Notes: df.ix[:] : returns sliced rows based on passed indexes

# Predicting US Stocks Returns Direction (UP/DOWN)

## Implementing Anchor bias theory

### Problem Identification :

Investors are anchored to the long term moving averages. The long term moving average is defined by the 252 moving average, and the short term is defined by the 21-Day moving average. The distance between the two moving averages is the moving average distance (MAD = 21-DAY MA / 252-DAY MA). When the MAD>1, the ditance is called a positive spread and when MAD< 1, the distance is caleed a negative spread.

The ancnchor bias theory, published in a research paper by Avramov, Kaplanski, Subrahmanyam(2018), states that when MAD spread is positive positive announcment (sentiment) drive the price of the stocks to go up more than than negative sentiment drive the price to go down. However, when MAD spread is negative, negative sentiment drives price to go down more than positive sentiment drives re price to go up. Noting that the larger/ smaller the MAD, in both cases, the more effective is the strategy

The model proposed is to predict US stocks returns ( +/-) based on several features but mainly on a BUY or SELL signal. The engineered feature, named trading signal is the main feature which is processed by the constructed pipeline. The BUY signal is construcetd by getting positive sentiment from 2 databases (Sentdex and stocktwits), a 7 days previous senitiment score and a positive MAD greater than 1.2. The SELL signal is set based on negative sentiment scores from 2 databases, also a 7 day previous negative score and a negative MAD less than 0.8.

The stated signals are passed to the pipeline to pass through more than 8000 US stocks and filter out each day, the stocks that passed the criteria. Several screens where passed to the timeline to insure no stock has a null sentiment score (in any of the two databases) or a zero return ( which was actually found).Several other features where passed to the pipeline to output a dataframe of the filtered stocks. After doing the nessary transformations, the data is based to two machine learing algorithms.

## Data Gathering using a Pipeline:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('white')
```

In [2]:
```
# Import Pipeline class and datasets
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data import USEquityPricing
from quantopian.pipeline.data.psychsignal import stocktwits
# from quantopian.interactive.data.sentdex import sentiment
from quantopian.pipeline.data.sentdex import sentiment
from quantopian.pipeline.factors import CustomFactor, MarketCap, Latest
from quantopian.pipeline.classifiers.fundamentals import SuperSector
# Import built-in moving average calculation
from quantopian.pipeline.factors import SimpleMovingAverage, DailyReturns, Ret
urns

# Import built-in trading universe
from quantopian.pipeline.experimental import QTradableStocksUS

# Define our own custom factor class
class SentimentSevenDaysAgo(CustomFactor):
    inputs = [sentiment.sentiment_signal]
    window_length=7

    def compute(self, today, assets, out, sentiment):
        out[:] = sentiment[0] #When I specified a window _length of 7 it gave
 back last 7 scores thus call the senitment Boundcolumn from the top index


def make_pipeline():
    # Create a reference to our trading universe
    base_universe = QTradableStocksUS()

    # Get latest closing price
    close_price = USEquityPricing.close.latest

    daily_returns = DailyReturns(
        inputs = [USEquityPricing.close])

    returns_3 = Returns(
        inputs = [USEquityPricing.close],
        window_length = 2)


    mean_close_21 = SimpleMovingAverage(
        inputs = [USEquityPricing.close],
        window_length= 21)

    mean_close_252 = SimpleMovingAverage(
        inputs = [USEquityPricing.close],
        window_length= 252)

    MAD = mean_close_21 / mean_close_252

    sentdex_score = sentiment.sentiment_signal.latest

    sentdex_lag = SentimentSevenDaysAgo()

    marketCap = MarketCap()
```

```
    stock_class = SuperSector()

    not_zero_returns = (daily_returns != 0) & (returns_3 !=0)
    # Calculate 3 day average of bull_minus_bear scores
    sentiment_score = SimpleMovingAverage(
        inputs=[stocktwits.bull_minus_bear],
        window_length=2,
    )
    # Create filter for positive/negative spread moving averages
    # Create filter for positive sentiment scores
    # Crate filter for 7 days lag sentiment score
    # assets based on their sentiment scores
    positive_MAD = MAD > 1.2
    negative_MAD = MAD < 0.8

    positive_sentiment_lag = sentdex_lag > 3
    negative_sentiment_lag = sentdex_lag < -1

    positive_sentiment = sentdex_score > 2
    negative_sentiment = sentdex_score < -1

    positive_twits = sentiment_score > 0
    negative_twits = sentiment_score < 0
#     Long = sentdex_lag.top(10, mask = positive_MAD)
#     short = sentdex_lag.bottom(10, mask = negative_MAD)




    Long = (positive_MAD & positive_sentiment_lag & positive_twits & positive_
sentiment)
    short = (negative_MAD & negative_sentiment_lag & negative_twits )

    tradeable_equities = (Long | short)
    # Return Pipeline containing all below columns and
    # sentiment_score that has our trading universe as screen
    return Pipeline(
        columns={
            'close_price': close_price,
            "Sentdex": sentdex_score,
            "Sentdex_lag": sentdex_lag,
            'sentiment_score': sentiment_score.zscore(), #apply zscore to norm
alize
            "MAD": MAD,
            "BUY": Long,
            "SHORT": short,
            "return": daily_returns,
            "Returns": returns_3,
            "Market Capital.": marketCap,
            "Stock Classfiaction": stock_class
        },
        screen=(base_universe
        & tradeable_equities& sentdex_lag.notnull() & sentiment_score.notnull
() & not_zero_returns)
    )
```

In [3]:
```
# Import run_pipeline method
from quantopian.research import run_pipeline

# Specify a time range to evaluate
period_start = '2013-01-01 07:12:03.6'
period_end = '2019-01-01 07:12:03.6 '

# Execute pipeline created by make_pipeline
# between start_date and end_date
pipeline_output = run_pipeline(
    make_pipeline(),
    start_date=period_start,
    end_date=period_end
)
# pipeline_output.add(sentiment_free.sentiment_signal, 'sentiment_signal')

