Ref.

[https://www.kaggle.com/apapiu/regularized-linear-models (https://www.kaggle.com/apapiu/regularized-linear-models)](https://www.kaggle.com/apapiu/regularized-linear-models)

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib

import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr

%config InlineBackend.figure_format = 'retina' #set 'png' here when working on no
%matplotlib inline
```

In [2]:
```
train = pd.read_csv("/home/hduser/jupyter/Comprehensive_data_exploration_with_Pyt
test = pd.read_csv("/home/hduser/jupyter/Comprehensive_data_exploration_with_Pyth
```

In [3]:
```
train
```

Out[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl
...	...	...	...	...	...	...	...	...	...
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	Lvl
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	Lvl
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	Lvl
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	Lvl
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	Lvl

1460 rows × 81 columns

In [4]:
```
#concate train and test by eliminatin id and target columns
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],
                      test.loc[:,'MSSubClass':'SaleCondition']))
```

In [5]:
```
all_data
```

Out[5]:

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilitie
0	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPu
1	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPu
2	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPu
3	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPu
4	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPu
...	...	...	...	...	...	...	...	...	.
1454	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	AllPu
1455	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	AllPu
1456	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	AllPu
1457	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	AllPu
1458	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	AllPu

2919 rows × 79 columns

Data preprocessing: --------------------------------------

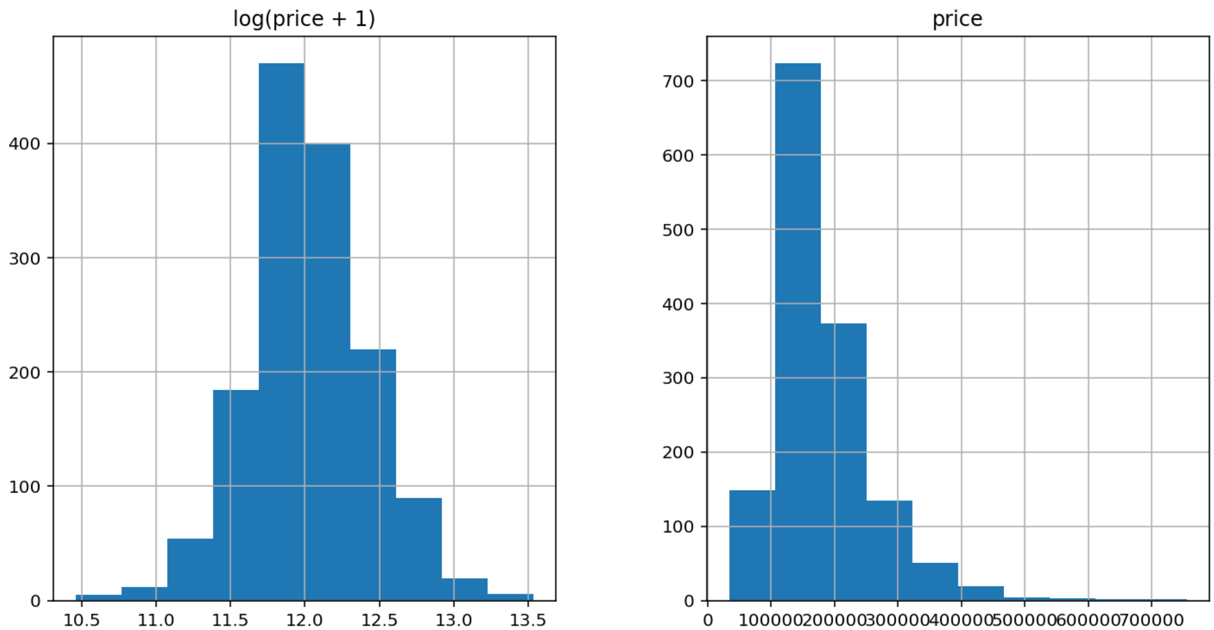
First I'll transform the skewed numeric features by taking log(feature + 1) - this will make the features more normal

Create Dummy variables for the categorical features

Replace the numeric missing values (NaN's) with the mean of their respective columns

In [6]:
```
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train['SalePrice'], 'log(price + 1)':np.log1p(trai
prices.hist()
```

Out[6]:
```
array([[<AxesSubplot:title={'center':'log(price + 1)'}>,
        <AxesSubplot:title={'center':'price'}>]], dtype=object)
```



In [7]:
```
#log transform the target:
train['SalePrice'] = np.log1p(train['SalePrice'])

