ASSIGNMENT 4

Abalone age prediction

In [1]:
```
%matplotlib inline

# ml
from sklearn import linear_model
from sklearn.model_selection import KFold

# numbers
import numpy as np
import pandas as pd

# stats
import statsmodels.api as sm
import scipy.stats as stats

# plots
import matplotlib.pyplot as plt
import seaborn as sns

# utils
import os, re
```

```
/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/site-packages/matplotlib/__
init__.py:872: UserWarning: axes.color_cycle is deprecated and replaced with axes.prop_cycl
e; please use the latter.
  warnings.warn(self.msg_depr % (key, alt_key))
```

In [2]:
```
# Copy-and-paste from prior notebook

def abalone_load(data_file, infant=False):
    # x data labels
    xnlabs = ['Sex']
    xqlabs = ['Length','Diameter','Height','Whole weight','Shucked weight','Viscera weight'
,'Shell weight']
    xlabs = xnlabs + xqlabs

    # y data labels
    ylabs = ['Rings']

    # data
    df = pd.read_csv(data_file, header=None, sep=' ', names=xlabs+ylabs)

    if(infant):
        new_df = df[ df['Sex']=='I' ]
    else:
        new_df = df[ df['Sex']<>'I' ]
    return new_df

def infant_abalone_load(data_file):
    return abalone_load(data_file,True)

def adult_abalone_load(data_file):
    return abalone_load(data_file,False)

def abalone_removeoutliers(df,mse_tol,verbose=False):
    df.loc[:,'Volume'] = df['Length'].values*df['Diameter'].values*df['Height'].values

    X = df['Volume']
    Y = df['Shell weight']
    lin = sm.OLS(Y,X).fit()

    Yhat = lin.predict(df['Volume'])
    df.loc[:,'Predicted shell weight'] = Yhat
    df.loc[:,'Residual'] = Y - Yhat
    df.loc[:,'MSE'] = (Y - Yhat)**2
    MSE = df['MSE']

    thresh = 0.5
    new_df = df[df['MSE'] < thresh]
    records_removed = len(df) - len(new_df)

    if(verbose):
        print "Number of data points removed: %d"%(records_removed)
        print "%0.2f%% of data was removed"%((float(records_removed)/len(df))*100)

    del df['Predicted shell weight']
    del df['Residual']
    del df['MSE']

    return df

def infant_abalone_removeoutliers(df):
    return abalone_removeoutliers(df,0.5)

def adult_abalone_removeoutliers(df):
    return abalone_removeoutliers(df,1.0)
```

In [3]:
```
inf_df = infant_abalone_removeoutliers(infant_abalone_load('abalone/Dataset.data'))
adt_df = adult_abalone_removeoutliers(adult_abalone_load('abalone/Dataset.data'))
```

In [4]:
```
inf_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1342 entries, 4 to 4166
Data columns (total 10 columns):
Sex               1342 non-null object
Length            1342 non-null float64
Diameter          1342 non-null float64
Height            1342 non-null float64
Whole weight      1342 non-null float64
Shucked weight    1342 non-null float64
Viscera weight    1342 non-null float64
Shell weight      1342 non-null float64
Rings             1342 non-null int64
Volume            1342 non-null float64
dtypes: float64(8), int64(1), object(1)
memory usage: 115.3+ KB
```

In [5]:
```
adt_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2835 entries, 0 to 4176
Data columns (total 10 columns):
Sex               2835 non-null object
Length            2835 non-null float64
Diameter          2835 non-null float64
Height            2835 non-null float64
Whole weight      2835 non-null float64
Shucked weight    2835 non-null float64
Viscera weight    2835 non-null float64
Shell weight      2835 non-null float64
Rings             2835 non-null int64
Volume            2835 non-null float64
dtypes: float64(8), int64(1), object(1)
memory usage: 243.6+ KB
```

scikit-learn: Linear Regression

In [6]:
```
# x data labels
xnlabs = ['Sex']
xqlabs = ['Length','Diameter','Height','Whole weight','Shucked weight','Viscera weight','Sh
ell weight']
xlabs = xnlabs + xqlabs

# y data labels
ylabs = ['Rings']

print xqlabs
```

```
['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell
weight']
```

```
['Rings']
```

In [8]:
```
lmreg = linear_model.LinearRegression( fit_intercept = False )
lmreg.fit( adt_df[xqlabs], adt_df[ylabs] )
```

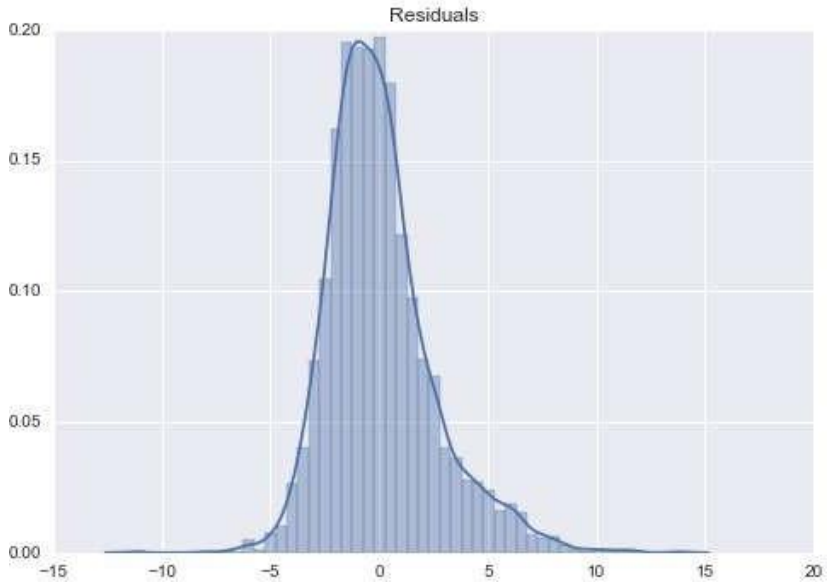
```
/usr/local/lib/python2.7/site-packages/scipy/linalg/basic.py:884: RuntimeWarning: internal
gelsd driver lwork query error, required iwork dimension not returned. This is likely the r
esult of LAPACK bug 0038, fixed in LAPACK 3.2.2 (released July 21, 2010). Falling back to
'gelss' driver.
  warnings.warn(mesg, RuntimeWarning)
```

Out[8]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
```

In [9]:
```
lmreg.coef_
```

Out[9]:
```
array([[  8.91404042,  12.22048227,  11.91342775,   8.72210978,
        -21.56383663, -12.44905461,   6.33154266]])
```

In [10]:
```
# Compute residuals
yhat = lmreg.predict(adt_df[xqlabs])
lmresid = adt_df[ylabs] - yhat
sns.distplot(lmresid)
plt.title('Residuals')

plt.show()
```



In [11]:
```
lmreg.score(adt_df[xqlabs],adt_df[ylabs])
```

In [12]:
```
mean = adt_df['Rings'].mean()
var = np.sqrt(adt_df['Rings'].var())
print "scikit-learn linear regression model:"
print "mean response value = %0.2f"%(mean)
print "L2 residual = %0.2f"%(var)
```

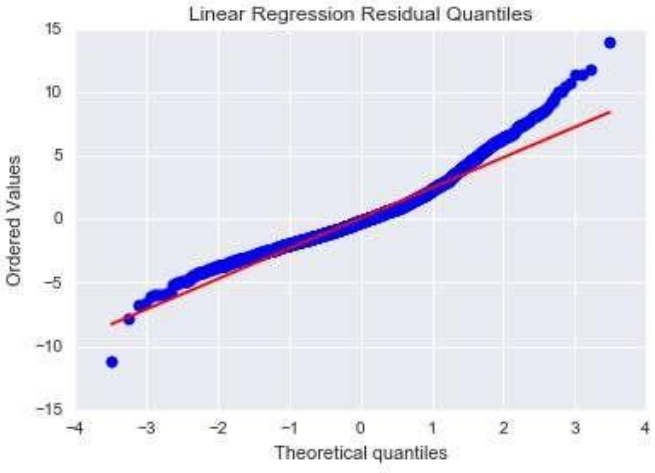
```
scikit-learn linear regression model:
mean response value = 10.90
L2 residual = 3.07
```

Most of the model residuals are within ± 5 of 0. However, this is larger than the standard deviation of the data.

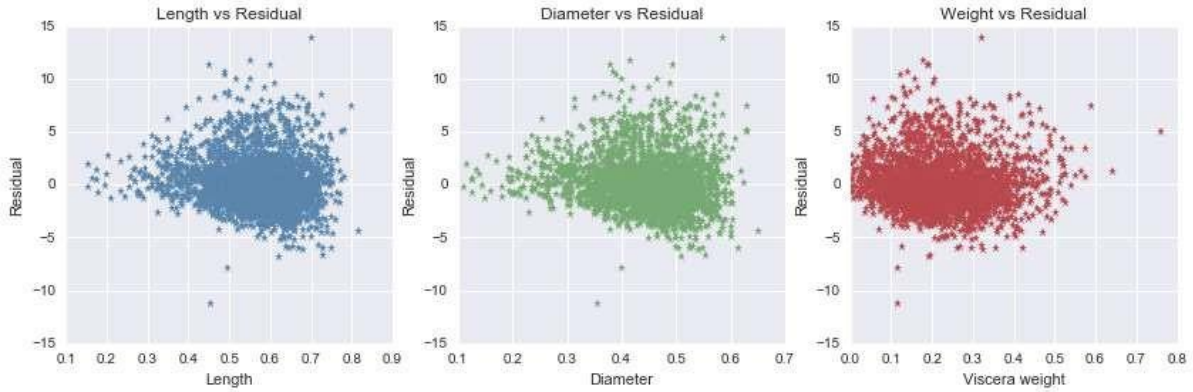
scikit-learn: Linear Regression Residuals

We can use a quantile plot of the residuals to figure out how they are distributed.

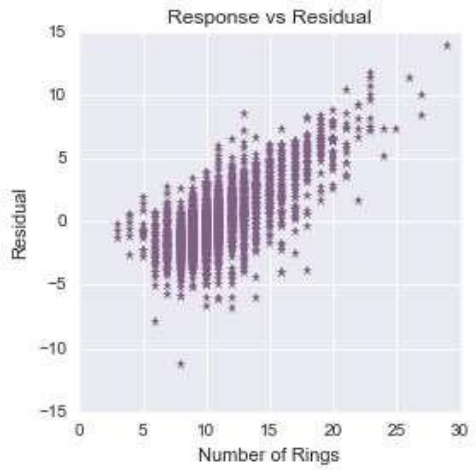
In [13]:
```
fig = plt.figure(figsize=(6,4))
ax1 = fig.add_subplot(111)

stats.probplot(lmresid['Rings'].values, dist='norm',plot=ax1)
ax1.set_title('Linear Regression Residual Quantiles')
plt.show()
```



In [14]:
```
fig, (ax1,ax2,ax3) = plt.subplots(1,3,figsize=(14,4))

ax1.plot(adt_df['Length'],lmresid['Rings'],'*', color=sns.xkcd_rgb['dusty blue'])
ax1.set_xlabel('Length')
ax1.set_ylabel('Residual')
ax1.set_title('Length vs Residual')

ax2.plot(adt_df['Diameter'],lmresid['Rings'],'*', color=sns.xkcd_rgb['dusty green'])
ax2.set_xlabel('Diameter')
ax2.set_ylabel('Residual')
ax2.set_title('Diameter vs Residual')

ax3.plot(adt_df['Viscera weight'],lmresid['Rings'],'*', color=sns.xkcd_rgb['dusty red'])
ax3.set_xlabel('Viscera weight')
ax3.set_ylabel('Residual')
ax3.set_title('Weight vs Residual')

plt.show()
```



In [15]:
```
fig, ax1 = plt.subplots(1,1,figsize=(4,4))

ax1.plot(adt_df[ylabs],lmresid['Rings'],'*', color=sns.xkcd_rgb['dusty purple'])
ax1.set_xlabel('Number of Rings')
ax1.set_ylabel('Residual')
ax1.set_title('Response vs Residual')

plt.show()
```



scikit-learn: Linear Regression Residuals with Transformed Variables

Linear models for the response in terms of the factors (input variables) provided with the data set do not do a great job of predicting the number of rings. To improve the model, we can transform the variables we have into new variables, and try to improve the fit between the model and the data by using these derived quantities.

In a previous notebook, the product of length, diameter, and height gave us a "volume" of each abalone, correlated closely with the abalone's shell and viscera weight. Let's see if we can improve the results of the linear model by using the volume variable in our regression, either in place of or together with length, diameter, and height. Note that this is equivalent to adding an interaction term $x_1 x_2 x_3$ for the input variables x_1, x_2, x_3 (length, diameter, height).

In [16]:
```
print adt_df.columns
```

```
Index([u'Sex', u'Length', u'Diameter', u'Height', u'Whole weight',
       u'Shucked weight', u'Viscera weight', u'Shell weight', u'Rings',
       u'Volume'],
      dtype='object')
```

In [17]:
```
newxqlabs = ['Length','Diameter','Height','Whole weight','Shucked weight','Viscera weight',
'Shell weight','Volume']
print xqlabs
print newxqlabs
```

```
['Length',             'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell
weight']
['Length', 'Diameter',
weight', 'Volume']
```

In [18]:
```
lmxreg = linear_model.LinearRegression( fit_intercept = False )
lmxreg.fit( adt_df[newxqlabs], adt_df[ylabs] )
```

Out[18]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
```

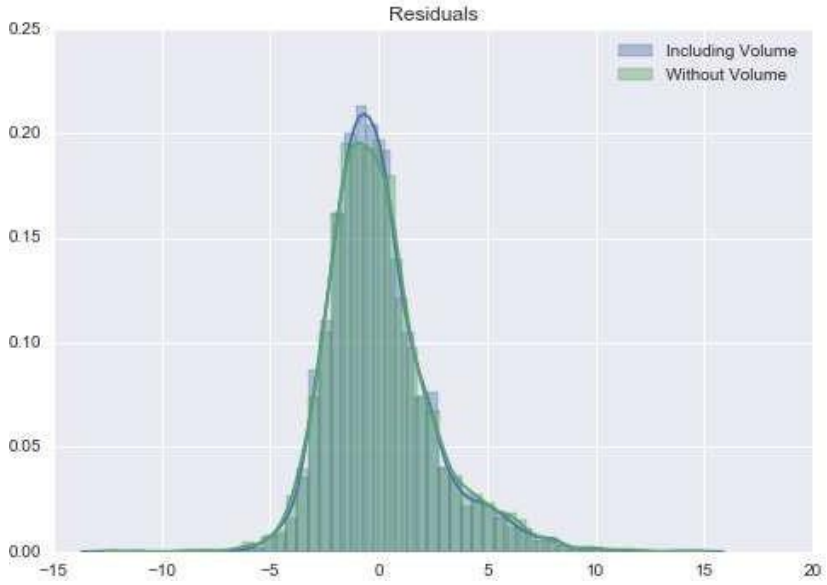
In [19]:
```
print lmreg.coef_
print lmxreg.coef_
```

In [20]:
```
print lmxreg.score(adt_df[newxqlabs],adt_df[ylabs])
```

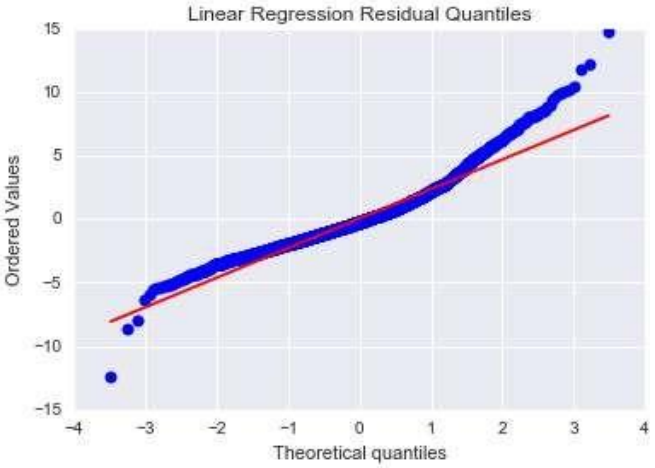
```
'Diameter',
0.391937121154         'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell
```

```
[[  8.91404042  12.22048227  11.91342775   8.72210978 -21.56383663
  -12.44905461   6.33154266]]
[[   1.73633061   16.28024187   30.08527083   10.05883223  -19.51392789
    -8.98888089   10.1848586  -109.33040707]]
```

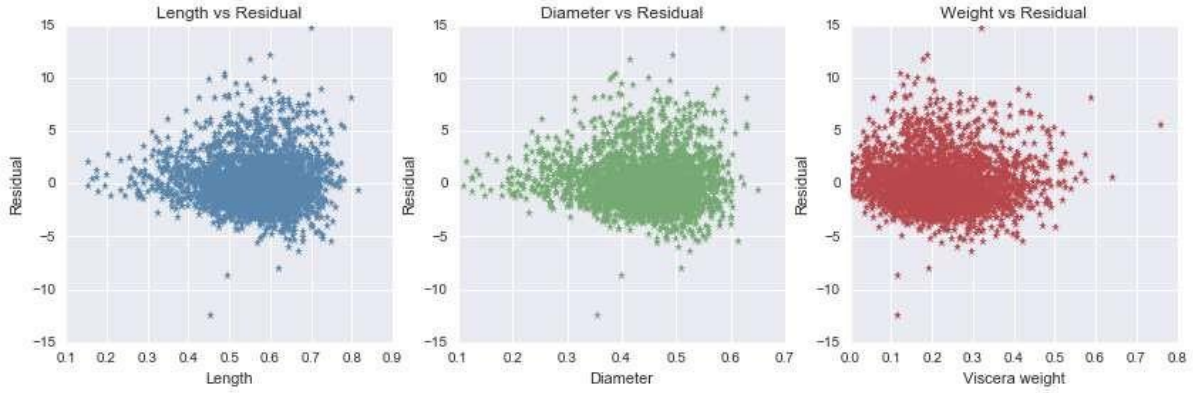
In [21]:
```
# Compute residuals
yhatx = lmxreg.predict(adt_df[newxqlabs])
lmxresid = adt_df[ylabs] - yhatx
sns.distplot(lmxresid, label='Including Volume')
sns.distplot(lmresid,  label='Without Volume')
plt.title('Residuals')
plt.legend()
plt.show()
```



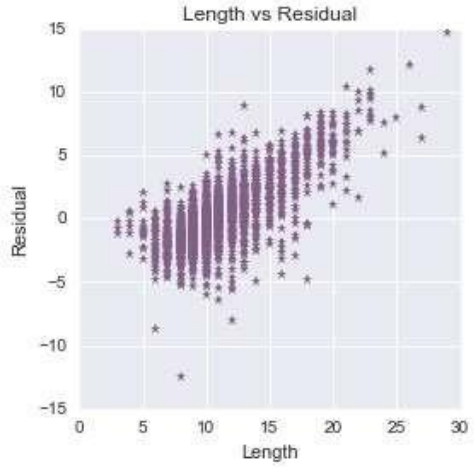
In [22]:
```
fig = plt.figure(figsize=(6,4))
ax1 = fig.add_subplot(111)

stats.probplot(lmxresid['Rings'].values, dist='norm',plot=ax1)
ax1.set_title('Linear Regression Residual Quantiles')
plt.show()
```



In [23]:
```
fig, (ax1,ax2,ax3) = plt.subplots(1,3,figsize=(14,4))

ax1.plot(adt_df['Length'],lmxresid['Rings'],'*', color=sns.xkcd_rgb['dusty blue'])
ax1.set_xlabel('Length')
ax1.set_ylabel('Residual')
ax1.set_title('Length vs Residual')

ax2.plot(adt_df['Diameter'],lmxresid['Rings'],'*', color=sns.xkcd_rgb['dusty green'])
ax2.set_xlabel('Diameter')
ax2.set_ylabel('Residual')
ax2.set_title('Diameter vs Residual')

ax3.plot(adt_df['Viscera weight'],lmxresid['Rings'],'*', color=sns.xkcd_rgb['dusty red'])
ax3.set_xlabel('Viscera weight')
ax3.set_ylabel('Residual')
ax3.set_title('Weight vs Residual')

plt.show()
```



In [24]:
```
fig, ax1 = plt.subplots(1,1,figsize=(4,4))

ax1.plot(adt_df[ylabs],lmxresid['Rings'],'*', color=sns.xkcd_rgb['dusty purple'])
ax1.set_xlabel('Length')
ax1.set_ylabel('Residual')
ax1.set_title('Length vs Residual')

plt.show()
```



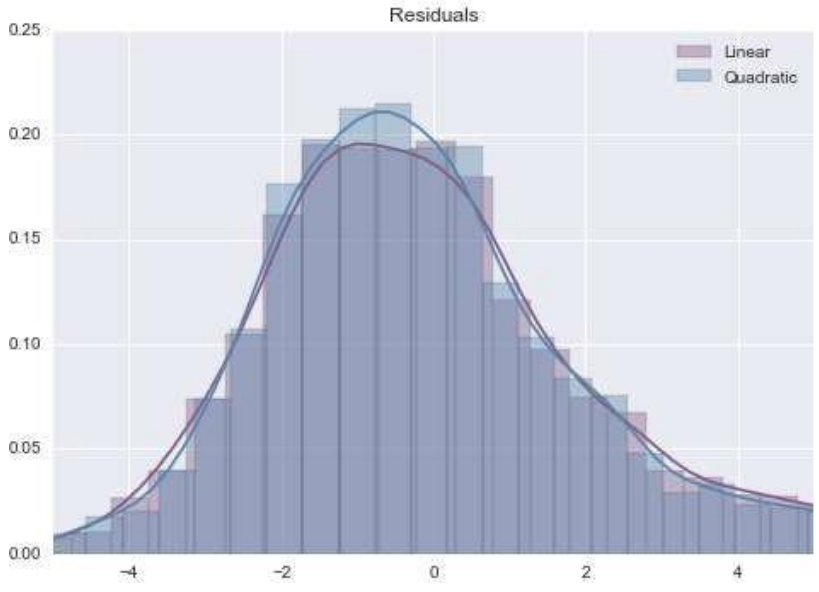
scikit-learn: Linear Regression Residuals with Quadratic Terms

In [25]:
```
sqf = lambda x : x*x

adt_df.loc[:,'LengthSquared']   = adt_df['Length'].apply( sqf )
adt_df.loc[:,'DiameterSquared'] = adt_df['Diameter'].apply( sqf )
adt_df.loc[:,'HeightSquared']   = adt_df['Height'].apply( sqf )
```

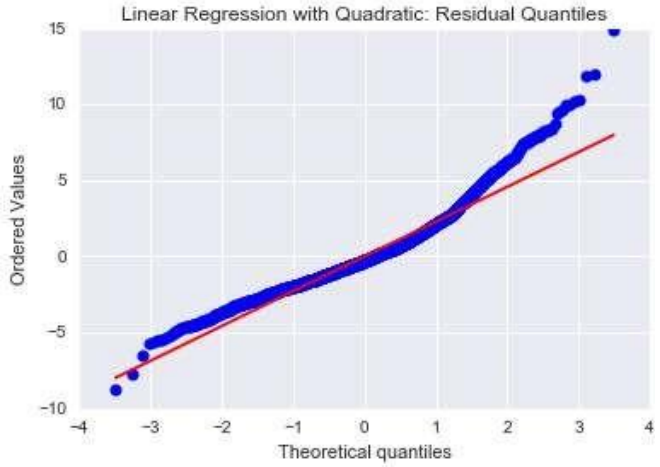
In [26]:
```
quadratic_xqlabs = xqlabs + ['LengthSquared','DiameterSquared','HeightSquared']
print quadratic_xqlabs

quadratic_reg = linear_model.LinearRegression( fit_intercept = False )
quadratic_reg.fit( adt_df[quadratic_xqlabs], adt_df[ylabs] )

print quadratic_reg.score( adt_df[quadratic_xqlabs], adt_df[ylabs] )
```

```
['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell
weight', 'LengthSquared', 'DiameterSquared', 'HeightSquared']
0.4100404631
```

In [27]:
```
quadratic_yhat = quadratic_reg.predict(adt_df[quadratic_xqlabs])
quadratic_resid = adt_df[ylabs] - quadratic_yhat

sns.distplot(lmresid,         label='Linear',    color = sns.xkcd_rgb['dusty purple'])
sns.distplot(quadratic_resid, label='Quadratic', color = sns.xkcd_rgb['dusty blue'])

plt.title('Residuals')
plt.xlim([-5,5])
plt.legend()
plt.show()
```



In [28]:
```
fig = plt.figure(figsize=(6,4))
ax1 = fig.add_subplot(111)

stats.probplot(quadratic_resid['Rings'].values, dist='norm',plot=ax1)
ax1.set_title('Linear Regression with Quadratic: Residual Quantiles')
plt.show()
```



scikit-learn: Ridge Regression

In [29]:
```
rreg = linear_model.Ridge( alpha=0.1, fit_intercept=False )
rreg.fit(adt_df[xqlabs],adt_df[ylabs])
```

Out[29]:
```
Ridge(alpha=0.1, copy_X=True, fit_intercept=False, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

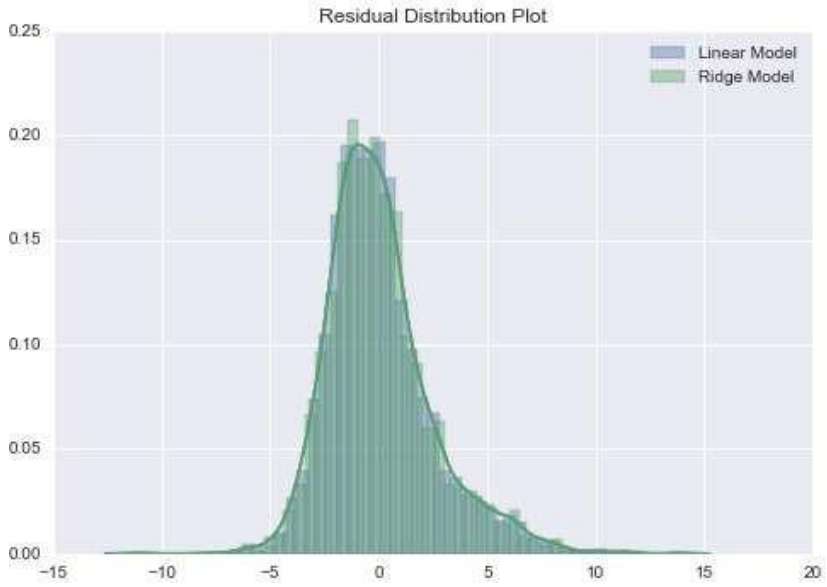
In [30]:
```
print lmreg.coef_
print rreg.coef_
```

```
[[             12.22048227  11.91342775   8.72210978 -21.56383663
  -12.44905461
[[  9.32922993
  -11.82394158
```

In [31]:
```
rresid = adt_df[ylabs] - rreg.predict(adt_df[xqlabs])
```

```
8.91404042
             6.33154266]]
            11.7675061   11.4932108    8.31168588 -21.15602306
             6.77249103]]
```

In [32]:
```
sns.distplot(lmresid, label='Linear Model')
sns.distplot(rresid, label='Ridge Model')
plt.title('Residual Distribution Plot')
plt.legend()
plt.show()
```



In [33]:
```
rreg.score(adt_df[xqlabs],adt_df[ylabs])
```

Out[33]:
```
0.36158698143414691
```

Differences in residuals between the linear regression and ridge regression models are essentially zero. Ridge regression gets us nowhere closer to predicting the number of rings.

In [34]:
```
mse = np.mean( (rreg.predict(adt_df[xqlabs]) - adt_df[ylabs])**2 )
ybar = np.mean(adt_df[ylabs])
print "L2 residuals: %0.4f"%( np.sqrt(mse) )
print "Mean Response Level: %0.4f"%(ybar)
```

```
L2 residuals: 2.4519
Mean Response Level: 10.9009
```

StatsModel: Linear Regression Model

We can use the StatsModel library in Python to perform an ordinary least squares linear regression. This will give us the same results as the scikit-learn least squares model.

In [35]:
```
import statsmodels.api as sm
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Rings   R-squared:                       0.953
Model:                            OLS   Adj. R-squared:                  0.953
Method:                 Least Squares   F-statistic:                     8215.
Date:                Sat, 07 Jan 2017   Prob (F-statistic):               0.00
Time:                        13:20:03   Log-Likelihood:                -6565.1
No. Observations:                2835   AIC:                         1.314e+04
Df Residuals:                    2828   BIC:                         1.319e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Length         8.9140      2.242      3.977      0.000         4.519    13.310
Diameter      12.2205      2.926      4.176      0.000         6.483    17.958
Height        11.9134      1.786      6.672      0.000         8.412    15.415
Whole weight   8.7221      0.875      9.973      0.000         7.007    10.437
Shucked weight -21.5638    0.981    -21.990      0.000       -23.487   -19.641
Viscera weight -12.4491    1.511     -8.241      0.000       -15.411    -9.487
Shell weight   6.3315      1.341      4.721      0.000         3.702     8.961
==============================================================================
Omnibus:                      443.772   Durbin-Watson:                   1.448
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              890.194
Skew:                           0.945   Prob(JB):                    4.98e-194
Kurtosis:                       4.991   Cond. No.                          117.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [37]:
```
print olm.params
print lmreg.coef_
```

```
Length            8.914040
Diameter         12.220482
Height           11.913428
Whole weight      8.722110
Shucked weight  -21.563837
Viscera weight  -12.449055
Shell weight      6.331543
dtype: float64
[[  8.91404042  12.22048227  11.91342775   8.72210978 -21.56383663
  -12.44905461   6.33154266]]
```

scikit-learn: Lasso with Cross-Validation

In [39]:
```
## Option 1:
## Specify an alpha
#lass = Lasso(random_state=0, alpha=0.5)
#lass = lass.fit( train_df[xqlabs], train_df[ylabs] )
#print lass.alpha

# Option 2:
# Use a Lasso model with cross-validation
# to optimize the value of alpha
lass = LassoCV(random_state=0)
lass = lass.fit( adt_df[xqlabs], adt_df[ylabs] )

print "LassoCV Model automatically selected the following value for alpha: %0.6f"%(lass.alpha_)
```

```
LassoCV Model automatically selected the following value for alpha: 0.002431
```

```
/usr/local/lib/python2.7/site-packages/sklearn/linear_model/coordinate_descent.py:1082: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [40]:
```
print lass.score( adt_df[xqlabs], adt_df[ylabs] )
```

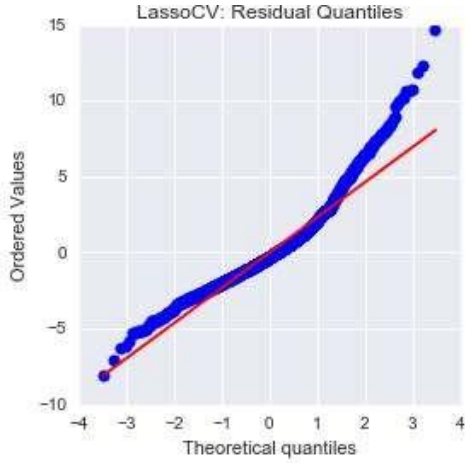
```
0.393742353568
```

In [41]:
```
print xqlabs
print lass.coef_
```

```
['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell weight']
[  0.          3.59585288  4.01971746   8.10163174 -18.41438066
  -6.18645066  10.79485582]
```

The score tells us that we get very little improvement using the Lasso model with cross validation, compared to ordinary least squares or ridge regression. However, one particular result that is interesting is that the Lasso model set the length input variable's coefficient to 0.

In [42]:
```
lassyhat = lass.predict( adt_df[xqlabs] )
lassy = adt_df['Rings']
lassresid = lassy - lassyhat
```



The Lasso regression didn't get us very far - but what if we add (independent) quadratic terms to the Lasso model?

In [44]:
```
lass2 = LassoCV(random_state=0)
lass2 = lass.fit( adt_df[quadratic_xqlabs], adt_df[ylabs] )
print "LassoCV Model automatically selected the following value for alpha: %0.6f"%(lass2.al
pha_)
print lass2.score( adt_df[quadratic_xqlabs], adt_df[ylabs] )
```

```
LassoCV Model automatically selected the following value for alpha: 0.000455
0.408913351524
```

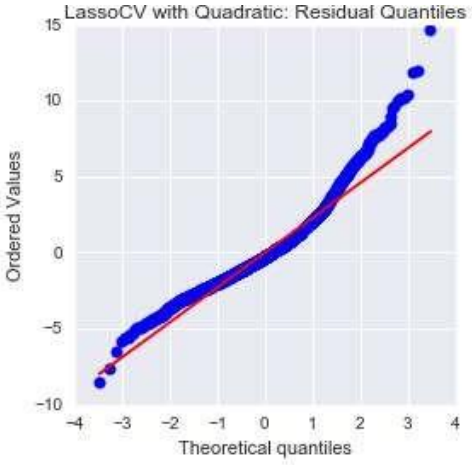
In [45]:
```
print quadratic_xqlabs
print lass2.coef_
```

```
['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight', 'Viscera weight', 'Shell
weight', 'LengthSquared', 'DiameterSquared', 'HeightSquared']
[ 14.18124344   5.60456286  13.58652532   9.69061828 -18.89303643
  -8.37335852   9.64138407 -19.07147246  -0.          -9.94176515]
```

If we add quadratic terms for length, diameter, and height, it doesn't get us very far: the score improves by only a point or so, and the coefficent for the length input variable (which was previously 0) is now 14. This is an indication that the model is only supriously fitting the data. This conclusion is also borne out by the fact that the quantile plots of each linear model have deviated substantially from the red line, representing the normal distribution.

The fact that the optimal alpha values, computed using cross-validation, are all very tiny.

In [46]:
```
lass2yhat = lass2.predict( adt_df[quadratic_xqlabs] )
lass2y = adt_df['Rings']
lass2resid = lass2y - lass2yhat
```



Adding a quadratic term shifted some of the points to be slightly closer to the line, but overall the quadratic terms do very little to improve model predictions.

scikit-learn: Cross Validation

In [48]:
```
lasso_cv = LassoCV(random_state=0)
k_fold = KFold(3)

for k, (train, test) in enumerate(k_fold.split(adt_df[xqlabs], adt_df[ylabs])):
    lasso_cv.fit( adt_df[xqlabs].iloc[train], adt_df[ylabs].iloc[train] )

    print("[fold {0}] alpha: {1:.5f}, score: {2:.5f}".
          format(k+1, lasso_cv.alpha_, lasso_cv.score( adt_df[xqlabs].iloc[test], adt_df[yl
abs].iloc[test] )) )
```

```
[fold 1] alpha: 0.00333, score: 0.38488
[fold 2] alpha: 0.00046, score: 0.35154
[fold 3] alpha: 0.00124, score: 0.26222
```

Notes:

- I have very little idea about how to interpret these numbers - what does it mean if they're all about the same? What does it mean if they all increase, or decrease? What does it mean if one is a huge outlier?

scikit-learn: Lasso Lars Regression

```
  13.87617852   8.14750654   2.06446954   2.8091537  -16.13423428   0.
  10.92616795]
[[  8.91404042
  -12.44905461
```

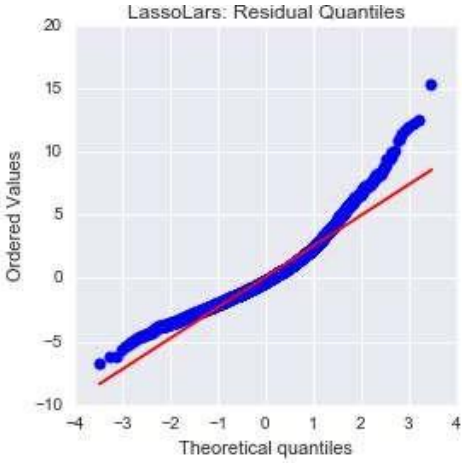
In [50]:
```
print lars.score( adt_df[xqlabs], adt_df[ylabs] )
print lmreg.score( adt_df[xqlabs], adt_df[ylabs] )
```

In [49]:
```
lars = linear_model.LassoLars( alpha = 0.01, fit_intercept = False )
lars = lars.fit( adt_df[xqlabs], adt_df[ylabs] )
print lars.coef_
print lmreg.coef_
```

```
0.333196729611
[
0.361672124059
```

```
                  12.22048227  11.91342775   8.72210978 -21.56383663
  6.33154266]]
```

In [51]:
```
larsyhat = lars.predict( adt_df[xqlabs] )
larsy = adt_df['Rings']
larsresid = larsy - larsyhat
```



Like with other linear models tried so far, the residuals computed from the LassoLars regression have moved a bit with respect to the red line (representing a normal distribution) but has not changed the fact that the residuals are not normally distributed. As before, we can interpret this plot to mean that no linear model will be sufficient to make accurate predictions with this data.

Scaling

Next we'll see what happens if we scale the input variables - and whether that leads to any improvement in how well the linear models can match our data.

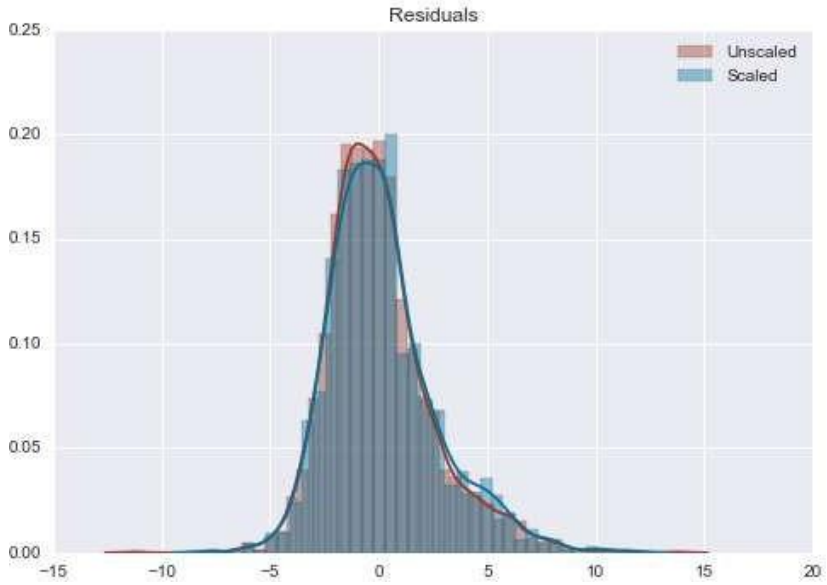
In [53]:
```
from sklearn import preprocessing
from sklearn import datasets, svm
from sklearn.model_selection import train_test_split
```

In [54]:
```
X_train, X_test, y_train, y_test = train_test_split( adt_df[xqlabs], adt_df[ylabs], test_si
ze=0.4, random_state=1)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_transformed = scaler.transform(X_train)
```

In [55]:
```
lmregn = linear_model.LinearRegression( fit_intercept = False )
lmregn.fit( X_train[xqlabs], y_train[ylabs] )
print lmregn.score(X_test,y_test)
```

