).mean())
y_train_las = lasso.predict(X_train)
y_test_las = lasso.predict(X_test)

# Plot residuals
plt.scatter(y_train_las, y_train_las - y_train, c = "blue", marker = "s", label =
plt.scatter(y_test_las, y_test_las - y_test, c = "lightgreen", marker = "s", labe
plt.title("Linear regression with Lasso regularization")
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.legend(loc = "upper left")
plt.hlines(y = 0, xmin = 10.5, xmax = 13.5, color = "red")
plt.show()

# Plot predictions
plt.scatter(y_train_las, y_train, c = "blue", marker = "s", label = "Training dat
plt.scatter(y_test_las, y_test, c = "lightgreen", marker = "s", label = "Validati
plt.title("Linear regression with Lasso regularization")
plt.xlabel("Predicted values")
plt.ylabel("Real values")
plt.legend(loc = "upper left")
plt.plot([10.5, 13.5], [10.5, 13.5], c = "red")
plt.show()

# Plot important coefficients
coefs = pd.Series(lasso.coef_, index = X_train.columns)
print("Lasso picked " + str(sum(coefs != 0)) + " features and eliminated the othe
      str(sum(coefs == 0)) + " features")
imp_coefs = pd.concat([coefs.sort_values().head(10),
                     coefs.sort_values().tail(10)])
imp_coefs.plot(kind = "barh")
plt.title("Coefficients in the Lasso Model")
plt.show()
```

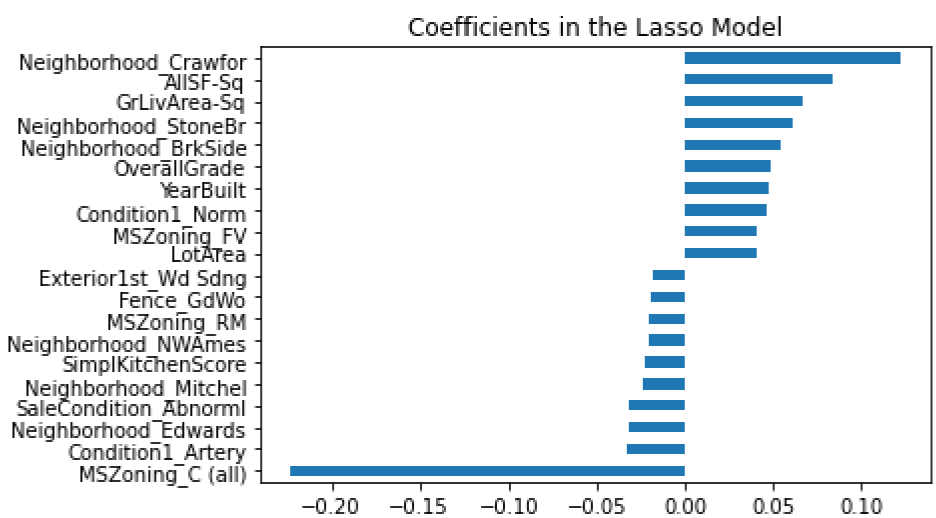
```
Best alpha : 0.0006
Try again for more precision with alphas centered around 0.0006
Best alpha : 0.0006
```

```
Lasso RMSE on Training set : 0.11360359018427942
Lasso RMSE on Test set : 0.11613054053923282
```