#log transform skewed numeric features:
numeric_feats = all_data.dtypes[all_data.dtypes != 'object'].index
numeric_feats
```

Out[7]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold'],
      dtype='object')
```

In [8]:
```
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #compute sk
skewed_feats
```

Out[8]:
```
MSSubClass        1.406210
LotFrontage       2.160866
LotArea          12.195142
OverallQual       0.216721
OverallCond       0.692355
YearBuilt        -0.612831
YearRemodAdd     -0.503044
MasVnrArea        2.666326
BsmtFinSF1        1.683771
BsmtFinSF2        4.250888
BsmtUnfSF         0.919323
TotalBsmtSF       1.522688
1stFlrSF          1.375342
2ndFlrSF          0.812194
LowQualFinSF      9.002080
GrLivArea         1.365156
BsmtFullBath      0.595454
BsmtHalfBath      4.099186
FullBath          0.036524
HalfBath          0.675203
BedroomAbvGr      0.211572
KitchenAbvGr      4.483784
TotRmsAbvGrd      0.675646
Fireplaces        0.648898
GarageYrBlt      -0.648708
GarageCars       -0.342197
GarageArea        0.179796
WoodDeckSF        1.539792
OpenPorchSF       2.361912
EnclosedPorch     3.086696
3SsnPorch        10.293752
ScreenPorch       4.117977
PoolArea         14.813135
MiscVal          24.451640
MoSold            0.211835
YrSold            0.096170
dtype: float64
```

In [9]:
```
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats
```

Out[9]:
```
MSSubClass        1.406210
LotFrontage       2.160866
LotArea          12.195142
MasVnrArea        2.666326
BsmtFinSF1        1.683771
BsmtFinSF2        4.250888
BsmtUnfSF         0.919323
TotalBsmtSF       1.522688
1stFlrSF          1.375342
2ndFlrSF          0.812194
LowQualFinSF      9.002080
GrLivArea         1.365156
BsmtHalfBath      4.099186
KitchenAbvGr      4.483784
WoodDeckSF        1.539792
OpenPorchSF       2.361912
EnclosedPorch     3.086696
3SsnPorch        10.293752
ScreenPorch       4.117977
PoolArea         14.813135
MiscVal          24.451640
dtype: float64
```

In [10]:
```
skewed_feats = skewed_feats.index
skewed_feats
```

Out[10]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'MasVnrArea', 'BsmtFinSF1',
       'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtHalfBath', 'KitchenAbvGr',
       'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal'],
      dtype='object')
```

In [11]:
```
all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
```

In [12]:
```
all_data = pd.get_dummies(all_data)
```

In [13]:
```
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
```

In [14]:
```
#creating matrices for sklearn:
print(all_data.shape)
print(train.shape)

X_train = all_data[:train.shape[0]]
print(X_train.shape)
X_test = all_data[train.shape[0]:]
print(X_test.shape)
y = train['SalePrice']
```

```
(2919, 288)
(1460, 81)
(1460, 288)
(1459, 288)
```

Models

Now we are going to use regularized linear regression models from the scikit learn module. I'm going to try both l_1(Lasso) and l_2(Ridge) regularization. I'll also define a function that returns the cross-validation rmse error so we can evaluate our models and pick the best tuning par

In [15]:
```
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, LassoLarsCV
from sklearn.model_selection import cross_val_score

def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, X_train, y, scoring='neg_mean_squared_
    return(rmse)
```

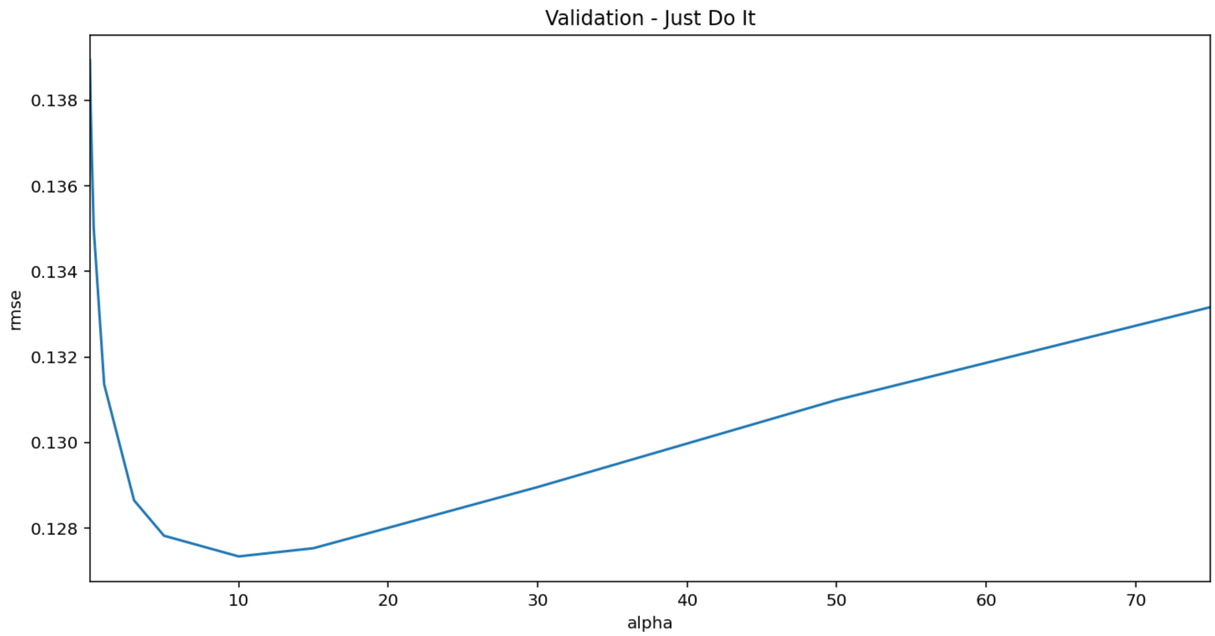
In [16]:
```
model_ridge = Ridge()
```

The main tuning parameter for the Ridge model is alpha - a regularization parameter that measures how flexible our model is. The higher the regularization the less prone our model will be to overfit. However it will also lose flexibility and might not capture all of the signal in the data.

In [17]:
```
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
cv_ridge = [rmse_cv(Ridge(alpha = alpha)).mean()
            for alpha in alphas]
```

In [18]:
```
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = 'Validation - Just Do It')
plt.xlabel('alpha')
plt.ylabel('rmse')
```

Out[18]: Text(0, 0.5, 'rmse')



Note the U-ish shaped curve above. When alpha is too large the regularization is too strong and the model cannot capture all the complexities in the data. If however we let the model be too flexible (alpha small) the model begins to overfit. **A value of alpha = 10 is about right based on the plot above.**

In [19]:
```
cv_ridge.min()
```

Out[19]: 0.12733734668670765

So for the Ridge regression we get a rmsle of about 0.00122

Let' try out the Lasso model. We will do a slightly different approach here and use the built in Lasso CV to figure out the best alpha for us. For some reason the alphas in Lasso CV are really the inverse or the alphas in Ridge.

In [20]:
```
model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train, y)
```

In [21]:
```
rmse_cv(model_lasso).mean()
```

Out[21]:
```
0.12256735885048145
```

Another neat thing about the Lasso is that it does feature selection for you - setting coefficients of features it deems unimportant to zero. Let's take a look at the coefficients:

In [22]:
```
coef = pd.Series(model_lasso.coef_, index=X_train.columns)
```

In [23]:
```
print('lasso picked '+str(sum(coef!= 0)) +  " variables and eliminated the other
```