# Display last 10 rows
pipeline_output.tail(20)
# print('Number of securities that passed the filter: %d' % len(pipeline_outpu
t))
```

**Pipeline Execution Time:** 7 Minutes, 12.20 Seconds

Out[3]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentde |
|---|---|---|---|---|---|---|---|---|
| **2018-12-24 00:00:00+00:00** | **Equity(337 [AMAT])** | False | 0.743352 | 2.905535e+10 | -0.019417 | True | -3.0 | |
| | **Equity(4705 [MKC])** | True | 1.257794 | 1.818684e+10 | -0.007614 | False | 5.0 | |
| | **Equity(7447 [TIF])** | False | 0.790511 | 9.237367e+09 | -0.029694 | True | -3.0 | |
| **2018-12-26 00:00:00+00:00** | **Equity(4705 [MKC])** | True | 1.252156 | 1.767615e+10 | -0.029099 | False | 5.0 | |
| | **Equity(7447 [TIF])** | False | 0.779504 | 9.045988e+09 | -0.021633 | True | -3.0 | |
| **2018-12-27 00:00:00+00:00** | **Equity(337 [AMAT])** | False | 0.735128 | 2.937169e+10 | 0.057971 | True | -1.0 | |
| | **Equity(3149 [GE])** | False | 0.568228 | 6.427907e+10 | 0.066378 | True | -3.0 | |
| | **Equity(5029 [MRK])** | True | 1.207782 | 1.924279e+11 | 0.038607 | False | 6.0 | |
| | **Equity(13635 [DO])** | False | 0.653096 | 1.330368e+09 | 0.056831 | True | -3.0 | |
| | **Equity(39778 [QEP])** | False | 0.698206 | 1.337756e+09 | 0.108930 | True | 2.0 | |
| **2018-12-28 00:00:00+00:00** | **Equity(5029 [MRK])** | True | 1.206543 | 1.960164e+11 | 0.018654 | False | 6.0 | |
| | **Equity(9883 [ATVI])** | False | 0.690358 | 3.589391e+10 | 0.013569 | True | -3.0 | |
| | **Equity(39778 [QEP])** | False | 0.685779 | 1.361433e+09 | 0.015929 | True | 6.0 | |
| **2018-12-31 00:00:00+00:00** | **Equity(3149 [GE])** | False | 0.570777 | 6.532284e+10 | 0.034388 | True | -3.0 | |
| | **Equity(5029 [MRK])** | True | 1.204639 | 1.959904e+11 | 0.000133 | False | 6.0 | |
| | **Equity(39778 [QEP])** | False | 0.673540 | 1.325917e+09 | -0.025261 | True | 6.0 | |
| **2019-01-02 00:00:00+00:00** | **Equity(3149 [GE])** | False | 0.571820 | 6.584473e+10 | 0.005984 | True | -3.0 | |
| | **Equity(4705 [MKC])** | True | 1.234640 | 1.832881e+10 | 0.002015 | False | 5.0 | |
| | **Equity(5029 [MRK])** | True | 1.202823 | 1.986948e+11 | 0.014727 | False | 6.0 | |
| | **Equity(13635 [DO])** | False | 0.634035 | 1.297384e+09 | -0.031828 | True | -3.0 | |

In [4]:
```
pipeline_output.describe()
```

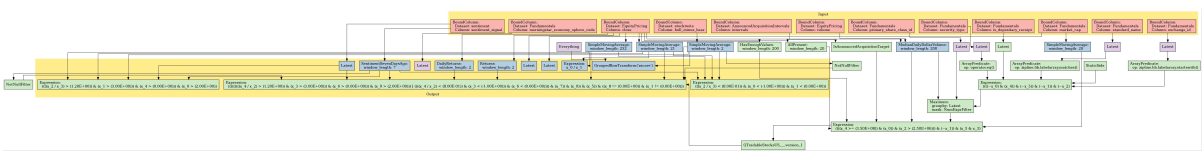
Out[4]:

| | MAD | Market Capital. | Returns | Sentdex | Sentdex_lag | Stock Classfiaction | close_\| |
|---|---|---|---|---|---|---|---|
| **count** | 7721.000000 | 7.721000e+03 | 7721.000000 | 7721.000000 | 7721.000000 | 7721.000000 | 7721.00 |
| **mean** | 1.169052 | 2.067004e+10 | 0.000524 | 3.778267 | 3.717394 | 2.036006 | 62.20 |
| **std** | 0.280565 | 3.440918e+10 | 0.026242 | 3.194239 | 3.488942 | 0.883938 | 95.30 |
| **min** | 0.212168 | 2.632199e+08 | -0.332248 | -3.000000 | -3.000000 | 1.000000 | 0.91 |
| **25%** | 1.204733 | 5.088438e+09 | -0.010000 | 4.000000 | 4.000000 | 1.000000 | 22.62 |
| **50%** | 1.239767 | 1.209133e+10 | 0.001032 | 5.000000 | 6.000000 | 2.000000 | 41.93 |
| **75%** | 1.302449 | 2.334478e+10 | 0.011609 | 6.000000 | 6.000000 | 3.000000 | 70.46 |
| **max** | 2.138074 | 7.399760e+11 | 0.528736 | 6.000000 | 6.000000 | 3.000000 | 1370.43 |

## Pipeline Schema to Fetch US Tradeable Equities

In [5]:
```
# pipeline_output['return'] = pd.get_dummies(pipeline_output['return'],drop_fi
rst=True)
# pipeline_output.head(30)
make_pipeline().show_graph(format='jpeg') #or png
```

Out[5]:



## Feature Engineering

In [ ]:
```

```

In [4]:
```
# return_encoding = pd.get_dummies(pipeline_output['return'],drop_first=True)
# pipeline_output['return'] = return_encoding
# pipeline_output['Returns'] = pipeline_output["Returns"].apply(np.sign)

pipeline_output['return'] = pipeline_output["return"].apply(np.sign)
pipeline_output['Market Capital.'] = pipeline_output["Market Capital."].apply(
np.log)
# Applied zscore to Market Capitalization but got an outlier which was Apple s
tock
```

In [5]:
```
pipeline_output.head(10)
```

Out[5]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_la |
|---|---|---|---|---|---|---|---|---|
| **2013-01-03 00:00:00+00:00** | **Equity(754 [BBY])** | False | 0.630091 | 22.111144 | -0.004219 | True | -3.0 | -3 |
| **2013-01-04 00:00:00+00:00** | **Equity(754 [BBY])** | False | 0.629542 | 22.111144 | 0.013559 | True | -1.0 | -3 |
| | **Equity(3645 [HOV])** | True | 1.890448 | 20.662923 | 0.010130 | False | 3.0 | 6 |
| **2013-01-07 00:00:00+00:00** | **Equity(754 [BBY])** | False | 0.631212 | 22.111144 | 0.011706 | True | -1.0 | -3 |
| | **Equity(3212 [GILD])** | True | 1.307798 | 24.744883 | 0.010538 | False | 6.0 | 6 |
| | **Equity(32902 [FSLR])** | True | 1.287577 | 21.712406 | -0.024695 | False | 6.0 | 4 |
| **2013-01-08 00:00:00+00:00** | **Equity(3212 [GILD])** | True | 1.306559 | 24.744883 | 0.014916 | False | 6.0 | 4 |
| | **Equity(3645 [HOV])** | True | 1.918963 | 20.662923 | -0.043353 | False | 3.0 | 6 |
| **2013-01-09 00:00:00+00:00** | **Equity(3212 [GILD])** | True | 1.306288 | 24.744883 | 0.005202 | False | 6.0 | 4 |
| **2013-01-10 00:00:00+00:00** | **Equity(14848 [AABA])** | True | 1.218424 | 23.823049 | -0.018293 | False | 3.0 | 6 |

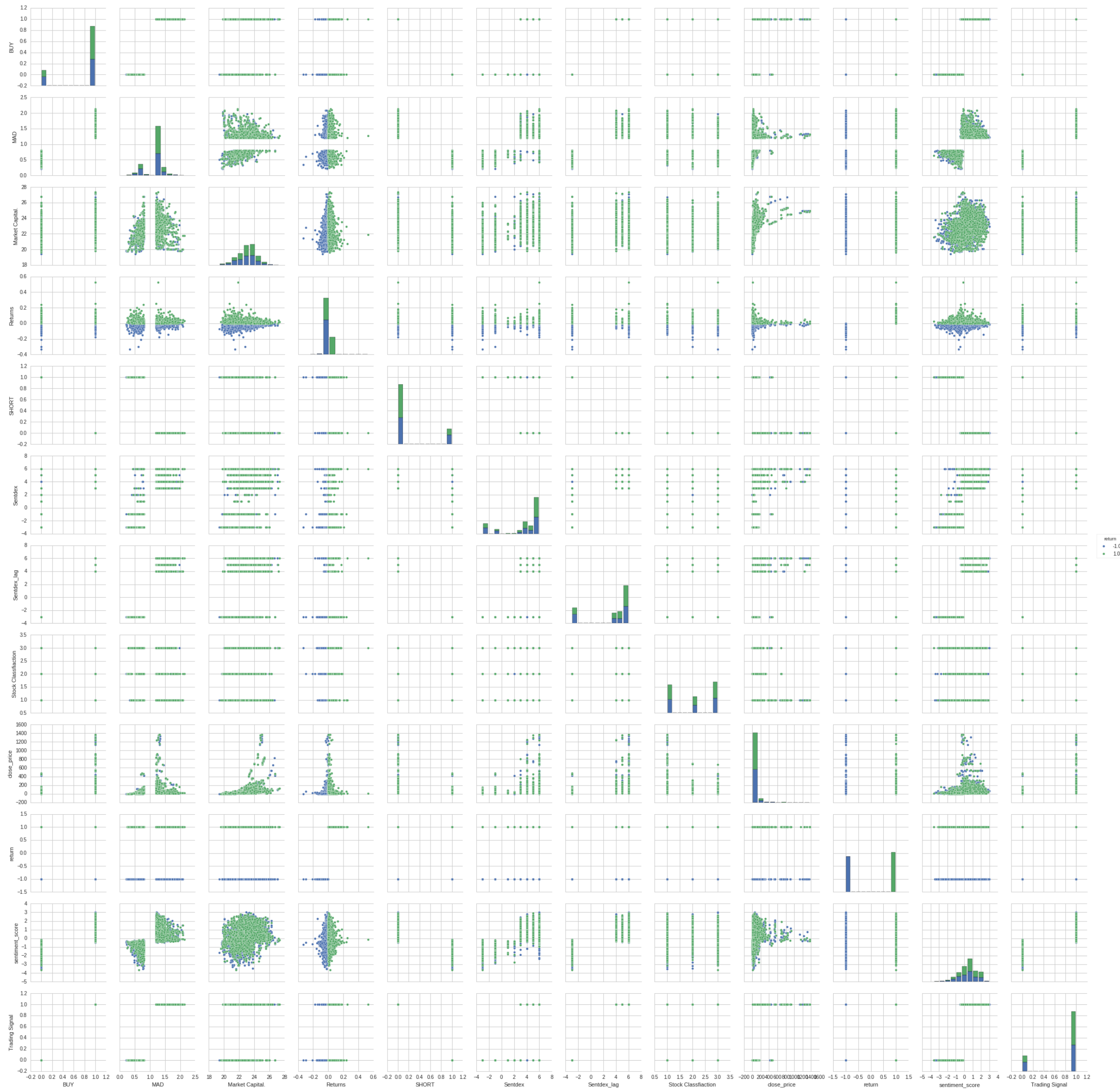
In [6]:
```
pipeline_output["Trading Signal"] = pd.get_dummies(pipeline_output['BUY'],drop
_first=True)

pipeline_output.tail(20)
# if BUY "FALSE" --> 0 | if BUY "TRUE" --> 1
```

Out[6]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_la |
|---|---|---|---|---|---|---|---|---|
| | Equity(337 [AMAT]) | False | 0.743352 | 24.092468 | -0.019417 | True | -3.0 | -3 |
| 2018-12-24 00:00:00+00:00 | Equity(4705 [MKC]) | True | 1.257794 | 23.623964 | -0.007614 | False | 5.0 | 5 |
| | Equity(7447 [TIF]) | False | 0.790511 | 22.946523 | -0.029694 | True | -3.0 | -3 |
| 2018-12-26 00:00:00+00:00 | Equity(4705 [MKC]) | True | 1.252156 | 23.595482 | -0.029099 | False | 5.0 | 5 |
| | Equity(7447 [TIF]) | False | 0.779504 | 22.925587 | -0.021633 | True | -3.0 | -3 |
| | Equity(337 [AMAT]) | False | 0.735128 | 24.103297 | 0.057971 | True | -1.0 | -3 |
| | Equity(3149 [GE]) | False | 0.568228 | 24.886500 | 0.066378 | True | -3.0 | -3 |
| 2018-12-27 00:00:00+00:00 | Equity(5029 [MRK]) | True | 1.207782 | 25.982987 | 0.038607 | False | 6.0 | 6 |
| | Equity(13635 [DO]) | False | 0.653096 | 21.008721 | 0.056831 | True | -3.0 | -3 |
| | Equity(39778 [QEP]) | False | 0.698206 | 21.014259 | 0.108930 | True | 2.0 | -3 |
| | Equity(5029 [MRK]) | True | 1.206543 | 26.001464 | 0.018654 | False | 6.0 | 6 |
| 2018-12-28 00:00:00+00:00 | Equity(9883 [ATVI]) | False | 0.690358 | 24.303833 | 0.013569 | True | -3.0 | -3 |
| | Equity(39778 [QEP]) | False | 0.685779 | 21.031804 | 0.015929 | True | 6.0 | -3 |
| | Equity(3149 [GE]) | False | 0.570777 | 24.902608 | 0.034388 | True | -3.0 | -3 |
| 2018-12-31 00:00:00+00:00 | Equity(5029 [MRK]) | True | 1.204639 | 26.001331 | 0.000133 | False | 6.0 | 6 |
| | Equity(39778 [QEP]) | False | 0.673540 | 21.005370 | -0.025261 | True | 6.0 | -3 |
| | Equity(3149 [GE]) | False | 0.571820 | 24.910565 | 0.005984 | True | -3.0 | -3 |
| | Equity(4705 [MKC]) | True | 1.234640 | 23.631740 | 0.002015 | False | 5.0 | 5 |
| 2019-01-02 00:00:00+00:00 | Equity(5029 [MRK]) | True | 1.202823 | 26.015036 | 0.014727 | False | 6.0 | 6 |
| | Equity(13635 [DO]) | False | 0.634035 | 20.983615 | -0.031828 | True | -3.0 | -3 |

In [14]:
```
sns.pairplot(pipeline_output,hue='return')
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x7f1772eb93d0>



# Data Analysis and Insights Generation

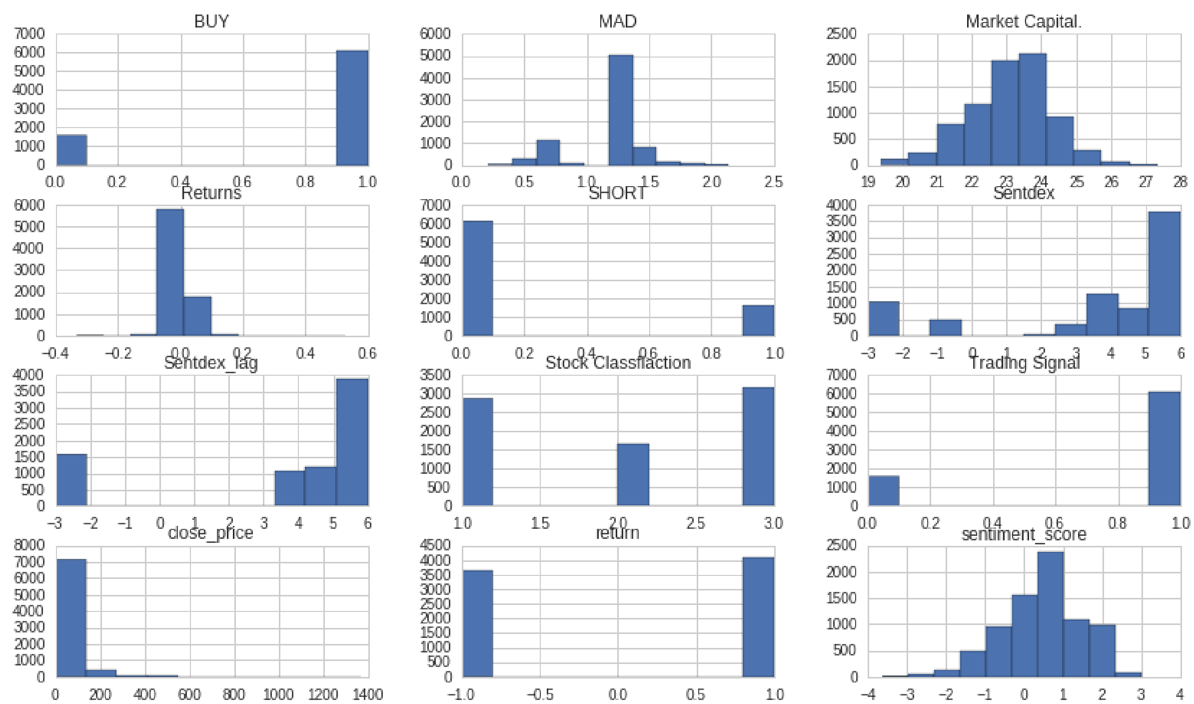
In [9]:
```
n=float(len(pipeline_output[pipeline_output["return"]>0]))
m=float(len(pipeline_output[pipeline_output["Trading Signal"]==1]))
a=float(len(pipeline_output[pipeline_output["return"]<0]))
b=float(len(pipeline_output[pipeline_output["Trading Signal"]==0]))
z=float(len(pipeline_output))
print"The percentage of positive returns is:", ((n/z)*100),"%"
print"The percentage of BUY Trading Signal is:", ((m/z)*100),"%"
print"The percentage of negative returns is:", ((a/z)*100),"%"
print"The percentage of SELL Trading Signal is:", ((b/z)*100),"%"
```

```
The percentage of positive returns is: 52.7910892371 %
The percentage of BUY Trading Signal is: 79.406812589 %
The percentage of negative returns is: 47.2089107629 %
The percentage of SELL Trading Signal is: 20.593187411 %
```

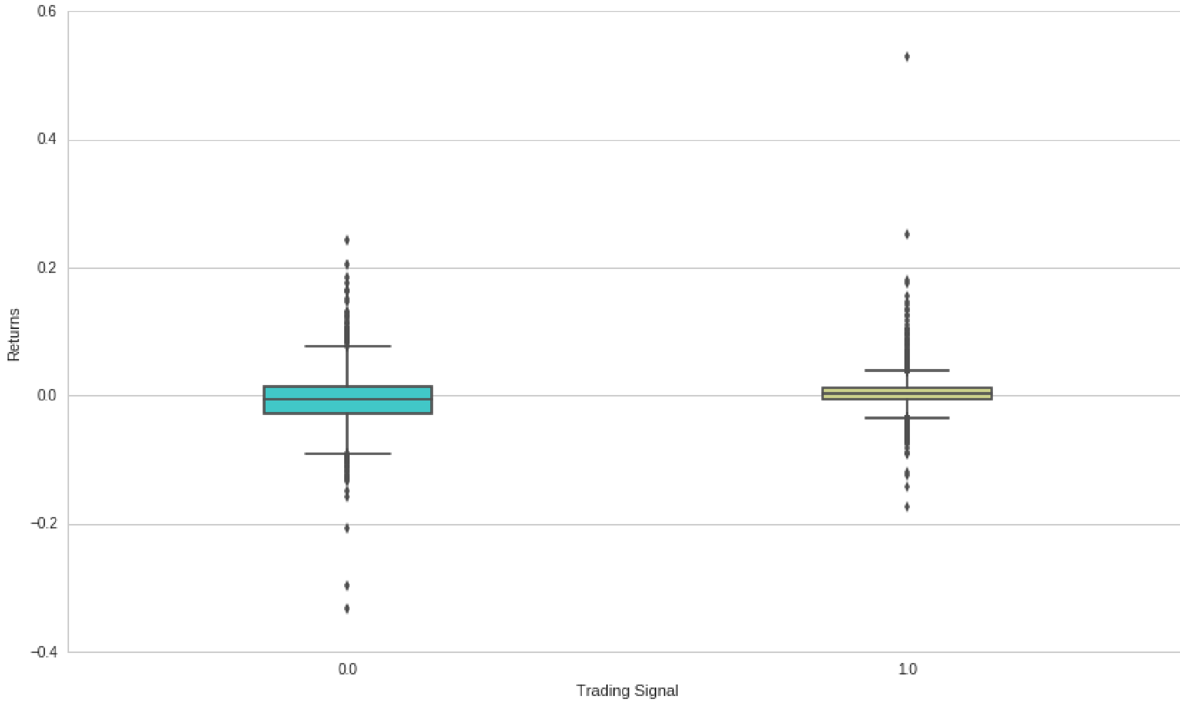
In [10]:
```
print(pipeline_output['Trading Signal'].value_counts())
print(pipeline_output['return'].value_counts())

#Unbalanced labels and datasets
```

```
1.0    6131
0.0    1590
Name: Trading Signal, dtype: int64
 1.0    4076
-1.0    3645
Name: return, dtype: int64
```

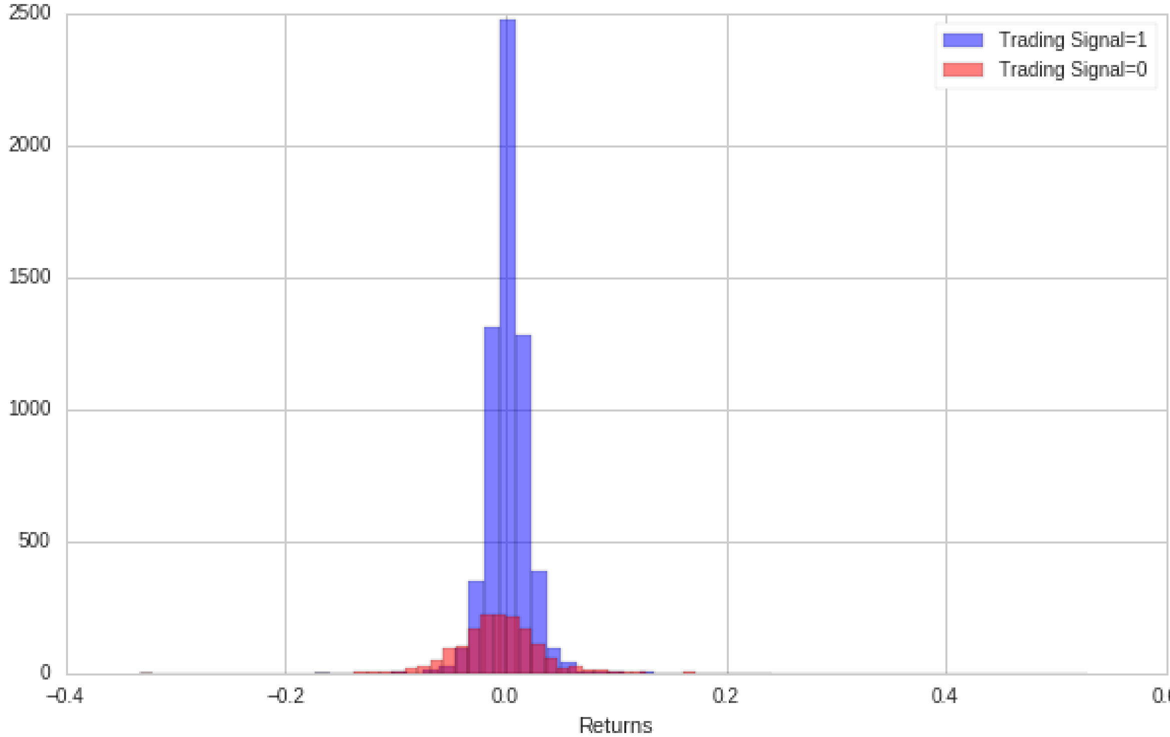
In [11]:
```
pipeline_output.hist();
```

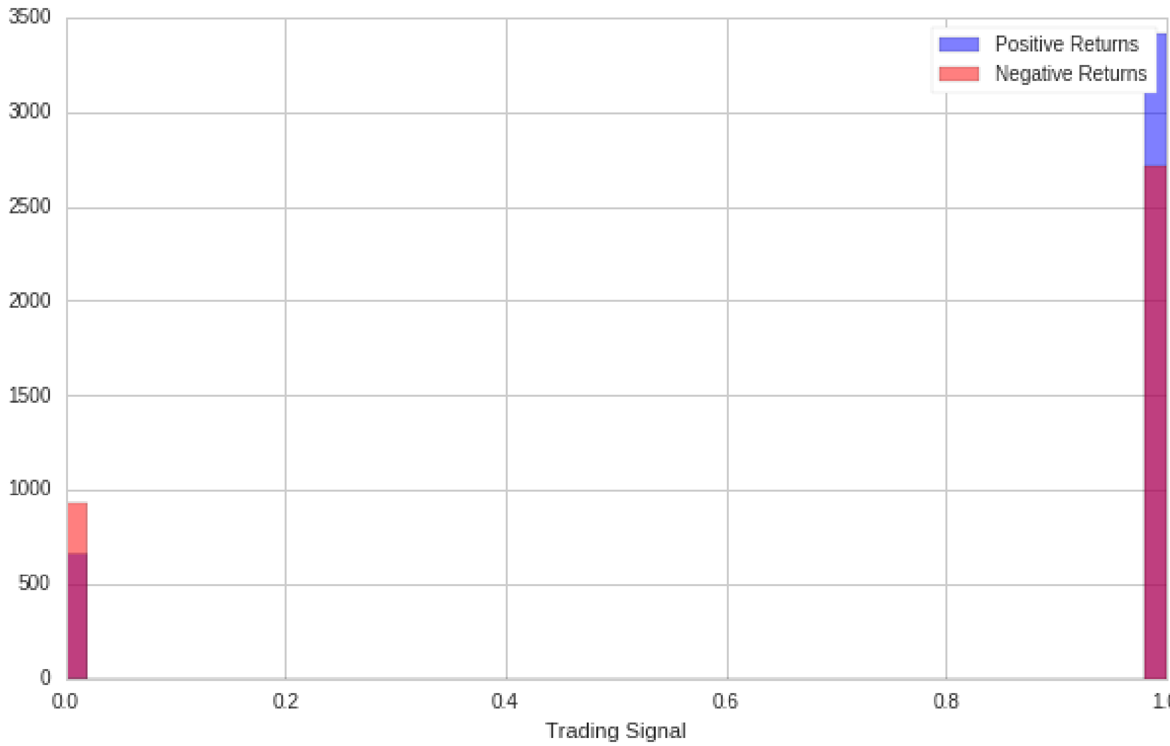


In [12]:
```
sns.boxplot(x='Trading Signal',y='Returns',data=pipeline_output,palette='rainb
ow', width= 0.3);
# For which I assigned buy (1), what where their returns
# For which I assigned sell(0), what where their returns
```

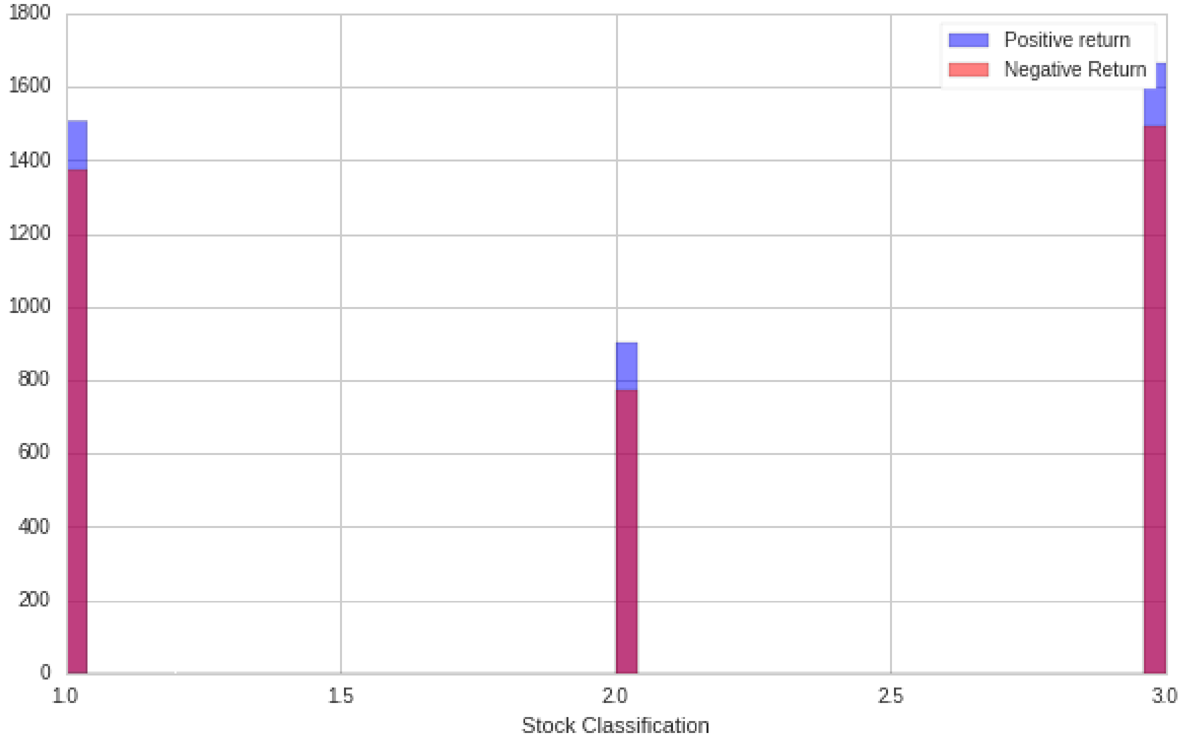


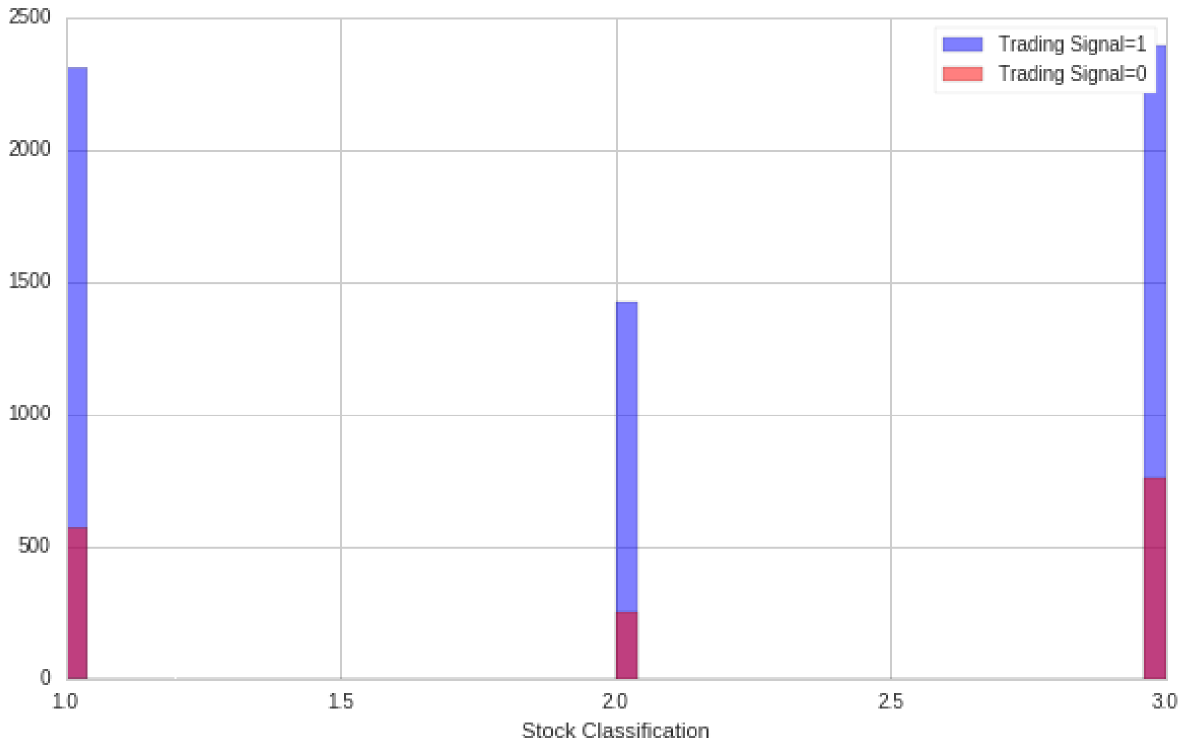
In [13]:
```
plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Trading Signal']==1]["Returns"].hist(alpha=0.
5,color='blue',
                                              bins=50,label='Trading Signal=1'
)
pipeline_output[pipeline_output['Trading Signal']==0]['Returns'].hist(alpha=0.
5,color='red',
                                              bins=50,label='Trading Signal=0'
)
plt.legend()
plt.xlabel('Returns');

plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Returns']>0]["Trading Signal"].hist(alpha=0.5
,color='blue',
                                              bins=50,label='Positive Returns'
)
pipeline_output[pipeline_output['Returns']<0]['Trading Signal'].hist(alpha=0.5
,color='red',
                                              bins=50,label='Negative Returns'
)
plt.legend()
plt.xlabel('Trading Signal');
```


Trading Signal=1
Trading Signal=0
2500
2000
1500
1000
500
0
−0.4
−0.2
0.0
0.2
0.4
0.6
Returns

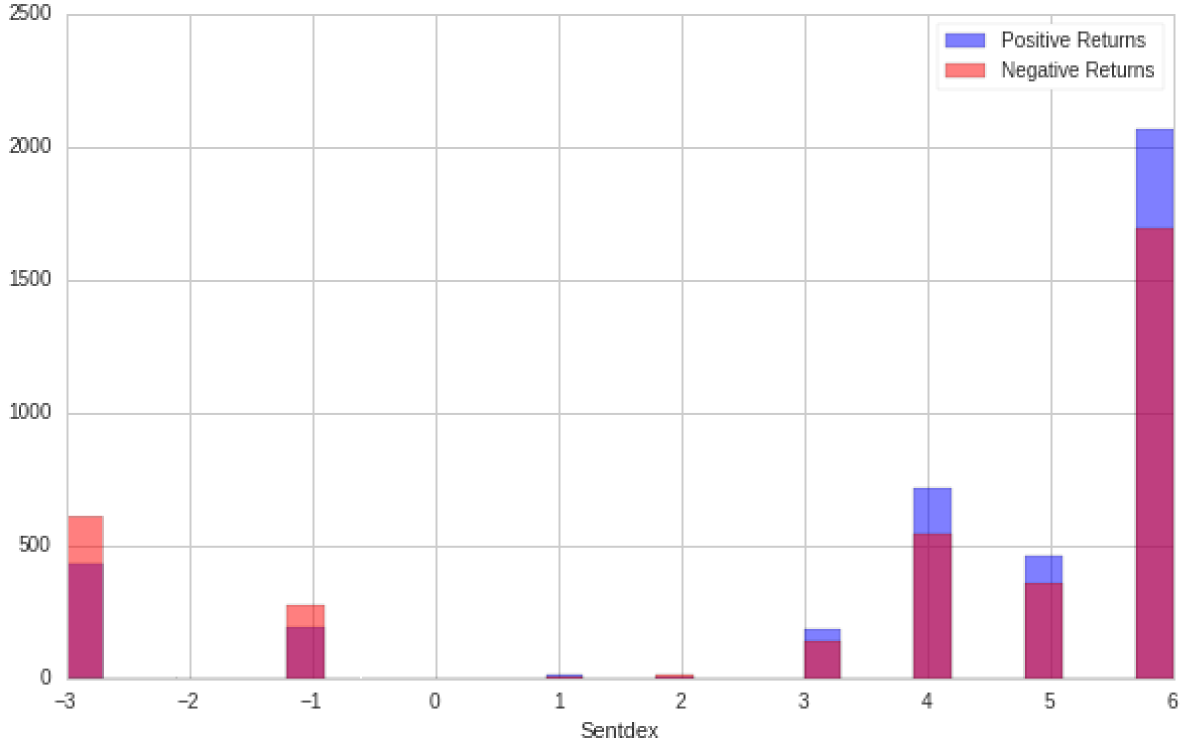

Positive Returns
Negative Returns
3500
3000
2500
2000
1500
1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0
Trading Signal

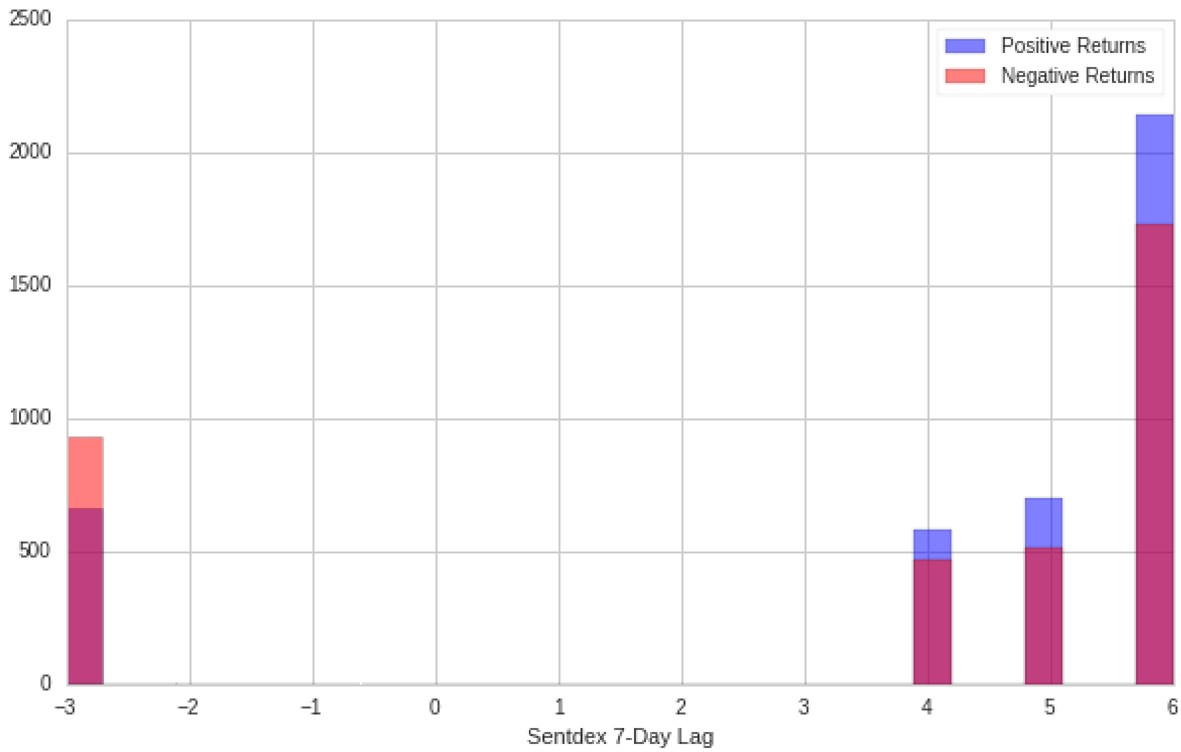
In [14]:
```
plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Returns']>0]['Stock Classfiaction'].hist(alph
a=0.5,color='blue',
                                              bins=50,label='Positive return')
pipeline_output[pipeline_output['Returns']<0]['Stock Classfiaction'].hist(alph
a=0.5,color='red',
                                              bins=50,label='Negative Return')
plt.legend()
plt.xlabel('Stock Classification');

plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Trading Signal']==1]['Stock Classfiaction'].h
ist(alpha=0.5,color='blue',
                                              bins=50,label='Trading Signal=1'
)
pipeline_output[pipeline_output['Trading Signal']==0]['Stock Classfiaction'].h
ist(alpha=0.5,color='red',
                                              bins=50,label='Trading Signal=0'
)
plt.legend()
plt.xlabel('Stock Classification');
```

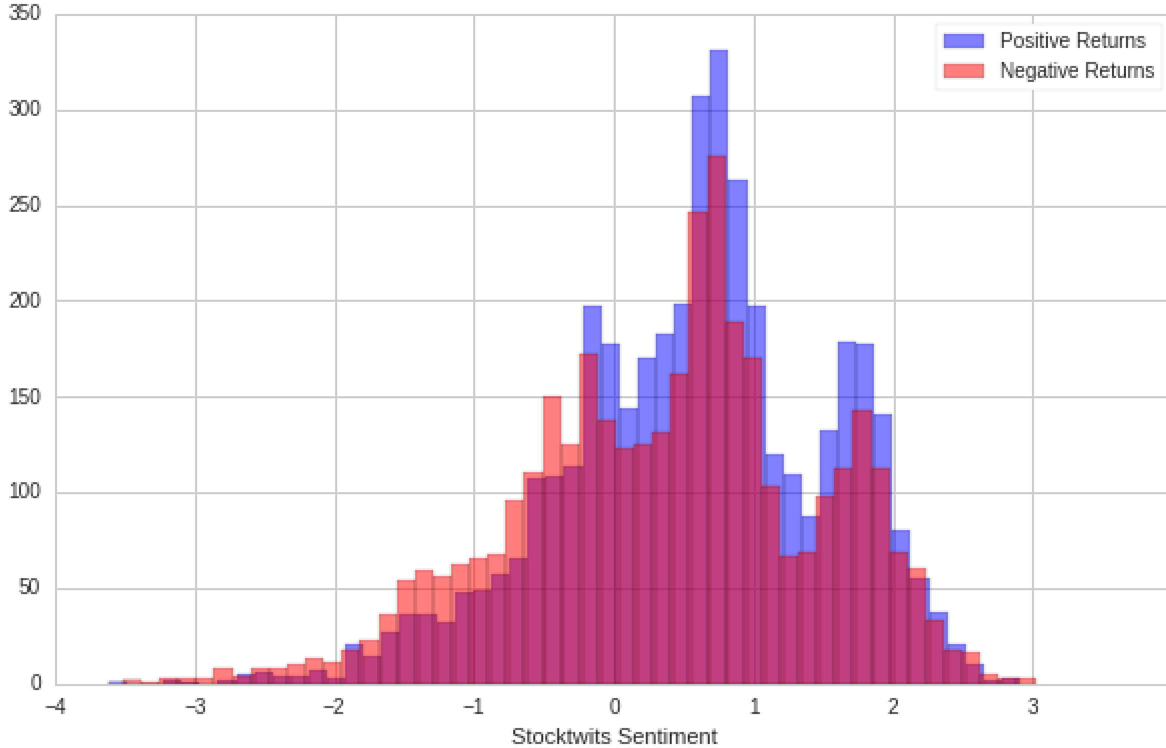

Positive return
Negative Return
1800
1600
1400
1200
1000
800
600
400
200
0
1.0
1.5
2.0
2.5
3.0
Stock Classification


Trading Signal=1
Trading Signal=0
2500
2000
1500
1000
500
0
1.0
1.5
2.0
2.5
3.0
Stock Classification

In [15]:
```
plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Returns']>0]['Sentdex'].hist(alpha=0.5,color=
'blue',
                                              bins=30,label='Positive Returns'
)
pipeline_output[pipeline_output['Returns']<0]['Sentdex'].hist(alpha=0.5,color=
'red',
                                              bins=30,label='Negative Returns'
)
plt.legend()
plt.xlabel('Sentdex');

plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Returns']>0]['Sentdex_lag'].hist(alpha=0.5,co
lor='blue',
                                              bins=30,label='Positive Returns'
)
pipeline_output[pipeline_output['Returns']<0]['Sentdex_lag'].hist(alpha=0.5,co
lor='red',
                                              bins=30,label='Negative Returns'
)
plt.legend()
plt.xlabel('Sentdex 7-Day Lag');

plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Returns']>0]['sentiment_score'].hist(alpha=0.
5,color='blue',
                                              bins=50,label='Positive Returns'
)
pipeline_output[pipeline_output['Returns']<0]['sentiment_score'].hist(alpha=0.
5,color='red',
                                              bins=50,label='Negative Returns'
)
plt.legend()
plt.xlabel('Stocktwits Sentiment');
```