```
Lasso picked 110 features and eliminated the other 213 features
```



RMSE results are better both on training and test sets. The most interesting thing is that Lasso used only one third of the available features. Another interesting tidbit : it seems to give big weights to Neighborhood categories, both in positive and negative ways. Intuitively it makes sense, house prices change a whole lot from one neighborhood to another in the same city.

The **"MSZoning_C (all)"** feature seems to have a disproportionate impact compared to the others. It is defined as general zoning classification : commercial. It seems a bit weird to me that having your house in a mostly commercial zone would be such a terrible thing.

--

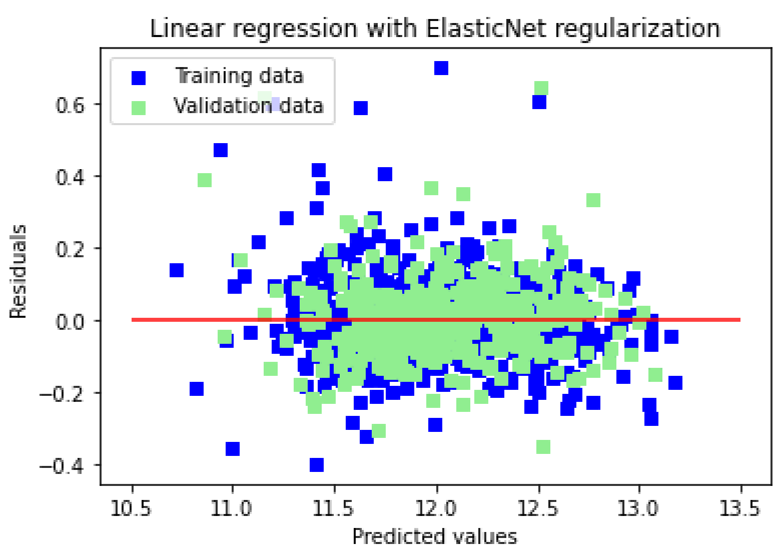
4* Linear Regression with ElasticNet regularization (L1 and L2 penalty)

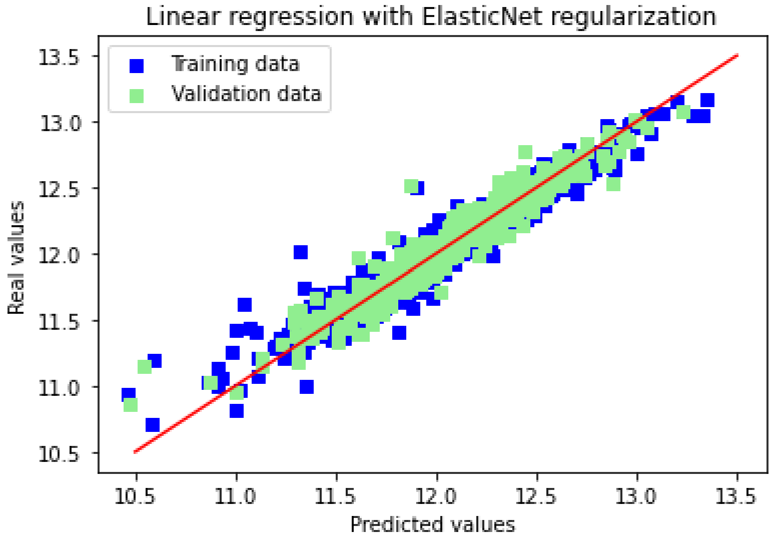
ElasticNet is a compromise between Ridge and Lasso regression. It has a L1 penalty to generate sparsity and a L2 penalty to overcome some of the limitations of Lasso, such as the number of variables (Lasso can't select more features than it has observations, but it's not the case here anyway).

In [34]:
```
# 4* ElasticNet
elasticNet = ElasticNetCV(l1_ratio = [0.1, 0.3, 0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.
                          alphas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006,
                                    0.01, 0.03, 0.06, 0.1, 0.3, 0.6, 1, 3, 6],
                          max_iter = 50000, cv = 10)
elasticNet.fit(X_train, y_train)
if elasticNet.l1_ratio_ > 1:
    elasticNet.l1_ratio_ = 1
alpha = elasticNet.alpha_
ratio = elasticNet.l1_ratio_
print("Best l1_ratio :", ratio)
print("Best alpha :", alpha )

print("Try again for more precision with l1_ratio centered around " + str(ratio))
# elasticNet = ElasticNetCV(l1_ratio = [ratio * .85, ratio * .9, ratio * .95, rat
#                           alphas = [0.0001, 0.0003, 0.0006, 0.001, 0.003, 0.006
#                           max_iter = 50000, cv = 10)
# elasticNet.fit(X_train, y_train)
# if elasticNet.l1_ratio_ > 1:
#     elasticNet.l1_ratio_ = 1
# alpha = elasticNet.alpha_
# ratio = elasticNet.l1_ratio_
# print("Best l1_ratio :", ratio)
# print("Best alpha :", alpha )

# print("Now try again for more precision on alpha, with l1_ratio fixed at " + st
#       " and alpha centered around " + str(alpha))
elasticNet = ElasticNetCV(l1_ratio = ratio,
                          alphas = [alpha * .6, alpha * .65, alpha * .7, alpha *
                                    alpha * .95, alpha, alpha * 1.05, alpha * 1.1
                                    alpha * 1.35, alpha * 1.4],
                          max_iter = 50000, cv = 10)
elasticNet.fit(X_train, y_train)
if (elasticNet.l1_ratio_ > 1):
    elasticNet.l1_ratio_ = 1
alpha = elasticNet.alpha_
ratio = elasticNet.l1_ratio_
print("Best l1_ratio :", ratio)
print("Best alpha :", alpha )

print("ElasticNet RMSE on Training set :", rmse_cv_train(elasticNet).mean())
print("ElasticNet RMSE on Test set :", rmse_cv_test(elasticNet).mean())
y_train_ela = elasticNet.predict(X_train)
y_test_ela = elasticNet.predict(X_test)