```
0.358323405235
```

```
/usr/local/lib/python2.7/site-packages/numpy/lib/function_base.py:564: VisibleDeprecationWa
rning: using a non-integer number instead of an integer will result in an error in the futu
re
  n = np.zeros(bins, ntype)
/usr/local/lib/python2.7/site-packages/numpy/lib/function_base.py:611: VisibleDeprecationWa
rning: using a non-integer number instead of an integer will result in an error in the futu
re
  n += np.bincount(indices, weights=tmp_w, minlength=bins).astype(ntype)
```



Apparently, scaling the input variables leads to slightly *worse* fits! The variance in the residuals increases slightly. As with all of the other attempts we've made to improve the fit of linear models, this step doesn't get us very far.

scikit-learn: Linear Support Vector Regression

Support vector machines are models for classification of data; typically the responses are either binary or are integers. Support vector regression applies the same ideas and concepts to floating point responses.

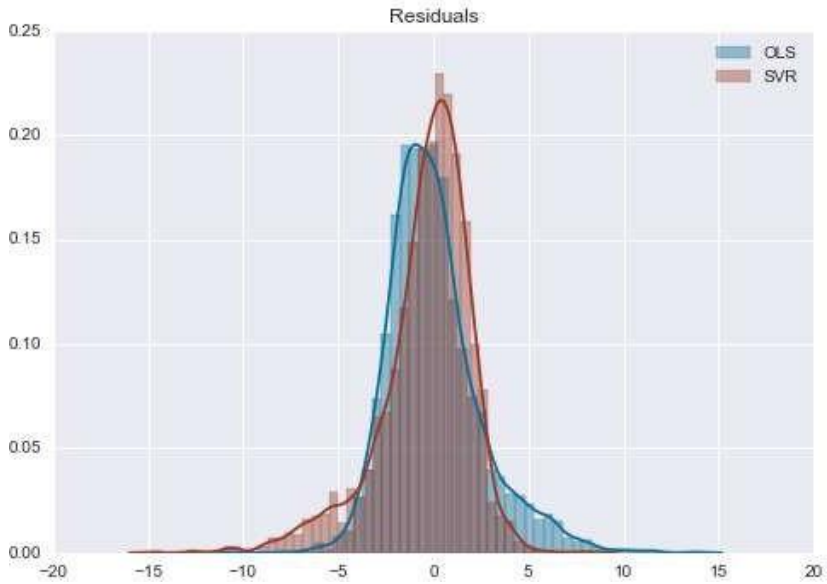
In [57]:
```
from sklearn.svm import SVR
```

In [58]:
```
svr_lin = SVR(kernel='linear', C=1e4)
svr_lin = svr_lin.fit( adt_df[xqlabs], adt_df['Rings'].values )
```

In [59]:
```
print svr_lin.score(adt_df[xqlabs],adt_df['Rings'].values)
```

```
0.373155977736
```

In [60]:
```
yhat = adt_df['Rings'].values
yhat_lin = svr_lin.predict(adt_df[xqlabs])

svresid = yhat_lin - yhat

sns.distplot(lmresid, color=sns.xkcd_rgb['ocean blue'], label='OLS')
sns.distplot(svresid,  color=sns.xkcd_rgb['brick'], label='SVR')

plt.title('Residuals')
plt.legend()
plt.show()
```



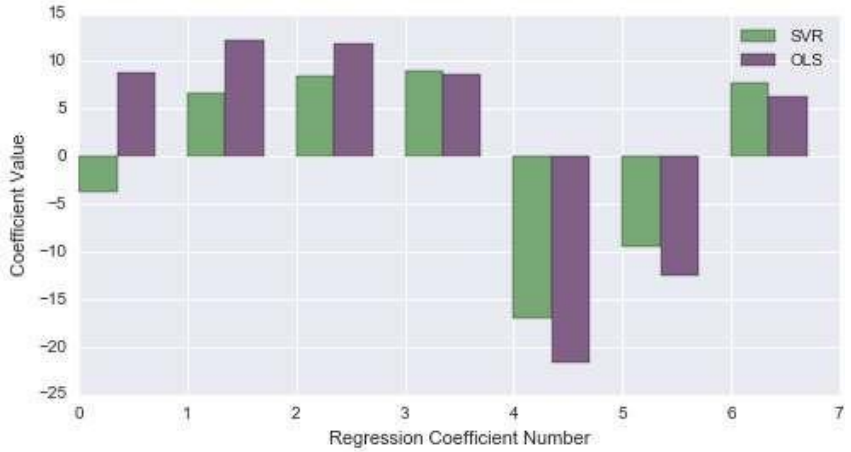
An interesting result!

The state vector regression technique has eliminated some bias in the residuals, as evidenced by the peak of the residuals histogram being more closely centered at 0. However, a large group of negative outliers has cropped up as a result. If we look at the coefficients, we can see they are slightly different:

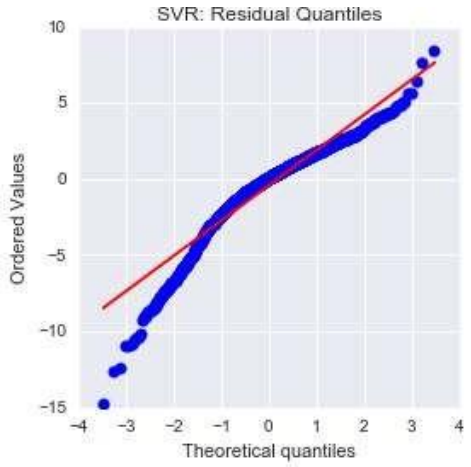
In [61]:
```
print svr_lin.coef_
print lmreg.coef_
```

```
[[ -3.69284789   6.68091811   8.50800478   9.00432672 -17.04983165
   -9.53431576   7.74145739]]
[[  8.91404042  12.22048227  11.91342775   8.72210978 -21.56383663
  -12.44905461   6.33154266]]
```

In [62]:
```
w = 0.35
fig = plt.figure(figsize=(8,4))
ax = fig.add_subplot(111)

ax.bar(np.arange(len(svr_lin.coef_[0])), svr_lin.coef_[0], w, color=sns.xkcd_rgb['dusty gre
en'],  label='SVR')
ax.bar(np.arange(len(lmreg.coef_[0]))+w,   lmreg.coef_[0], w, color=sns.xkcd_rgb['dusty pur
ple'], label='OLS')
ax.legend(loc='best')
ax.set_xlabel('Regression Coefficient Number')
ax.set_ylabel('Coefficient Value')
plt.show()
```



In [63]:
```
fig = plt.figure(figsize=(4,4))
ax1 = fig.add_subplot(111)

stats.probplot(svresid, dist='norm', plot=ax1)

ax1.set_title('SVR: Residual Quantiles')

plt.show()
```



The conclusion from this is the same as before: while SVR got us just a bit closer to a better model, it still couldn't get the job done - because it's still only a linear model, and still cannot capture higher order effects and variable interactions.