```
lasso picked 110 variables and eliminated the other 178 variables
```

One idea to try here is run Lasso a few times on boostrapped samples and see how stable the feature selection is.

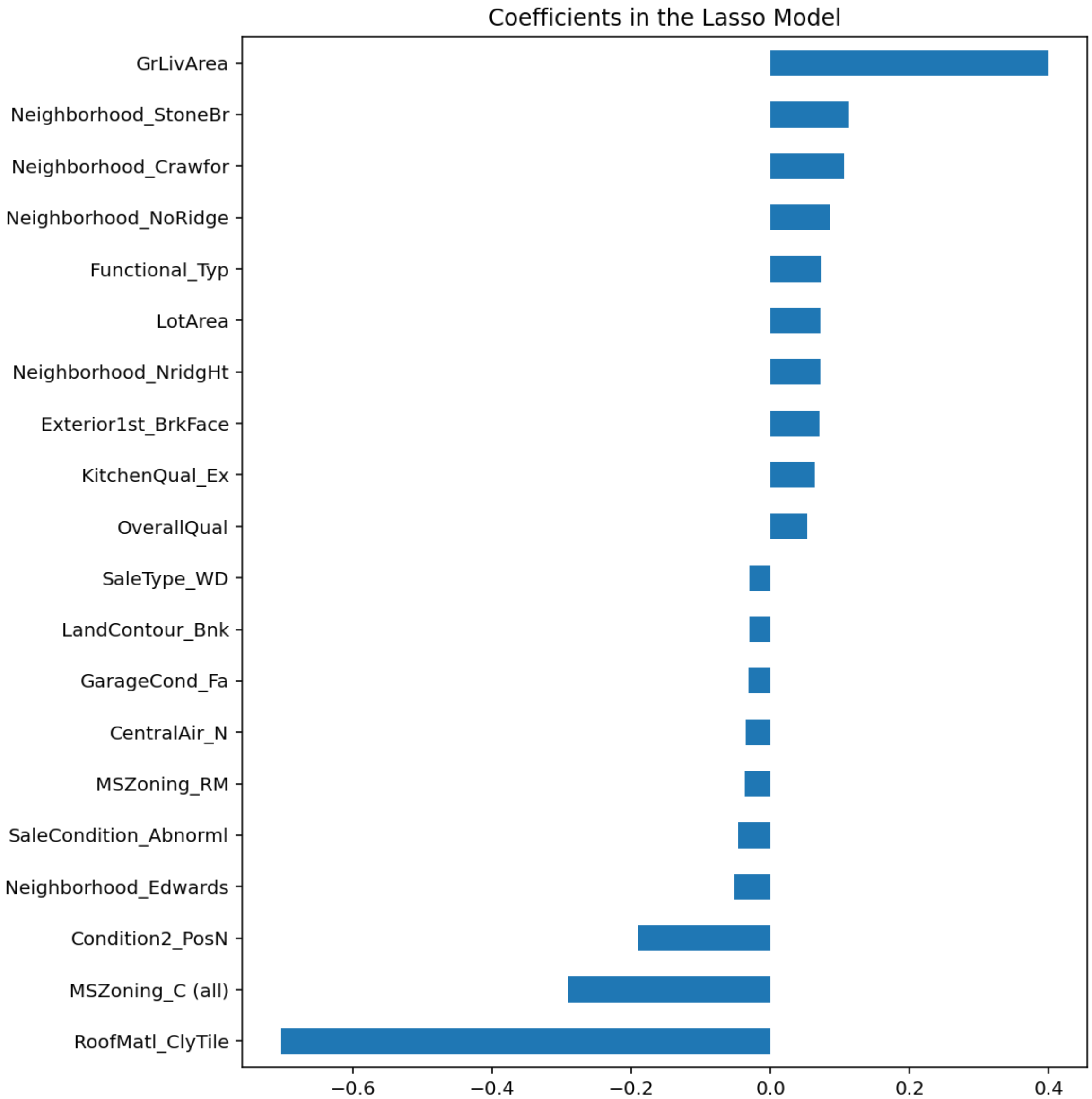
We can also take a look directly at what the most important coefficients are:

In [24]:
```
imp_coef = pd.concat([coef.sort_values().head(10),
                     coef.sort_values().tail(10)])
imp_coef
```

Out[24]:
```
RoofMatl_ClyTile        -0.704161
MSZoning_C (all)        -0.292023
Condition2_PosN         -0.190552
Neighborhood_Edwards    -0.052560
SaleCondition_Abnorml   -0.047116
MSZoning_RM             -0.037698
CentralAir_N            -0.035440
GarageCond_Fa           -0.031688
LandContour_Bnk         -0.030934
SaleType_WD             -0.030656
OverallQual              0.053160
KitchenQual_Ex           0.063709
Exterior1st_BrkFace      0.070464
Neighborhood_NridgHt     0.071620
LotArea                  0.071826
Functional_Typ           0.072597
Neighborhood_NoRidge     0.085717
Neighborhood_Crawfor     0.105138
Neighborhood_StoneBr     0.112493
GrLivArea                0.400009
dtype: float64
```

In [25]:
```
matplotlib.rcParams['figure.figsize'] = (8.0, 10.0)
imp_coef.plot(kind = "barh")
plt.title("Coefficients in the Lasso Model")
```

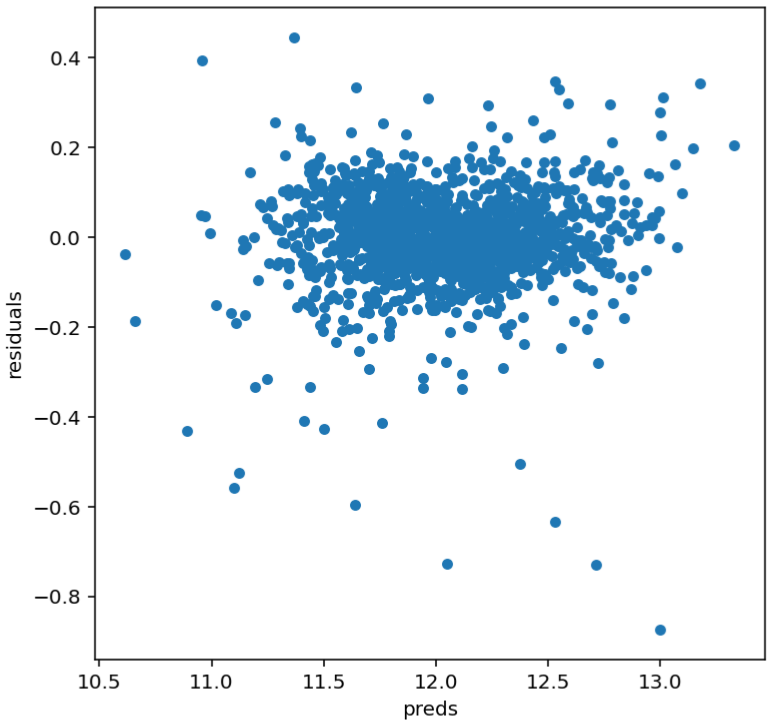
Out[25]:
```
Text(0.5, 1.0, 'Coefficients in the Lasso Model')
```



In [26]:
```
#let's look at the residuals as well:
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)

preds = pd.DataFrame({"preds":model_lasso.predict(X_train), "true":y})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals",kind = "scatter")
```

Out[26]:
```
<AxesSubplot:xlabel='preds', ylabel='residuals'>
```



The residual plot looks pretty good.To wrap it up let's predict on the test set