# Plot residuals
plt.scatter(y_train_ela, y_train_ela - y_train, c = "blue", marker = "s", label =
plt.scatter(y_test_ela, y_test_ela - y_test, c = "lightgreen", marker = "s", labe
plt.title("Linear regression with ElasticNet regularization")
plt.xlabel("Predicted values")
plt.ylabel("Residuals")
plt.legend(loc = "upper left")
plt.hlines(y = 0, xmin = 10.5, xmax = 13.5, color = "red")
plt.show()

# Plot predictions
```

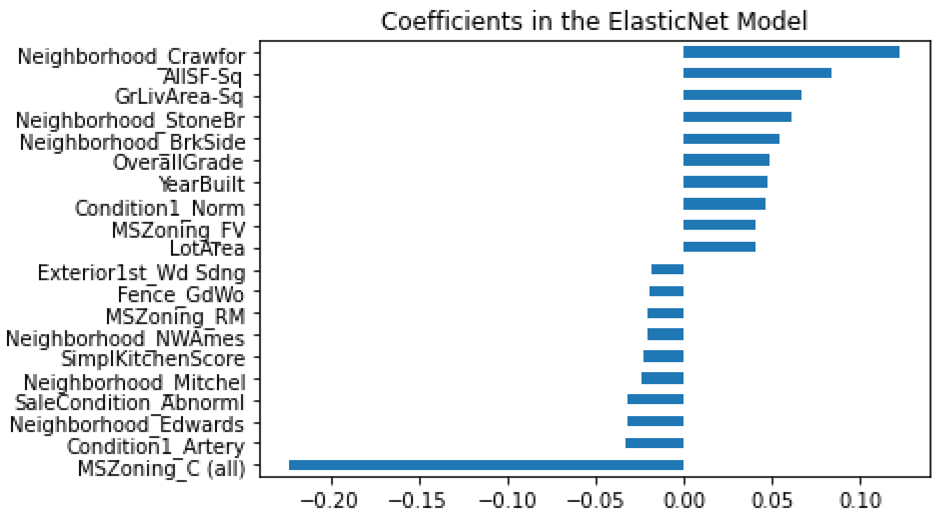
```
plt.scatter(y_train, y_train_ela, c = "blue", marker = "s", label = "Training dat
plt.scatter(y_test, y_test_ela, c = "lightgreen", marker = "s", label = "Validati
plt.title("Linear regression with ElasticNet regularization")
plt.xlabel("Predicted values")
plt.ylabel("Real values")
plt.legend(loc = "upper left")
plt.plot([10.5, 13.5], [10.5, 13.5], c = "red")
plt.show()

# Plot important coefficients
coefs = pd.Series(elasticNet.coef_, index = X_train.columns)
print("ElasticNet picked " + str(sum(coefs != 0)) + " features and eliminated the
imp_coefs = pd.concat([coefs.sort_values().head(10),
                     coefs.sort_values().tail(10)])
imp_coefs.plot(kind = "barh")
plt.title("Coefficients in the ElasticNet Model")
plt.show()
```

```
/home/hduser/.local/lib/python3.6/site-packages/sklearn/linear_model/_coordinat
e_descent.py:528: ConvergenceWarning: Objective did not converge. You might wan
t to increase the number of iterations. Duality gap: 0.05739253680864831, toler
ance: 0.01426910245430051
  tol, rng, random, positive)

Best l1_ratio : 1.0
Best alpha : 0.0006
Try again for more precision with l1_ratio centered around 1.0
Best l1_ratio : 1.0
Best alpha : 0.0006
ElasticNet RMSE on Training set : 0.11360359018427942
ElasticNet RMSE on Test set : 0.11613054053923282
```






```
ElasticNet picked 110 features and eliminated the other 213 features
```



The optimal L1 ratio used by ElasticNet here is equal to 1, which means it is exactly equal to the Lasso regressor we used earlier (and had it been equal to 0, it would have been exactly equal to our Ridge regressor). The model didn't need any L2 regularization to overcome any potential L1 shortcoming.

Note : I tried to remove the "MSZoning_C (all)" feature

In [ ]: