Stat 222 S'14 Capstone Project Analysis of Pairs Trading Stock Selection Methodologies

Jeffrey Darling and Willy Lai

May 5, 2014

This notebook, data files, and plots can be accessed on GitHub at:
https://github.com/jeffreydarling/222project

Introduction and Brief Overview of Pairs Trading

This project is motivated by the prevalence of websites hawking "safe" trading strategies, usually based on some form of statistical arbitrage. Pairs trading is one such method, where an investor tracks two stocks that tend to move in the same fashion. When one stock outperforms the other, the higher stock is sold short, and the other is bought, betting on a reversion to the mean. When the stock prices (hopefully) reconverge, the trades are undone, resulting in profit. Anectodally, it is said that there should be some known relationship between the stocks that helps to ensure consistent performance over time.

Websites such as Investopedia have pages devoted to describing such strategies, generally in an overly optimistic fashion. See this page on Investopedia, for example.

However, these strategies, particularly when applied with a fairly basic method such as correlation, can be very risky. This article, while perhaps a bit dramatic, does a good job of describing a couple of scenarios and reasons why it can be dangerous.

Our goal is to evaluate three of the most commonly recommended pairs trading stock selection strategies to determine how reliable these methods actually are. These methods are correlation, distance (sum of square difference), and cointegration. Further, we wish to evaluate the anecdotal claim that stocks should be related in some known fashion, and should not simply be selected by statistical means.

Note to users:

- This notebook was generated using the most recent version of Anaconda and iPython as of March 2014.
- The pairwise tests in this notebook (correlation, distance, and cointegration) can take many hours to run.
- We have saved the output of the tests as .pkl and .gz files in the github repository linked above. We suggest using this already-generated output to recreate plots.
- If you need to rerun the tests, they are currently configured for parallel processing across four cores; if your hardware does not support this, you can change the Pool(processors=4) code to reflect a different number of processors.
- "Run All" will not work unless you have already downloaded the .pkl data objects from the repository above, or if you follow the instructions in the WRDS Data section* to obtain the raw data download.

Import required libraries

In [1]:
```
import pandas as pd
import scipy.stats as stats
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as stat
import statsmodels.tsa.stattools as ts
from itertools import combinations
import multiprocessing
from multiprocessing import Pool
import cPickle as pkl
%pylab inline
pylab.rcParams['figure.figsize'] = (20.0, 16.0)
pylab.rcParams['xtick.major.size'] = 14
pylab.rcParams['xtick.labelsize'] = 12
pylab.rcParams['font.size'] = 14
```

```
Populating the interactive namespace from numpy and matplotlib
```

Download and Extract Wharton (WRDS) Data

```
# The initial data source for this project is Wharton Research Data
Services.
# This is a subscription-based service and may not be publicly
available to all users.
# We have included just the data used for this project in the github
repository linked above.
# After downloading that data, you can use this code to generate the
data objects used for further analysis.
# This process takes a few minutes, so we have saved the output in
.pkl format for much faster data loading.
# If you want to start from the raw data, change this cell from raw
text to code in order to run.

# Unzip and import sorted stock data - currently using test file to
manage load times
!gunzip -c wrdsDownload.csv.gz | head -n1 > wrdsSorted.csv
!gunzip -c wrdsDownload.csv.gz | tail -n +2 | sort -t, -k2,2 >>
wrdsSorted.csv

# Use absolute values to clean up incorrect negative values
wrdsData = pd.io.parsers.read_csv("wrdsSorted.csv", sep = ",",
index_col=[0,1], parse_dates=0, usecols = [1,2,3]).abs()

# Known issue with usecols is wrong index names
wrdsData.index.names = ['date', 'ticker']
wrdsData.head()

# Sort and remove duplicate indices
wrdsDirty =
wrdsData.reset_index().groupby(wrdsData.index.names).first()
wrdsClean = wrdsDirty.reset_index().pivot('date','ticker','PRC')

#Create training and testing data sets
```

```
trainData = wrdsClean.ix['20110103':'20121231'].dropna(axis=1,
how='all')
testData = wrdsClean.ix['20130102':'20131231']

# This cell takes the training dataset output from the previous cell
and saves it in .pkl format for near-instant loading.
# This should only be run if you ran the previous cell; if you are
using the downloaded .pkl objects, skip this step.
# To run this cell, change this cell from raw text to code.

trainOutput = open('trainData.pkl', 'wb')
pkl.dump(trainData, trainOutput, -1)
trainOutput.close()

# This cell takes the test dataset output from the above cell and
saves it in .pkl format for near-instant loading.
# This should only be run if you ran the previous cell; if you are
using the downloaded .pkl objects, skip this step.
# To run this cell, change this cell from raw text to code.

testOutput = open('testData.pkl', 'wb')
pkl.dump(testData, testOutput, -1)
testOutput.close()
```

In [2]:
```
# This cell loads the .pkl training dataset object, either created above
or downloaded from the github repository.

trainInput = open('trainData.pkl', 'rb')
trainData = pkl.load(trainInput)
trainInput.close()
```

In [3]:
```
# This cell loads the .pkl test dataset object, either created above
or downloaded from the github repository.

testInput = open('testData.pkl', 'rb')
testData = pkl.load(testInput)
testInput.close()
```

Determine the total number of pairs of stocks to analyze

In [4]:
```
# Extracts column headers from training data set and creates all possible pairs

tickerlist = sorted(trainData.columns.tolist())[1:]
tickerpairs = list(combinations(tickerlist,2))
print tickerpairs[:10] # Look at first 10 combinations of pairs
print len(tickerpairs) # Over 3 million possible pairs
```

```
[('AAME', 'AAON'), ('AAME', 'AAPL'), ('AAME', 'AAWW'), ('AAME',
'AAXJ'), ('AAME', 'ABAX'), ('AAME', 'ABCB'), ('AAME', 'ABCD'),
('AAME', 'ABCO'), ('AAME', 'ABFS'), ('AAME', 'ABIO')]
3232153
```

Training Data Set

Distance Method

```
# This cell calculates the mean SSD for each pair of stocks.  Due to
the number of pairs, this can take many hours.
# We have saved the output as a .pkl file that is loaded in future
steps, so we recommend skipping this and the next two cells.
# If you want to run the full analysis, change this cell from raw text
to code.

def dist(tlist):
    xname = tlist[0]
    yname = tlist[1]
    x = trainData[xname]/(trainData[xname][0])
    y = trainData[yname]/(trainData[yname][0])
    # Remove any na values
    z = (x-y).dropna()
    # Only consider pairs with most of the data present
    if len(z) > 495:
        return([(xname, yname)], sum(map(lambda z:z**2, z)))
    else:
        return()

if __name__ == '__main__':
    trainDistPool = Pool(processes=4)
    # Test just the first 100 pairs - remove [0:100] for full test -
warning, takes a long time!!
    trainDistResult = pd.DataFrame(trainDistPool.map(dist,
tickerpairs[0:100]))

trainDistPool.close()

# This cell takes the mean SSD output from the previous cell and saves
it in .pkl format for near-instant loading.
# This should only be run if you ran the previous cell; if you are
using the downloaded .pkl objects, skip this step.
# To run this cell, change this cell from raw text to code.

trainDistOutput = open('trainDistResult.pkl', 'wb')
pkl.dump(trainDistResult, trainDistOutput, -1)
trainDistOutput.close()

# This cell creates a .gz archive of the .pkl data so that it is small
enough for storage in the github repository.
# This should only be run if you intend to upload the data and need a
compressed format.
# To run this cell, change this cell from raw text to code.

!gzip -k trainDistResult.pkl
```

In [5]:
```
# This loads the training dataset mean SSD results from the .gz file available
on github.
# This should be run to access already-generated results and will just take
a short while.
# If you are creating the data yourself, please skip this step.

!gunzip -c trainDistResult.pkl.gz > trainDistResult.pkl
trainDistInput = open('trainDistResult.pkl', 'rb')
trainDistResult = pkl.load(trainDistInput)
trainDistInput.close()
```

In [6]:
```
# This cell sorts the mean SSD results and selects just the 100 "closest"
pairs of stocks.

smallssd = trainDistResult.sort(columns = 1)[0:100]
print smallssd.head()
```

```
                          0         1
3221878    [(VONE, VTHR)]  0.013202
2958608    [(PRFZ, VTWV)]  0.078365
3217295    [(VCSH, VMBS)]  0.112965
2459008  [(LBTYA, LBTYK)]  0.125308
3225367    [(VTWG, VTWO)]  0.129157

[5 rows x 2 columns]
```

Distance Method Training Plots

In [7]:
```
# This cell plots the standardized stock prices for the five 'closest'
pairs of stocks using the mean SSD method.
# The plot covers the entire training dataset time range, from January 1, 2011,
to December 31, 2012.
# This will plot inline, and the following cell is a figure caption.

plt.figure(1)
i = 1
for a in smallssd.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = trainData[stock1]/(trainData[stock1][0])
  pairsprice2 = trainData[stock2]/(trainData[stock2][0])
  plt.subplot(5,1,i)
  plt.plot_date(pairsprice1.index, pairsprice1,'r')
  plt.plot_date(pairsprice1.index, pairsprice2,'b')
  plt.legend([stock1,stock2],loc=4)
  plt.xlabel('Date')
  plt.ylabel('Standardized Price')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

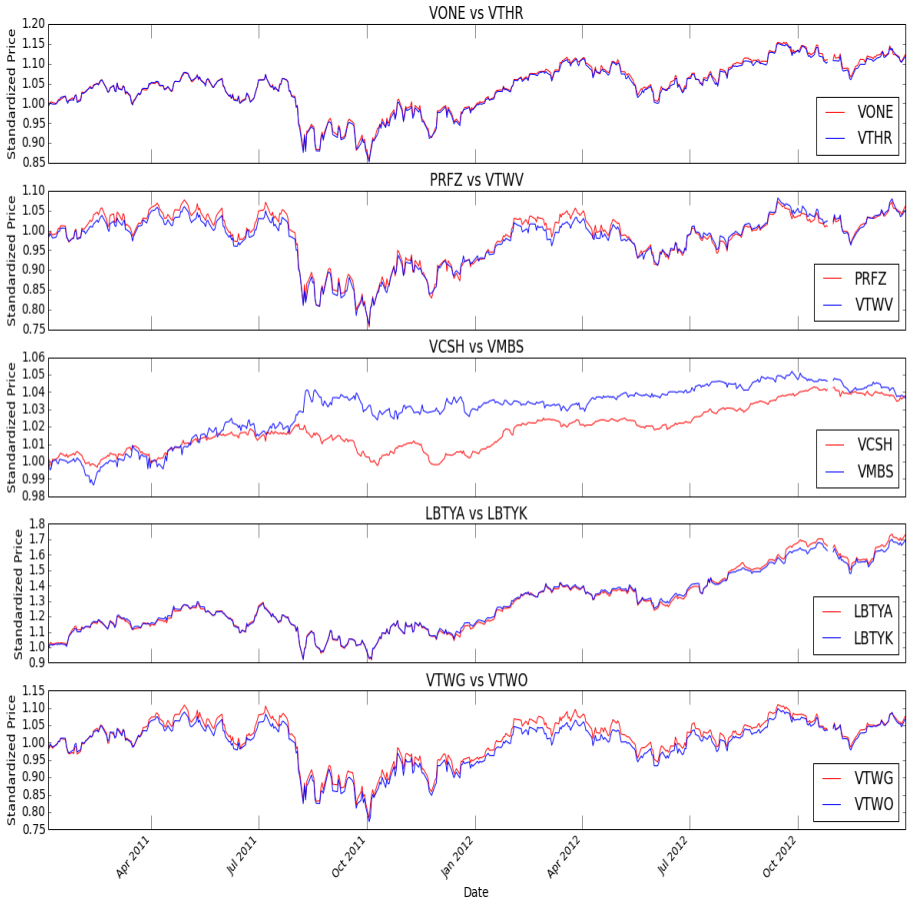

Figure 1: This figure shows the five 'closest' stock pairs using the mean SSD method. The values plotted are standardized by dividing that stock's daily closing price by the stock's closing price on January 1, 2011. The timeframe for this plot is January 1, 2011 to December 31, 2012.

In [8]:
```
# This cell plots the difference in standardized stock prices for the five
'closest' pairs of stocks using the mean SSD method.
# The plot covers the entire training dataset time range, from January 1, 2011,
to December 31, 2012.
# This will plot inline, and the following cell is a figure caption.
# A horizontal line at the mean gap value has been included to help distinguish
any trend in difference, also known as 'drift'

plt.figure(2)
i = 1
for a in smallssd.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = trainData[stock1]/(trainData[stock1][0])
  pairsprice2 = trainData[stock2]/(trainData[stock2][0])
  pairsgap = pairsprice1-pairsprice2
  plt.subplot(5,1,i)
  plt.plot(pairsprice1.index, pairsgap,'b')
  plt.axhline(y=mean(pairsgap), color='r')
  plt.xlabel('Date')
  plt.ylabel('Standardized Gap')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
```

```
plt.show()
```

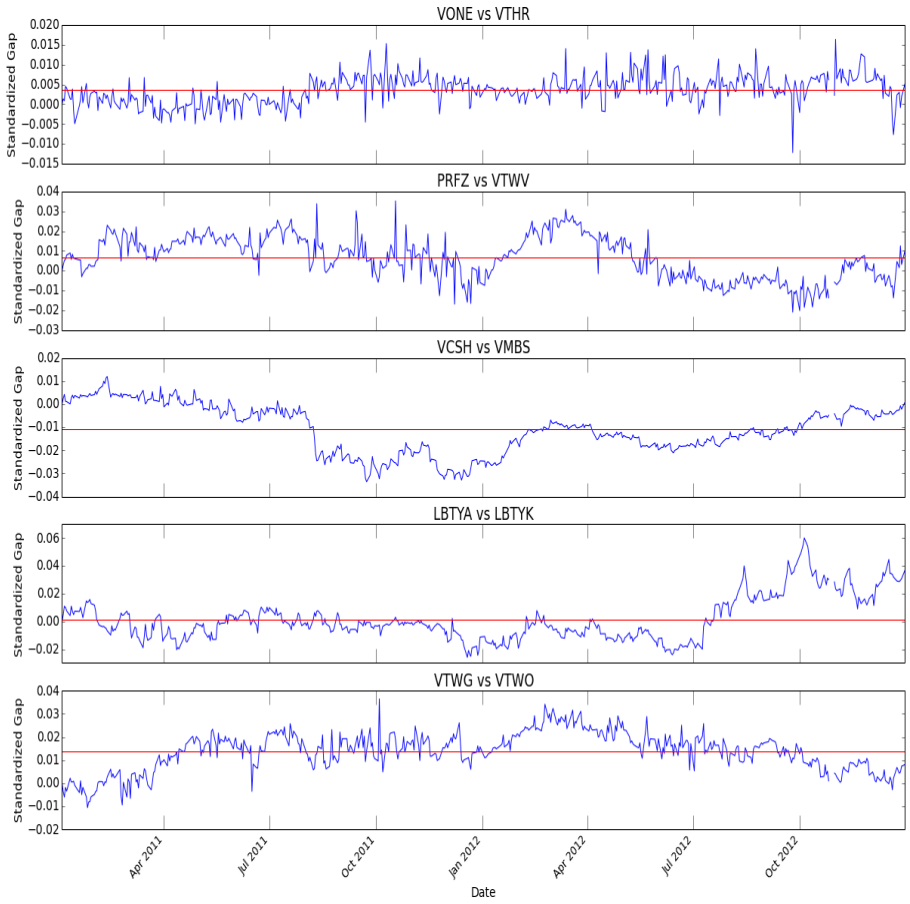

Figure 2: This figure shows the difference in standardized price of the five 'closest' stock pairs using the mean SSD method. A horizontal line at the mean gap value has been included to help visualize any trend in the difference, also known as 'drift'. The timeframe for this plot is January 1, 2011 to December 31, 2012.

Co-Integration (ADF) Test

```
# This cell calculates the p-value of the ADF (Augmented Dickey-
Fuller) co-integration test for each pair of stocks.
# Due to the number of pairs, this can take many hours.
# We have saved the output as a .pkl file that is loaded in future
steps, so we recommend skipping this and the next two cells.
# If you want to run the full analysis, change this cell from raw text
to code.

def cointegration_test(y, x):
        ctresult = stat.OLS(y, x).fit()
        return(ts.adfuller(ctresult.resid))


def coint(cointTrainingTlist):
    try:
        cointTrainingXname = cointTrainingTlist[0]
        cointTrainingYname = cointTrainingTlist[1]
        trainCointX = trainData[cointTrainingXname]
        trainCointY = trainData[cointTrainingYname]
        if min(trainCointX.count(), trainCointY.count()) > 495:
            trainxclean = trainCointX[trainCointX.notnull() &
trainCointY.notnull()]
            trainyclean = trainCointY[trainCointX.notnull() &
trainCointY.notnull()]
            trainxp = list(trainxclean)
            trainyp = list(trainyclean)
            return([(cointTrainingXname, cointTrainingYname)],
cointegration_test(trainxp,trainyp)[1]) # Get the p-value of test for
each pair
        else:
            return()
    except ValueError:
        return()
    except TypeError:
        return()


if __name__ == '__main__':
    trainCointPool = Pool(processes=4)
    # Test just the first 100 pairs - remove [0:100] for full test
    trainCointResult = pd.DataFrame(trainCointPool.map(coint,
tickerpairs[0:100]))

trainCointPool.close()

# This cell takes the ADF cointegration test output from the previous
cell and saves it in .pkl format for near-instant loading.
# This should only be run if you ran the previous cell; if you are
using the downloaded .pkl objects, skip this step.
# To run this cell, change this cell from raw text to code.

trainCointOutput = open('trainCointResult.pkl', 'wb')
```

```
pkl.dump(trainCointResult, trainCointOutput, -1)
trainCointOutput.close()

# This cell creates a .gz archive of the .pkl data so that it is small
enough for storage in the github repository.
# This should only be run if you intend to upload the data and need a
compressed format.
# To run this cell, change this cell from raw text to code.

!gzip -k trainCointResult.pkl
```

In [9]:
```
# This loads the training dataset ADF cointegration test results from
the .gz file available on github.
# This should be run to access already-generated results and will just take
a short while.
# If you are creating the data yourself, please skip this step.

!gunzip -c trainCointResult.pkl.gz > trainCointResult.pkl
trainCointInput = open('trainCointResult.pkl', 'rb')
trainCointResult = pkl.load(trainCointInput)
trainCointInput.close()
```

In [10]:
```
# This cell sorts the ADF cointegration test p-values and selects just the
100 pairs of stocks with the smallest p-values.

smallCoint = trainCointResult.sort(columns = 1)[0:100]
print smallCoint.head()
```

```
                          0             1
2876609    [(PATR, RDIB)]  0.000000e+00
2483677  [(LINTA, LINTB)]  0.000000e+00
124437     [(ADVS, HUBG)]  6.877388e-29
3091438  [(SENEA, SENEB)]  1.191406e-25
2876488    [(PATR, PNBK)]  1.387992e-25

[5 rows x 2 columns]
```

Controlling the false discovery rate: Benjamini–Hochberg procedure

```
# This cell is intended to control for false discovery rate using the
Benjamini-Hochberg Procedure
# There were still too many pairs, so we arbitrarily examine the top
100.
# Not in use, but to run change cell from raw text to code

# Procedure:
# Assume there are m hypothesis tests.
# Order the p-values in increasing order and call them
P_(1),P_(2),....,P_(m)
# Then the steps for the procedure are as follows:
# 1) For a given alpha, find the largest k such that P_(k) <=
(k/m)*alpha
# 2) Then reject all H_(i) for i = 1,2,...,k

from __future__ import division  #need this bc python cannot do
```

```
division for integers properly without it
Cointp = trainCointResult.sort(columns = 1)
cointpvalues = list(Cointp[Cointp.columns[1]])
m = len(cointpvalues)
alpha = 0.01  # False Discovery Rate (20% is used, Can tweak this if
necessary)

k = 0
while cointpvalues[k] <= ((k+1)/m)*alpha:  # Obtain the k from step 1)
    k += 1

print k
CointSaved = Cointp[:k]
print CointSaved.head() # Significant pairs under BH procedure
```

Cointegration Method Training Plots

In [11]:
```
# This cell plots the standardized stock prices for the five pairs of stocks
with the smallest ADF cointegration test p-values.
# The plot covers the entire training dataset time range, from January 1, 2011,
to December 31, 2012.
# This will plot inline, and the following cell is a figure caption.

plt.figure(3)
i = 1
for a in smallCoint.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = trainData[stock1]/(trainData[stock1][0])
  pairsprice2 = trainData[stock2]/(trainData[stock2][0])
  plt.subplot(5,1,i)
  plt.plot_date(pairsprice1.index, pairsprice1,'r')
  plt.plot_date(pairsprice1.index, pairsprice2,'b')
  plt.legend([stock1,stock2])
  plt.xlabel('Date')
  plt.ylabel('Standardized Price')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

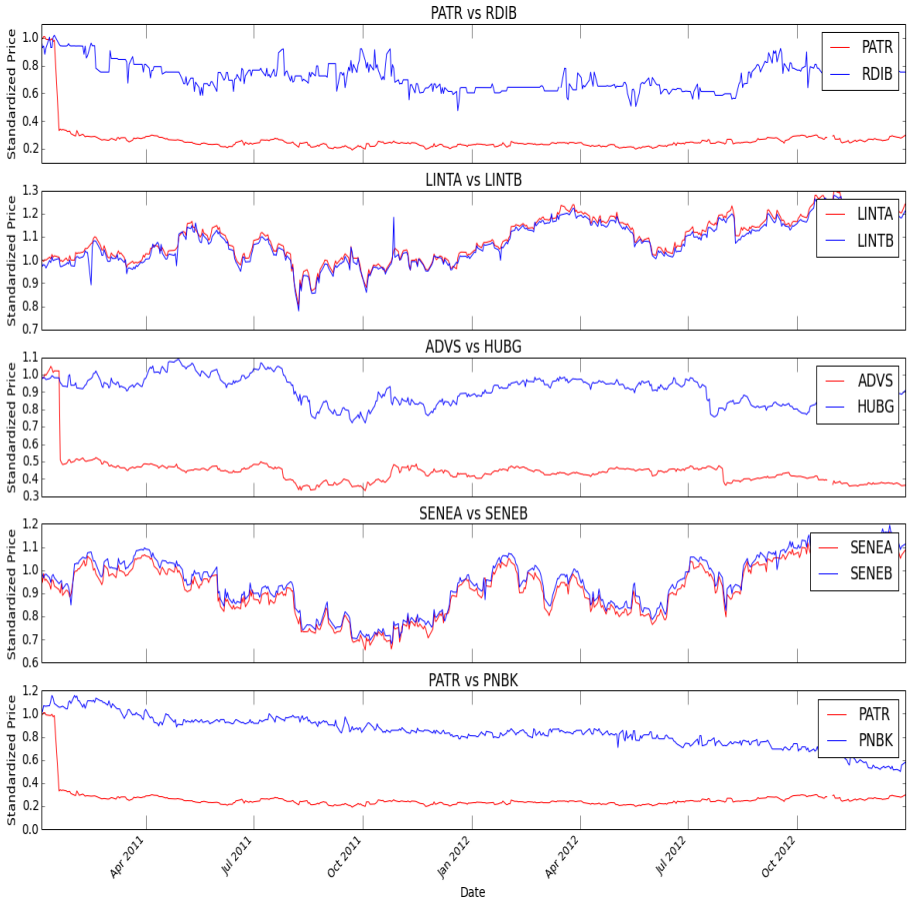

Figure 3: This figure shows the five stock pairs with the smallest ADF cointegration test p-values. The values plotted are standardized by dividing that stock's daily closing price by the stock's closing price on January 1, 2011. The timeframe for this plot is January 1, 2011 to December 31, 2012.

In [12]:
```
# This cell plots the difference in standardized stock prices for the five
pairs of stocks with the smallest ADF cointegration test p-values.
# The plot covers the entire training dataset time range, from January 1, 2011,
to December 31, 2012.
# This will plot inline, and the following cell is a figure caption.
# A horizontal line at the mean gap value has been included to help
distinguish any trend in difference, also known as 'drift'

plt.figure(4)
i = 1
for a in smallCoint.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = trainData[stock1]/(trainData[stock1][0])
  pairsprice2 = trainData[stock2]/(trainData[stock2][0])
  pairsgap = pairsprice1-pairsprice2
  plt.subplot(5,1,i)
  plt.plot(pairsprice1.index, pairsgap,'b')
  plt.axhline(y=mean(pairsgap), color='r')
  plt.xlabel('Date')
  plt.ylabel('Standardized Gap')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
```

```
plt.show()
```

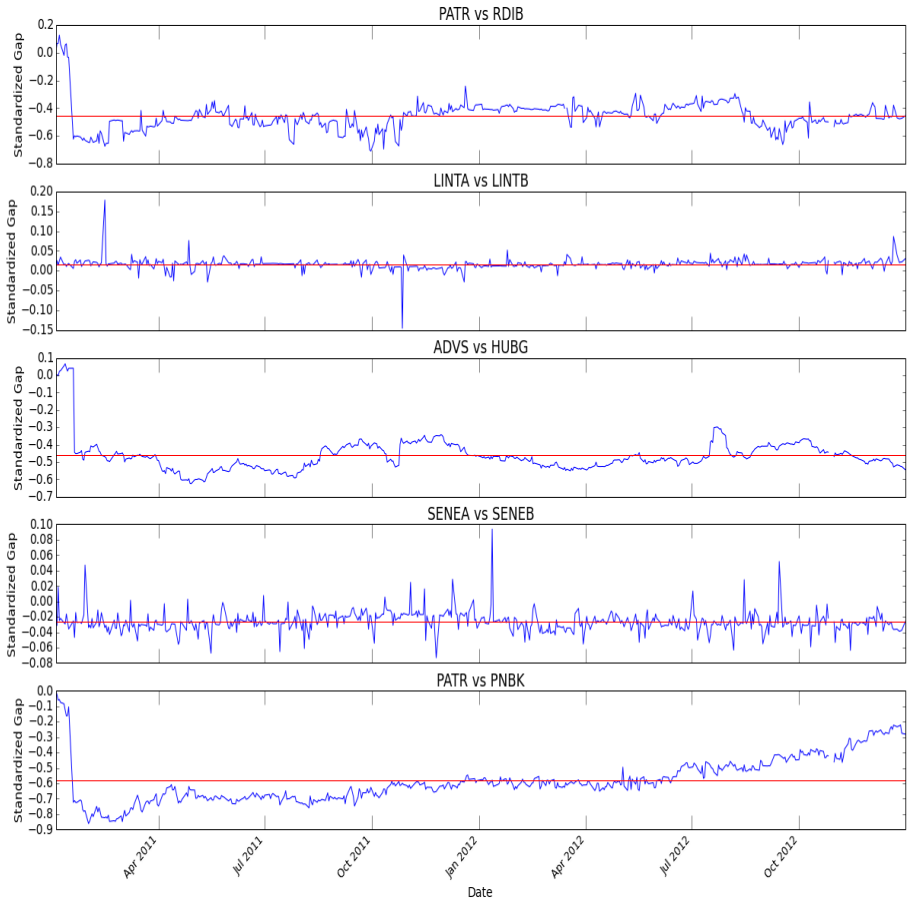

Figure 4: This figure shows the difference in standardized price of the five stock pairs with the smallest ADF cointegration test p-values. A horizontal line at the mean gap value has been included to help visualize any trend in the difference, also known as 'drift'. The timeframe for this plot is January 1, 2011 to December 31, 2012.

Correlation

```
# This cell calculates the correlation between each pair of stocks.
Due to the number of pairs, this can take many hours.
# We have saved the output as a .pkl file that is loaded in future
steps, so we recommend skipping this and the next two cells.
# If you want to run the full analysis, change this cell from raw text
to code.

def correlate(tlist):
    try:
        xname = tlist[0]
        yname = tlist[1]
        x = trainData[xname]
        y = trainData[yname]
        if min(x.count(), y.count()) > 490:
            corrs = x.corr(y)
            return([(xname, yname)], corrs)
        else:
            return()
    except ValueError:
        return()
    except TypeError:
        return()


if __name__ == '__main__':
    trainCorrPool = Pool(processes=4)
    # Test just the first 100 pairs - remove [0:100] for full test
    trainCorrResult = pd.DataFrame(trainCorrPool.map(correlate,
tickerpairs))

trainCorrPool.close()

# This cell takes the correlation output from the previous cell and
saves it in .pkl format for near-instant loading.
# This should only be run if you ran the previous cell; if you are
using the downloaded .pkl objects, skip this step.
# To run this cell, change this cell from raw text to code.

trainCorrOutput = open('trainCorrResult.pkl', 'wb')
pkl.dump(trainCorrResult, trainCorrOutput, -1)
trainCorrOutput.close()

pairscorrelated = trainCorrResult.sort(columns = 1,
ascending=False)[0:5]
print pairscorrelated

# This cell creates a .gz archive of the .pkl data so that it is small
enough for storage in the github repository.
# This should only be run if you intend to upload the data and need a
compressed format.
# To run this cell, change this cell from raw text to code.
```

```
!gzip -k trainCorrResult.pkl
```

In [13]:
```
# This loads the training dataset correlation results from the .gz file
available on github.
# This should be run to access already-generated results and will
just take a short while.
# If you are creating the data yourself, please skip this step.

!gunzip -c trainCorrResult.pkl.gz > trainCorrResult.pkl
trainCorrInput = open('trainCorrResult.pkl', 'rb')
trainCorrResult = pkl.load(trainCorrInput)
trainCorrInput.close()
```

In [14]:
```
# This cell sorts the correlation results and selects just the 100 most
positively correlated pairs of stocks.

trainCorrResult.columns = ['pair','corr']
trainCorrTop = trainCorrResult.dropna().sort(columns='corr', ascending=False)[0:100]
trainCorrTop.head()
```

Out [14]:

	pair	corr
3221878	[(VONE, VTHR)]	0.998255
2459008	[(LBTYA, LBTYK)]	0.998026
3178525	[(TECUA, TECUB)]	0.996770
1101073	[(CMCSA, CMCSK)]	0.995945
3052453	[(ROIA, ROIAK)]	0.995695

[5 rows x 2 columns]

Correlation Method Training Plots

In [15]:
```
# This cell plots the standardized stock prices for the five most
positively correlated pairs of stocks.
# The plot covers the entire training dataset time range, from January 1, 2011,
to December 31, 2012.
# This will plot inline, and the following cell is a figure caption.

plt.figure(5)
i = 1
for a in trainCorrTop.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = trainData[stock1]/(trainData[stock1][0])
  pairsprice2 = trainData[stock2]/(trainData[stock2][0])
  plt.subplot(5,1,i)
  plt.plot_date(pairsprice1.index, pairsprice1,'r')
  plt.plot_date(pairsprice1.index, pairsprice2,'b')
  plt.legend([stock1,stock2], loc=7)
  plt.xlabel('Date')
  plt.ylabel('Standardized Price')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

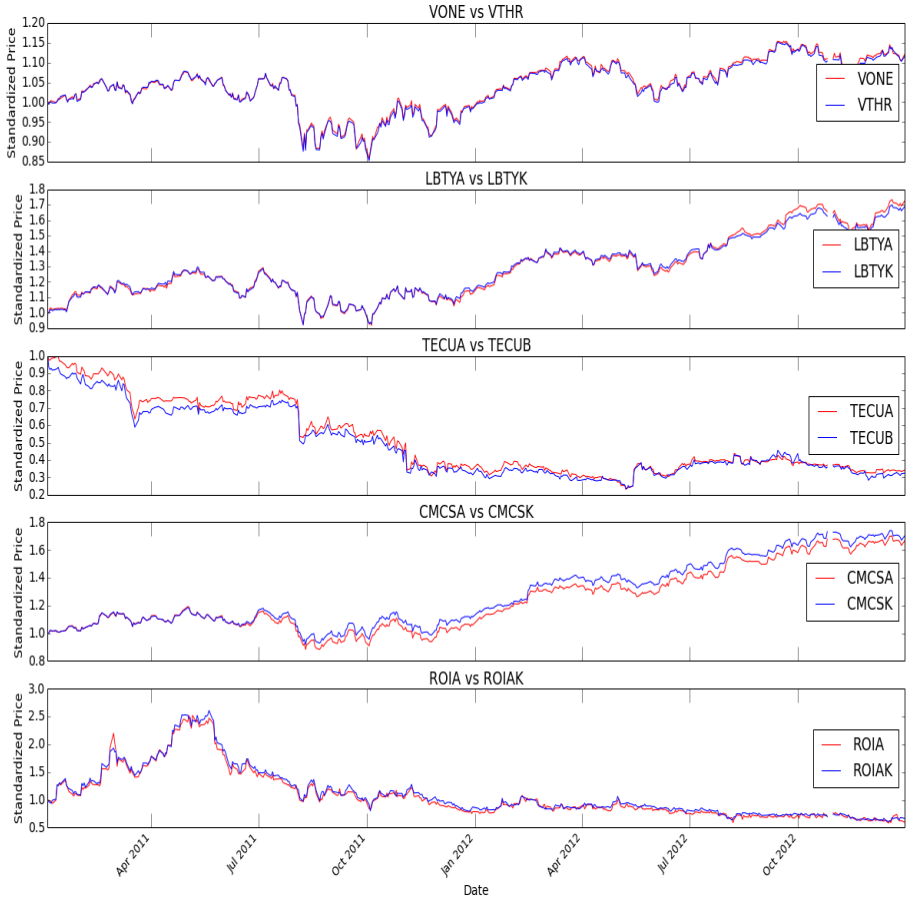

Figure 5: This figure shows the five most positively correlated stock pairs. The values plotted are standardized by dividing that stock's daily closing price by the stock's closing price on January 1, 2011. The timeframe for this plot is January 1, 2011 to December 31, 2012.

In [16]:
```
# This cell plots the difference in standardized stock prices for the
five most positively correlated pairs of stocks.
# The plot covers the entire training dataset time range, from January 1, 2011,
to December 31, 2012.
# This will plot inline, and the following cell is a figure caption.
# A horizontal line at the mean gap value has been included to help
distinguish any trend in difference, also known as 'drift'

plt.figure(6)
i = 1
for a in trainCorrTop.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = trainData[stock1]/(trainData[stock1][0])
  pairsprice2 = trainData[stock2]/(trainData[stock2][0])
  pairsgap = pairsprice1-pairsprice2
  plt.subplot(5,1,i)
  plt.plot(pairsprice1.index, pairsgap,'b')
  plt.axhline(y=mean(pairsgap), color='r')
  plt.xlabel('Date')
  plt.ylabel('Standardized Gap')
  plt.title(stock1+' vs '+stock2)
  i += 1
```

```
plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

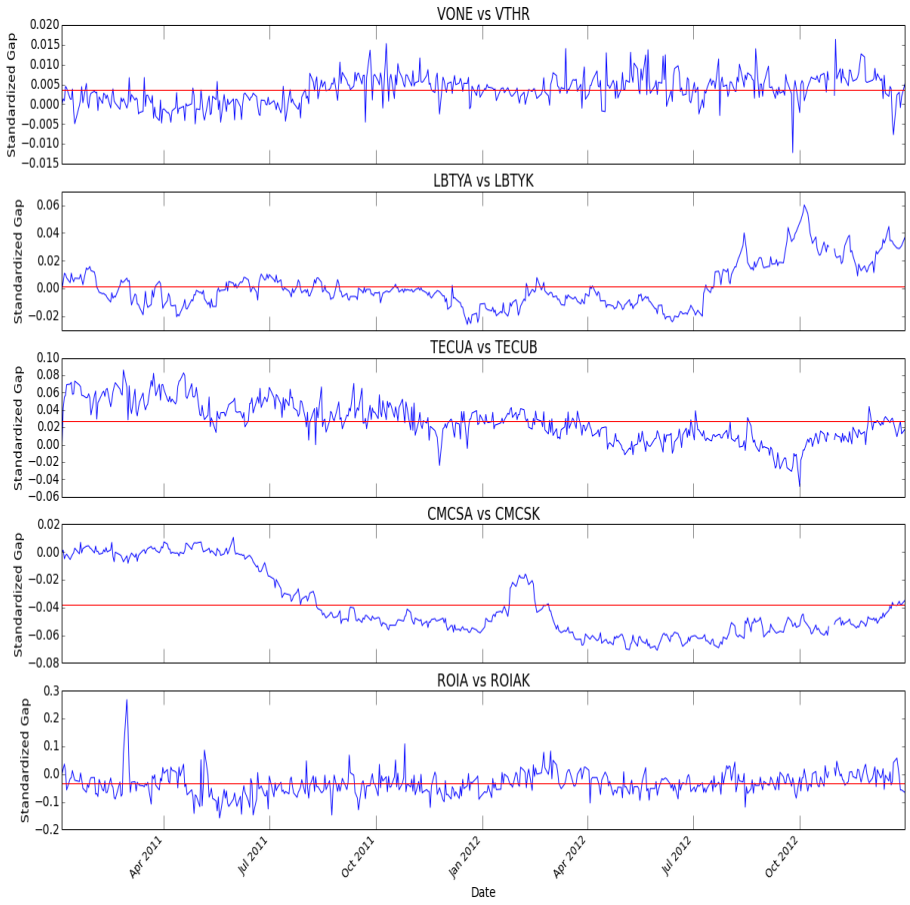

Figure 6: This figure shows the difference in standardized price of the five most positively correlated stock pairs. A horizontal line at the mean gap value has been included to help visualize any trend in the difference, also known as 'drift'. The timeframe for this plot is January 1, 2011 to December 31, 2012.

```
# This cell plots a histogram of the correlation of the stock pairs
# This did not seem to add to the evaluation of correlation as a
selection method.
# This cell is not in use, but change cell from raw text to code to
run.

correls = list(trainCorrResult[trainCorrResult.columns[1]])
categories = ['< 0.0','0.0-0.1','0.1-0.2','0.2-0.3','0.3-0.4','0.4-0.5
','0.5-0.6','0.6-0.7','0.7-0.8','0.8-0.9','0.9-1.0']

freq0 = len([cor for cor in correls if cor < 0.0])
freq1 = len([cor for cor in correls if 0.0 <= cor < 0.1])
freq2 = len([cor for cor in correls if 0.1 <= cor < 0.2])
freq3 = len([cor for cor in correls if 0.2 <= cor < 0.3])
freq4 = len([cor for cor in correls if 0.3 <= cor < 0.4])
freq5 = len([cor for cor in correls if 0.4 <= cor < 0.5])
freq6 = len([cor for cor in correls if 0.5 <= cor < 0.6])
freq7 = len([cor for cor in correls if 0.6 <= cor < 0.7])
```

```
freq8 = len([cor for cor in correls if 0.7 <= cor < 0.8])
freq9 = len([cor for cor in correls if 0.8 <= cor < 0.9])
freq10 = len([cor for cor in correls if 0.9 <= cor < 1.0])

cat = np.arange(len(categories))
frequencies =
[freq0,freq1,freq2,freq3,freq4,freq5,freq6,freq7,freq8,freq9,freq10]

width = 1.0     # gives histogram aspect to the bar diagram

ax = plt.axes()
ax.set_xticks(cat + (width / 2))
ax.set_xticklabels(categories)

plt.bar(cat,frequencies,width)
plt.xlabel('Correlation Ranges')
plt.ylabel('Frequencies')
plt.title('Histogram of Correlation of Stock Pairs')
plt.show()
```

Test Data Set

Now that we have determined the top pairs of stocks in the training data set using our suggested methods, we will evaluate those pairs in the test data set to see how well the methods hold up over time.

Distance Method

In [17]:
```
# This cell calculates the mean SSD for each pair of stocks in the test data set.

def dist(tlist):
    xname = tickerpairs[tlist][0]
    yname = tickerpairs[tlist][1]
    x = testData[xname]/(testData[xname][0])
    y = testData[yname]/(testData[yname][0])
    # Remove any NA values
    z = (x-y).dropna()
    # Only consider pairs with most of data available
    if len(z) > 248:
        return([(xname, yname)], sum(map(lambda z:z**2, z)))
    else:
        return()

if __name__ == '__main__':
    testDistPool = Pool(processes=4)
    # Tests just the number of pairs selected in smallssd
    testDistResult = pd.DataFrame(testDistPool.map(dist, smallssd.index))

testDistPool.close()
```

In [19]:
```
# This cell plots the standardized stock prices for the five 'closest'
pairs of stocks using the training mean SSD method.
# The plot covers the entire test dataset time range, from January 1, 2013,
to December 31, 2013.
# This will plot inline, and the following cell is a figure caption.

plt.figure(1)
i = 1
for a in smallssd.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = testData[stock1]/(testData[stock1][0])
  pairsprice2 = testData[stock2]/(testData[stock2][0])
  plt.subplot(5,1,i)
  plt.plot_date(pairsprice1.index, pairsprice1,'r')
  plt.plot_date(pairsprice1.index, pairsprice2,'b')
  plt.legend([stock1,stock2], loc=4)
  plt.xlabel('Date')
  plt.ylabel('Standardized Price')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

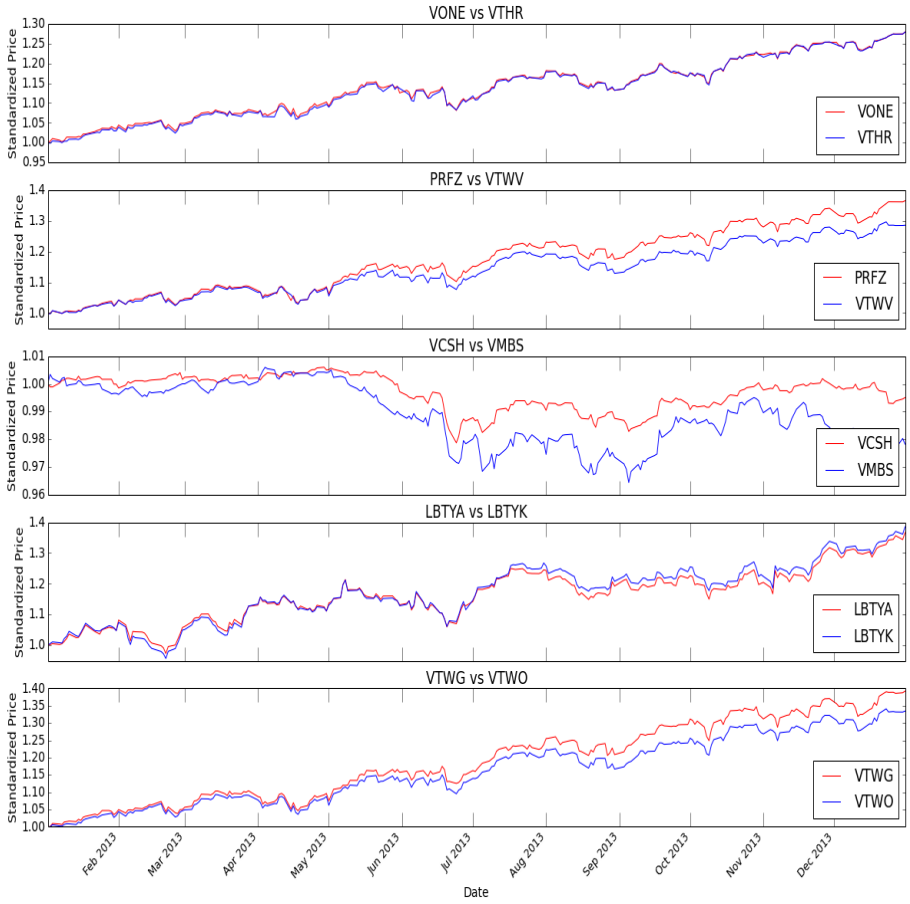

Figure 7: This figure shows the five 'closest' stock pairs using the training mean SSD method. The values plotted are standardized by dividing that stock's daily closing price by the stock's closing price on January 1, 2013. The timeframe for this plot is January 1, 2013 to December 31, 2013.

In [20]:
```
# This cell plots the difference in standardized stock prices for the five
'closest' pairs of stocks using the training mean SSD method.
# The plot covers the entire test dataset time range, from January 1, 2013,
to December 31, 2013.
# This will plot inline, and the following cell is a figure caption.
# A horizontal line at y=0 has been included to help distinguish any
trend in difference, also known as 'drift'

plt.figure(2)
i = 1
for a in smallssd.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  #Updated to reflect standardized price differential
  pairsprice1 = testData[stock1]/(testData[stock1][0])
  pairsprice2 = testData[stock2]/(testData[stock2][0])
  pairsgap = pairsprice1-pairsprice2
  plt.subplot(5,1,i)
  plt.plot(pairsprice1.index, pairsgap,'b')
  plt.axhline(y=0, color='r')
  plt.xlabel('Date')
  plt.ylabel('Standardized Gap')
  plt.title(stock1+' vs '+stock2)
  i += 1
```

```
plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

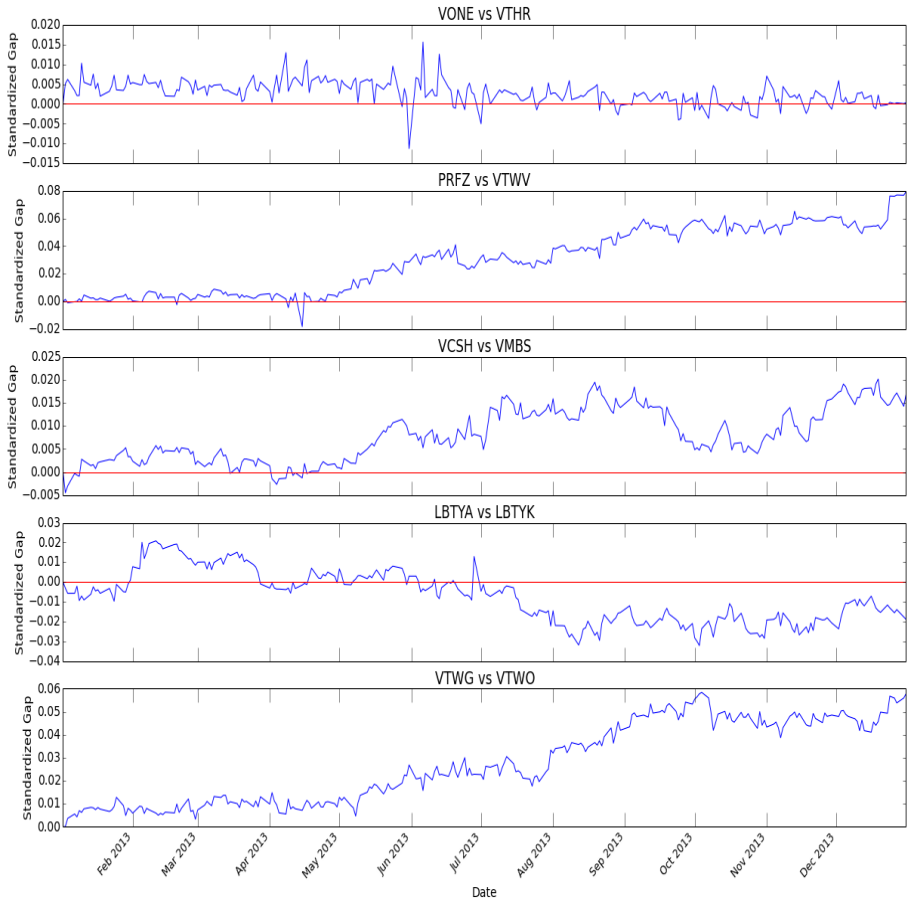

Figure 8: This figure shows the difference in standardized price of the five 'closest' stock pairs using the training mean SSD method. A horizontal line at y=0 has been included to help visualize any trend in the difference, also known as 'drift'. For the test data set, we want fluctuation around y=0 for a successful pairs trade. The timeframe for this plot is January 1, 2013 to December 31, 2013.

In [18]:
```
# This cell creates a scatterplot of training mean SSD vs test mean SSD.
# The top 100 'closest' stock pairs from the training mean SSD method are included.

plt.scatter(smallssd[1], testDistResult[1])
plt.vlines(.3, -2, 30, colors='r')
plt.text(.31,11,'x = .3')
plt.xlabel('Training SSD')
plt.ylabel('Test SSD')
plt.title('Consistency of SSD Method')
plt.show()
```

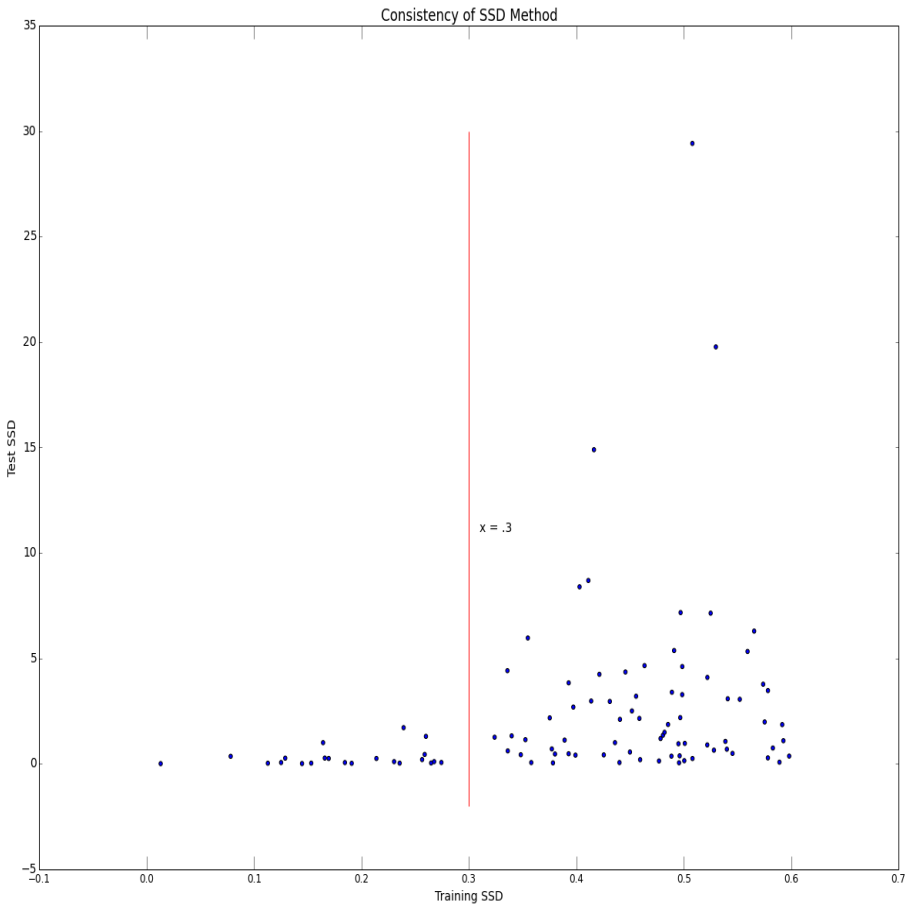

Figure 9: This figure compares the training mean SSD to the test mean SSD for the top 100 'closest' stock pairs. The scatter plot shows a high level of heteroscedasticity. A vertical line at x=0.3 has been included to show an observed difference in variance; this should not be taken as a causal inference.

Cointegration

In [21]:
```
# This cell calculates the ADF cointegration p-value for each pair
of stocks in the test data set.

def cointegration_test(y, x):
        result = stat.OLS(y, x).fit()
        return(ts.adfuller(result.resid))


def coint(cointTestingTlist):
    try:
        cointTestingXname = tickerpairs[cointTestingTlist][0]
        cointTestingYname = tickerpairs[cointTestingTlist][1]
        testCointX = testData[cointTestingXname]
        testCointY = testData[cointTestingYname]
        if min(testCointX.count(), testCointY.count()) > 248:
            testxclean = testCointX[testCointX.notnull() & testCointY.notnull()]
            testyclean = testCointY[testCointX.notnull() & testCointY.notnull()]
            testxp = list(testxclean)
            testyp = list(testyclean)
            return([(cointTestingXname, cointTestingYname)],
cointegration_test(testxp,testyp)[1]) # Get the p-value of test for each pair
        else:
            return()
    except ValueError:
        return()
    except TypeError:
        return()


if __name__ == '__main__':
    testCointPool = Pool(processes=4)
    # Test just the first 100 pairs - remove [0:100] for full test
    testCointResult = pd.DataFrame(testCointPool.map(coint, smallCoint.index))

testCointPool.close()
testsmallCoint = testCointResult.sort(columns = 1)[0:100]
print testsmallCoint.head()
```

```
                    0             1
1    [(LINTA, LINTB)]  1.231724e-26
3    [(SENEA, SENEB)]  1.257753e-08
85    [(ROIA, ROIAK)]  8.997874e-04
91     [(ADVS, LYTS)]  4.922607e-03
42     [(FBMS, PATR)]  7.081298e-03

[5 rows x 2 columns]
```

In [23]:
```
# Note scale of training p-values
smallCoint.head()
```

Out [23]:

```
                          0             1
2876609     [(PATR, RDIB)]  0.000000e+00
2483677  [(LINTA, LINTB)]  0.000000e+00
124437      [(ADVS, HUBG)]  6.877388e-29
3091438  [(SENEA, SENEB)]  1.191406e-25
2876488     [(PATR, PNBK)]  1.387992e-25
```

```
[5 rows x 2 columns]
```

In [24]:
```
# Note same pairs test p-values - 3 of 5 are NOT SIGNIFICANT
testCointResult.head()
```

Out [24]:

	0	1
0	[(PATR, RDIB)]	1.496377e-01
1	[(LINTA, LINTB)]	1.231724e-26
2	[(ADVS, HUBG)]	4.528192e-01
3	[(SENEA, SENEB)]	1.257753e-08
4	[(PATR, PNBK)]	9.694963e-01

[5 rows x 2 columns]

In [25]:
```
# Concatenate the cointegration results for further evaluation

df = pd.DataFrame(smallCoint.reset_index())
df['2'] = testCointResult[1]
df.dropna(axis=0,how='any')
df.columns = ['index','pair','train','test']
df.head()
```

Out [25]:

	index	pair	train	test
0	2876609	[(PATR, RDIB)]	0.000000e+00	1.496377e-01
1	2483677	[(LINTA, LINTB)]	0.000000e+00	1.231724e-26
2	124437	[(ADVS, HUBG)]	6.877388e-29	4.528192e-01
3	3091438	[(SENEA, SENEB)]	1.191406e-25	1.257753e-08
4	2876488	[(PATR, PNBK)]	1.387992e-25	9.694963e-01

[5 rows x 4 columns]

```
# This cell creates a box plot, but is not currently in use.  Change
from raw text to code in order to run.
# Note that training data box plot does not even appear due to scale
issues - the training p-values are so minute

bp = df.boxplot(column=['train','test'])
plt.show()
```

In [26]:
```
# This cell plots the standardized stock prices for the five pairs
of stocks with the smallest training ADF cointegration test p-values.
# The plot covers the entire test dataset time range, from January 1, 2013,
to December 31, 2013.
# This will plot inline, and the following cell is a figure caption.

plt.figure(1)
i = 1
for a in smallCoint.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = testData[stock1]/(testData[stock1][0])
  pairsprice2 = testData[stock2]/(testData[stock2][0])
  plt.subplot(5,1,i)
  plt.plot_date(pairsprice1.index, pairsprice1,'r')
  plt.plot_date(pairsprice1.index, pairsprice2,'b')
  plt.legend([stock1,stock2])
```

```
  plt.xlabel('Date')
  plt.ylabel('Standardized Price')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

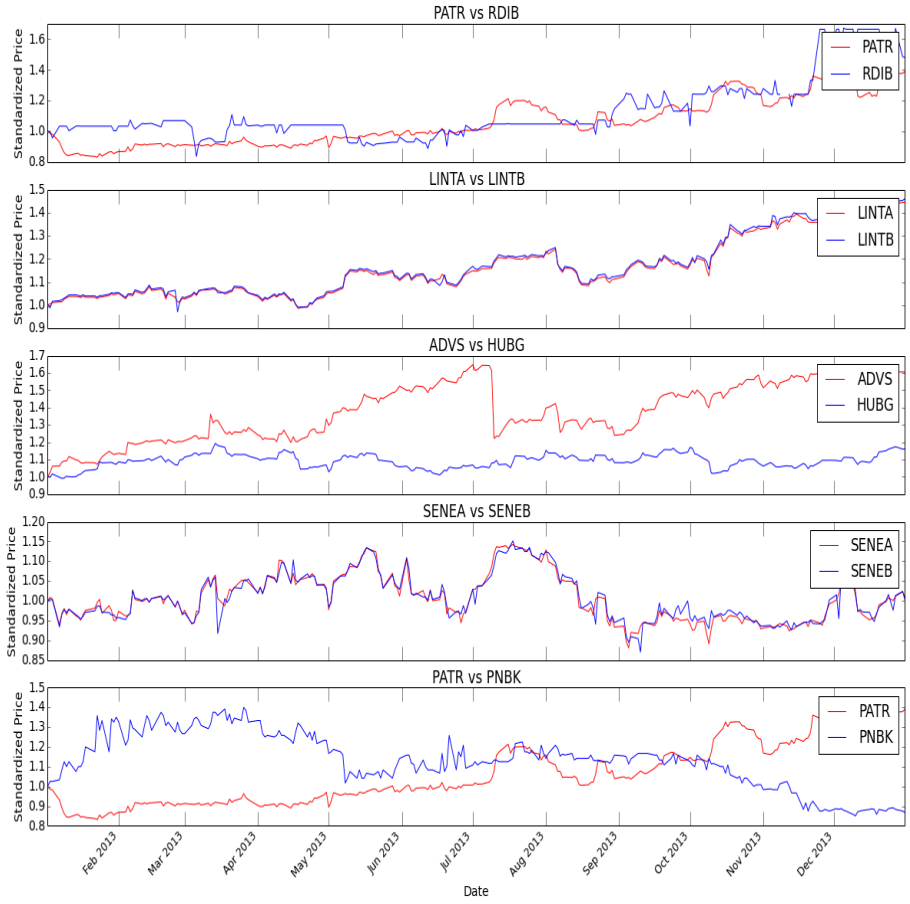

Figure 10: This figure shows the five stock pairs with the smallest training ADF cointegration test p-values. The values plotted are standardized by dividing that stock's daily closing price by the stock's closing price on January 1, 2013. The timeframe for this plot is January 1, 2013 to December 31, 2013.

In [27]:
```
# This cell plots the difference in standardized stock prices for the five
pairs of stocks with the smallest training ADF cointegration test p-values.
# The plot covers the entire test dataset time range, from January 1, 2013,
to December 31, 2013.
# This will plot inline, and the following cell is a figure caption.
# A horizontal line at y=0 has been included to help distinguish any trend
in difference, also known as 'drift'

plt.figure(2)
i = 1
for a in smallCoint.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  #Updated to reflect standardized price differential
  pairsprice1 = testData[stock1]/(testData[stock1][0])
  pairsprice2 = testData[stock2]/(testData[stock2][0])
```

```
  pairsgap = pairsprice1-pairsprice2
  plt.subplot(5,1,i)
  plt.plot(pairsprice1.index, pairsgap,'b')
  plt.axhline(y=0, color='r')
  plt.xlabel('Date')
  plt.ylabel('Standardized Gap')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

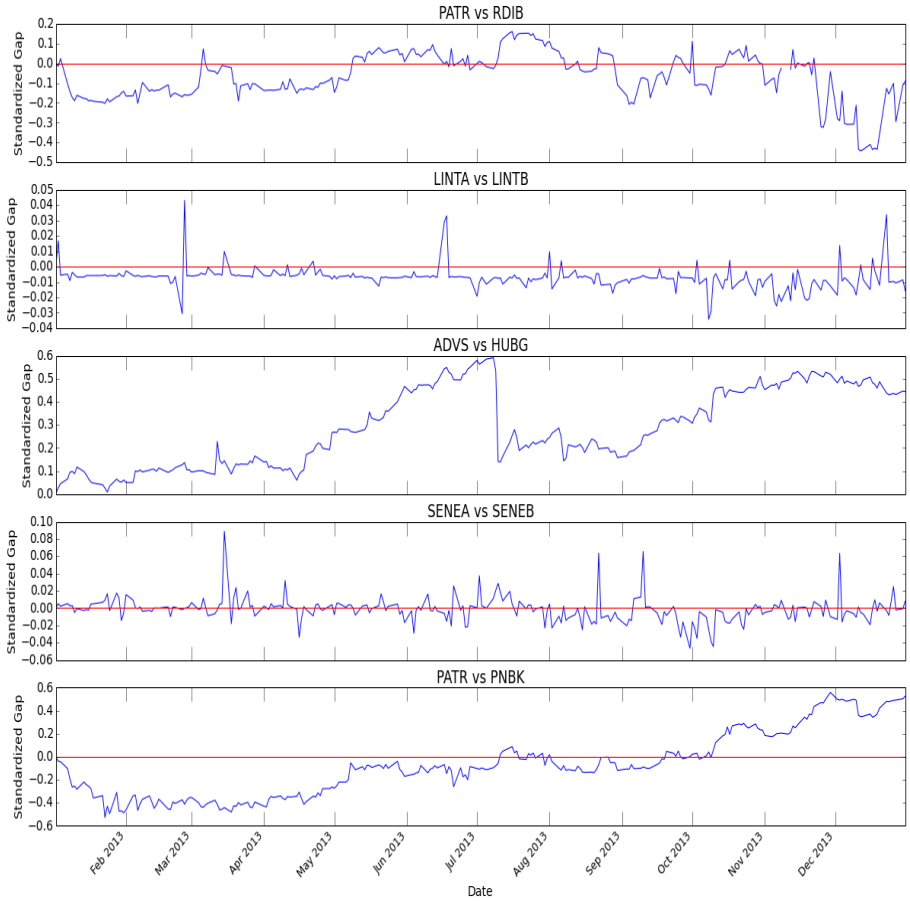

Figure 11: This figure shows the difference in standardized price of the five stock pairs with the smallest ADF cointegration test p-values. A horizontal line at y=0 has been included to help visualize any trend in the difference, also known as 'drift'. For the test data set, we want fluctuation around y=0 for a successful pairs trade. The timeframe for this plot is January 1, 2013 to December 31, 2013.

In [22]:
```
# This cell creates a scatterplot of training mean SSD vs test mean SSD.
# The top 100 'closest' stock pairs from the training mean SSD method are included.

plt.scatter(smallCoint[1][0:100], testCointResult[1][0:100])
plt.xlim(-1e-13,1e-12)
plt.xlabel('Training Cointegration')
plt.ylabel('Test Cointegration')
plt.title('Consistency of Cointegration Method')
plt.show()
```

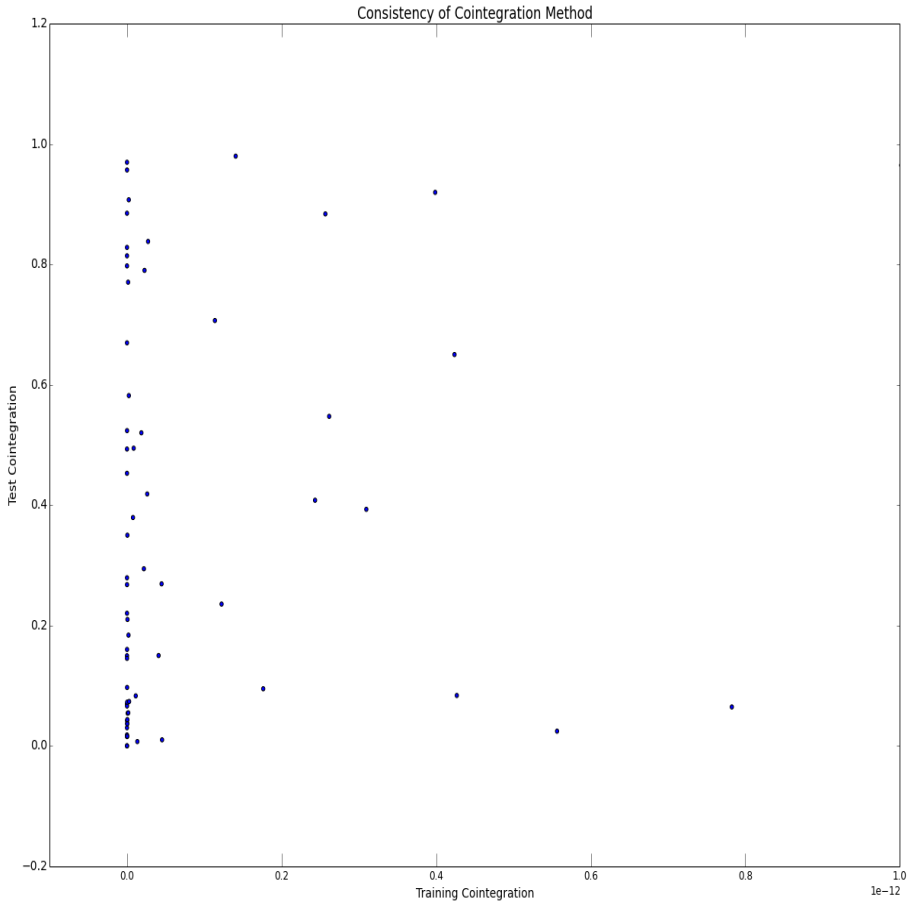

Figure 12: This figure compares the training ADF cointegration p-value to the test ADF cointegration p-value for the top 100 stock pairs with the smallest training p-value. The scatter plot shows a nearly uniform distribution across all training p-values, indicating little to no reliability for cointegration as a selection method.

Correlation

In [28]:
```
# Parallel processing for correlation test
# Consider updating to take advantage of pandas.DataFrame.corr()
but would need to change handling of output

def correlate(tlist):
    try:
        xname = tickerpairs[tlist][0]
        yname = tickerpairs[tlist][1]
        x = testData[xname]
        y = testData[yname]
        if min(x.count(), y.count()) > 240:
            corrs = x.corr(y)
            return([(xname, yname)], corrs)
        else:
            return()
    except ValueError:
        return()
    except TypeError:
        return()


if __name__ == '__main__':
    testCorrPool = Pool(processes=4)
    #Test just the first 100 pairs - remove [0:100] for full test
    testCorrResult = pd.DataFrame(testCorrPool.map(correlate, trainCorrTop.index))

testCorrPool.close()
```

In [30]:
```
# This cell plots the standardized stock prices for the training set
five most positively correlated pairs of stocks.
# The plot covers the entire test dataset time range, from January 1, 2013,
to December 31, 2013.
# This will plot inline, and the following cell is a figure caption.

plt.figure(1)
i = 1
for a in trainCorrTop.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  pairsprice1 = testData[stock1]/(testData[stock1][0])
  pairsprice2 = testData[stock2]/(testData[stock2][0])
  plt.subplot(5,1,i)
  plt.plot_date(pairsprice1.index, pairsprice1,'r')
  plt.plot_date(pairsprice1.index, pairsprice2,'b')
  plt.legend([stock1,stock2],loc=4)
  plt.xlabel('Date')
  plt.ylabel('Standardized Price')
  plt.title(stock1+' vs '+stock2)
  i += 1

plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

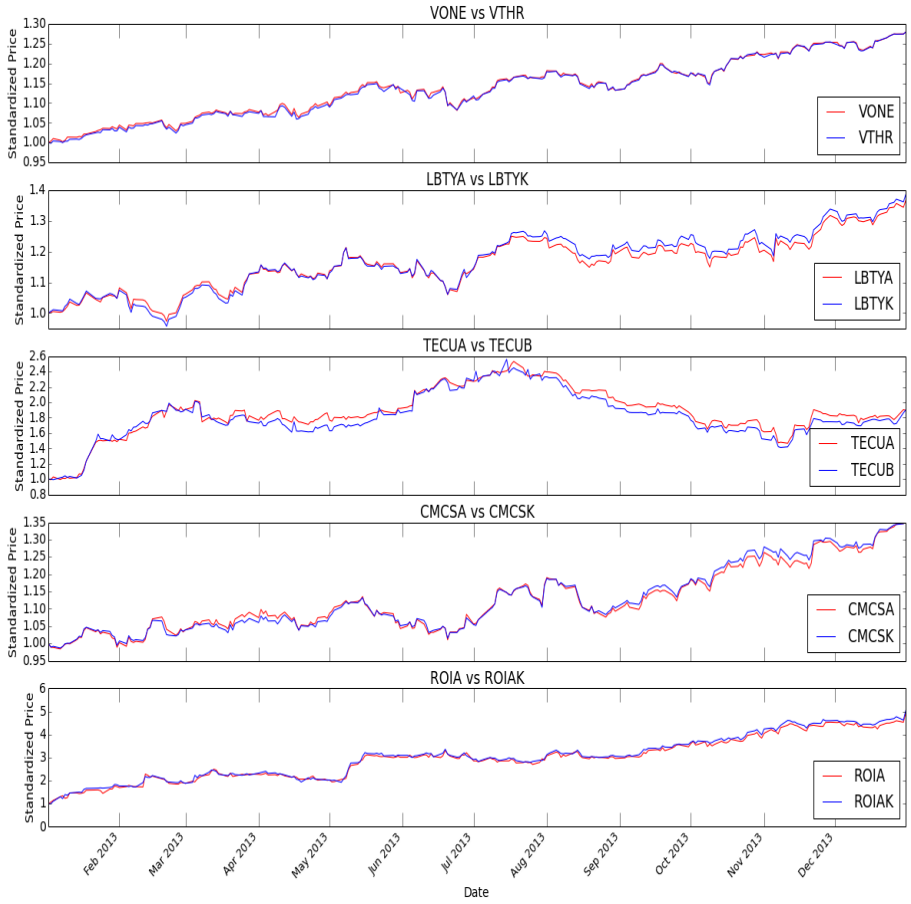

Figure 13: This figure shows the five most positively correlated stock pairs from the training correlation method. The values plotted are standardized by dividing that stock's daily closing price by the stock's closing price on January 1, 2013. The timeframe for this plot is January 1, 2013 to December 31, 2013.

In [31]:
```
# This cell plots the difference in standardized stock prices for the
five most positively correlated pairs of stocks.
# The plot covers the entire test dataset time range, from January 1, 2013,
to December 31, 2013.
# This will plot inline, and the following cell is a figure caption.
# A horizontal line at y=0 has been included to help distinguish any trend
in difference, also known as 'drift'

plt.figure(2)
i = 1
for a in trainCorrTop.index[0:5]:
  stock1 = tickerpairs[a][0]
  stock2 = tickerpairs[a][1]
  #Updated to reflect standardized price differential
  pairsprice1 = testData[stock1]/(testData[stock1][0])
  pairsprice2 = testData[stock2]/(testData[stock2][0])
  pairsgap = pairsprice1-pairsprice2
  plt.subplot(5,1,i)
  plt.plot(pairsprice1.index, pairsgap,'b')
  plt.axhline(y=0, color='r')
  plt.xlabel('Date')
  plt.ylabel('Standardized Gap')
  plt.title(stock1+' vs '+stock2)
  i += 1
```

```
plt.gcf().autofmt_xdate(rotation=45)
plt.show()
```

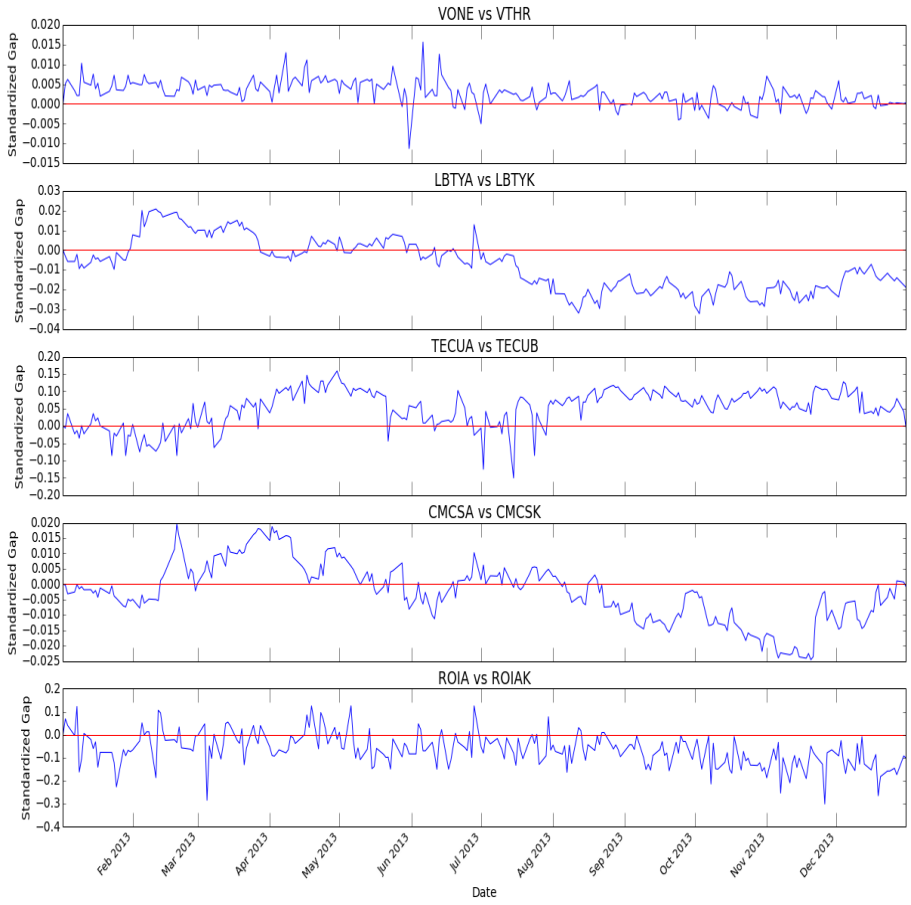

Figure 14: This figure shows the difference in standardized price of the five most positively correlated stock pairs from the training correlation method. A horizontal line at y=0 has been included to help visualize any trend in the difference, also known as 'drift'. For the test data set, we want fluctuation around y=0 for a successful pairs trade. The timeframe for this plot is January 1, 2013 to December 31, 2013.

In [29]:
```
# This cell creates a scatterplot of training correlation vs test correlation.
# The top 100 most positively correlated stock pairs from the training set
are included.

testCorrResult.columns = ['pair','corr']
plt.scatter(trainCorrTop['corr'], testCorrResult['corr'])
plt.vlines(.993,-1,1, colors='r')
plt.text(.994,0,'x = .993')
plt.xlabel('Training Correlation')
plt.ylabel('Test Correlation')
plt.title('Consistency of Correlation Method')
plt.show()
```

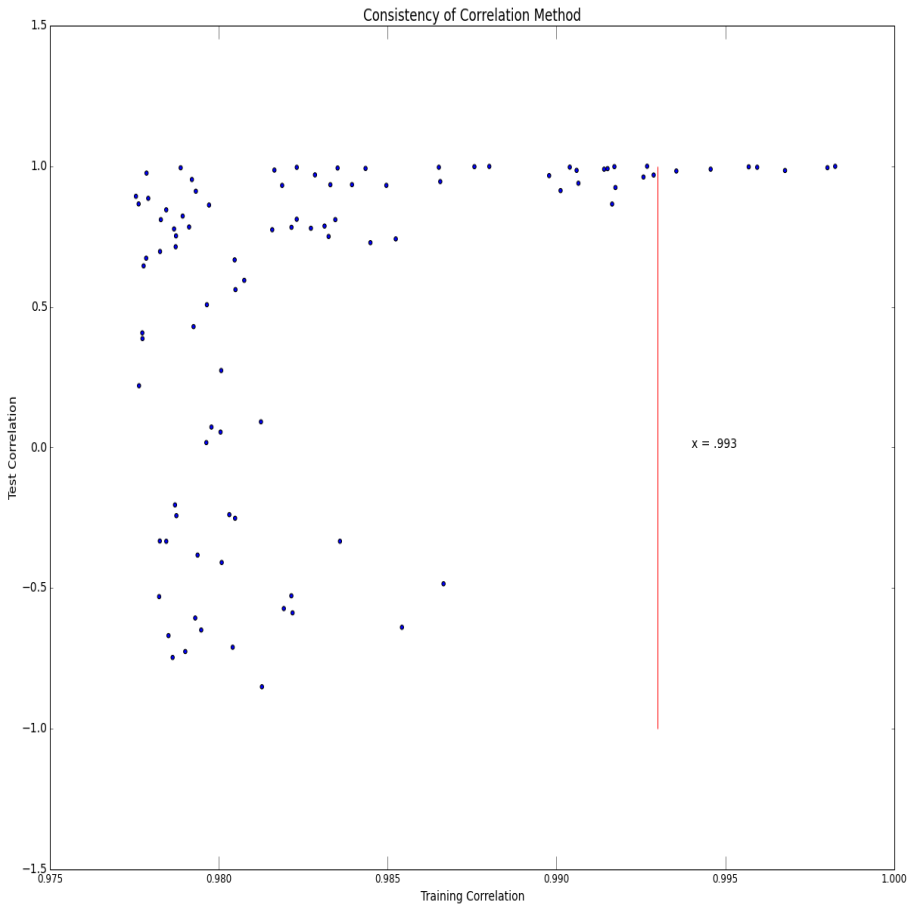

Figure 15: This figure compares the training correlation to the test correlation for the top 100 most positively correlated training stock pairs. The scatter plot shows a high level of heteroscedasticity. A vertical line at x=0.993 has been included to show an observed difference in variance; this should not be taken as a causal inference.

Conclusion

In summary, we have found no evidence that any of these statistical methods alone is enough to select a pair of stocks for pairs trading. Further, we have not found any evidence refuting the anecdotal information that stocks should be related in a known manner to be suitable for pairs trading.

Of the three methods, the distance method performed the worst, with the largest drift away from the mean over time. Correlation performed somewhat better, but both correlation and distance showed little consistency between the training and test data. Cointegration showed almost no consistency between training and test, yet the related stocks selected by cointegration showed promising results.

We would suggest further research into selection of pairs of related stocks using the cointegration method.