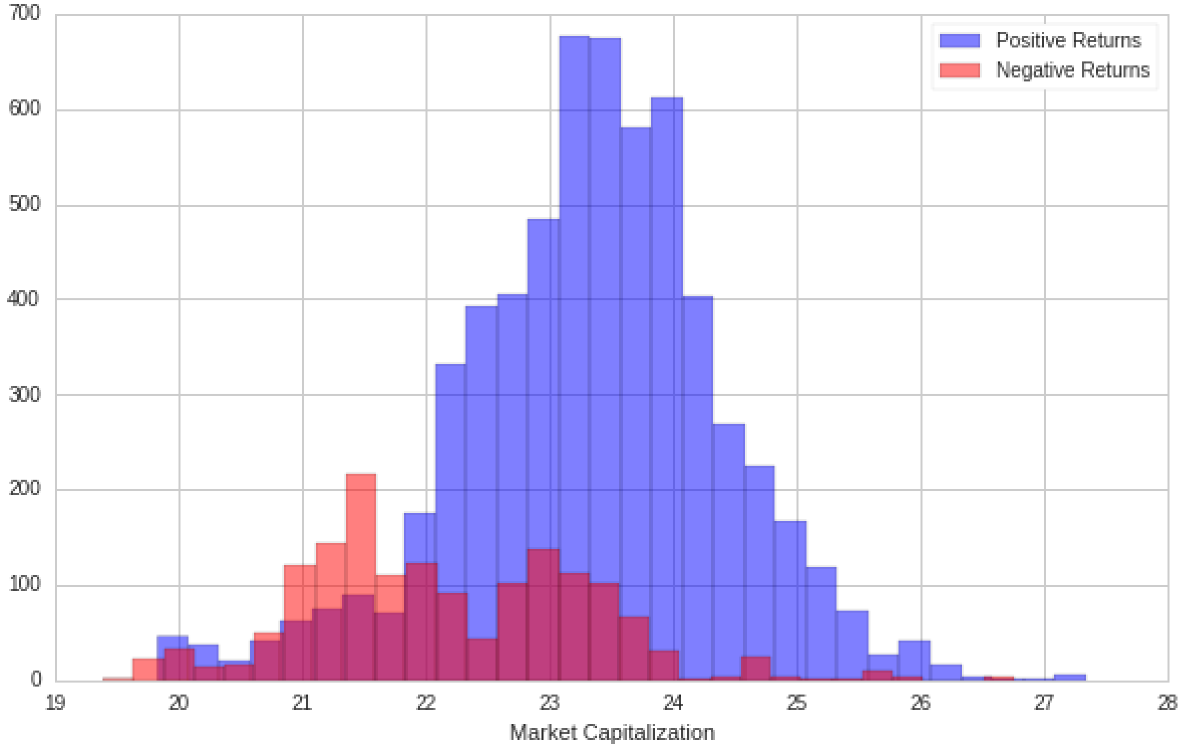

Positive Returns
Negative Returns
Sentdex

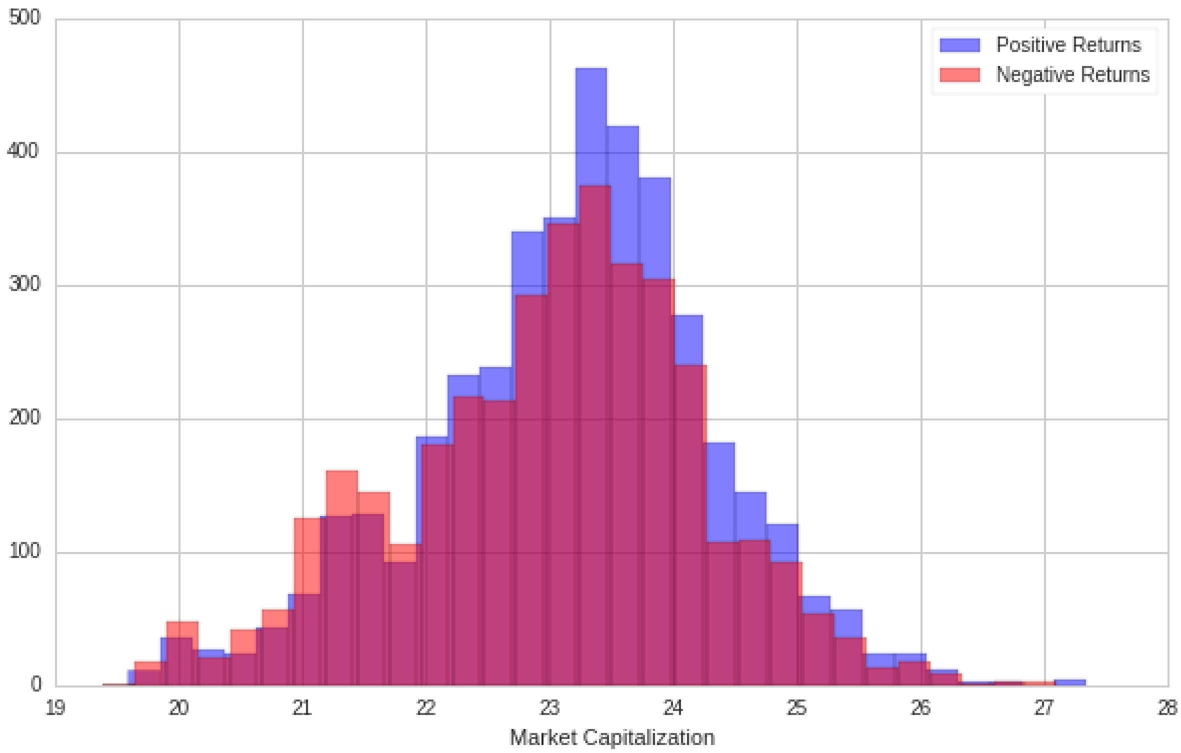

Positive Returns
Negative Returns
Sentdex 7-Day Lag



Positive Returns
Negative Returns
350
300
250
200
150
100
50
0
−4
−3
−2
−1
0
1
2
3
4
Stocktwits Sentiment

In [16]:
```
plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Trading Signal']==1]['Market Capital.'].hist(
alpha=0.5,color='blue',
                                              bins=30,label='Positive Returns'
)
pipeline_output[pipeline_output['Trading Signal']==0]['Market Capital.'].hist(
alpha=0.5,color='red',
                                              bins=30,label='Negative Returns'
)
plt.legend()
plt.xlabel('Market Capitalization');

plt.figure(figsize=(10,6))
pipeline_output[pipeline_output['Returns']>0]['Market Capital.'].hist(alpha=0.
5,color='blue',
                                              bins=30,label='Positive Returns'
)
pipeline_output[pipeline_output['Returns']<0]['Market Capital.'].hist(alpha=0.
5,color='red',
                                              bins=30,label='Negative Returns'
)
plt.legend()
plt.xlabel('Market Capitalization');
```


700
600
500
400
300
200
100
0
19
20
21
22
23
24
25
26
27
28
Market Capitalization
Positive Returns
Negative Returns


500
400
300
200
100
0
19
20
21
22
23
24
25
26
27
28
Market Capitalization
Positive Returns
Negative Returns

In [17]:
```
print('Number of securities that passed the filter: %d' % len(pipeline_output.
index.levels[1].unique()))
pipeline_output.columns
```

```
Number of securities that passed the filter: 365
```

Out[17]:
```
Index([u'BUY', u'MAD', u'Market Capital.', u'Returns', u'SHORT', u'Sentdex',
       u'Sentdex_lag', u'Stock Classfiaction', u'close_price', u'return',
       u'sentiment_score', u'Trading Signal'],
      dtype='object')
```

In [18]:
```
pipeline_output.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 7721 entries, (2013-01-03 00:00:00+00:00, Equity(754 [BBY])) to
(2017-01-03 00:00:00+00:00, Equity(34913 [RF]))
Data columns (total 12 columns):
BUY                    7721 non-null bool
MAD                    7721 non-null float64
Market Capital.        7721 non-null float64
Returns                7721 non-null float64
SHORT                  7721 non-null bool
Sentdex                7721 non-null float64
Sentdex_lag            7721 non-null float64
Stock Classfiaction    7721 non-null int64
close_price            7721 non-null float64
return                 7721 non-null float64
sentiment_score        7721 non-null float64
Trading Signal         7721 non-null float64
dtypes: bool(2), float64(9), int64(1)
memory usage: 678.6+ KB
```

# **BUILDING THE MODEL PHASE**

In [7]:
```
from sklearn.linear_model import LogisticRegression
# Testing how the model's intuition
model = LogisticRegression()
model.fit([[-2,-3],[1,0],[1,1]],["T","F","T"])
# model.coef_, model.intercept_,model.predict([-3,21])
```

Out[7]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0)
```

In [8]:
```
feature_cols = ["Market Capital.", "sentiment_score", "Sentdex_lag","MAD","Tra
ding Signal", "Stock Classfiaction", "Sentdex"]
X = pipeline_output[feature_cols]
y = pipeline_output["return"] #target column (classified T or F)
```

In [9]:
```
# split X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=
None)
```

In [10]:
```
# instantiate the model (using the default parameters)
logmodel = LogisticRegression()
# fit the model with data
logmodel.fit(X_train,y_train)
```

Out[10]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0)
```

In [11]:
```
#
y_pred=logmodel.predict(X_test)
y_pred
logmodel.coef_, logmodel.intercept_,logmodel.predict(X_test)
```

Out[11]:
```
(array([[ 0.05054444,  0.08148023,  0.00740886,  0.18409867,  0.28884548,
          0.00258946, -0.02378607]]),
 array([-1.46557072]),
 array([ 1.,  1., -1., ...,  1.,  1.,  1.]))
```

In [12]:
```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[12]:
```
array([[ 416, 1503],
       [ 297, 1885]])
```

In [13]:
```
print"Accuracy:",metrics.accuracy_score(y_test, y_pred)
print"Precision:",metrics.precision_score(y_test, y_pred)
print"Recall:",metrics.recall_score(y_test, y_pred)
```

```
Accuracy: 0.561082662765
Precision: 0.556375442739
Recall: 0.863886342805
```

In [14]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

       -1.0       0.58      0.22      0.32      1919
        1.0       0.56      0.86      0.68      2182

avg / total       0.57      0.56      0.51      4101
```

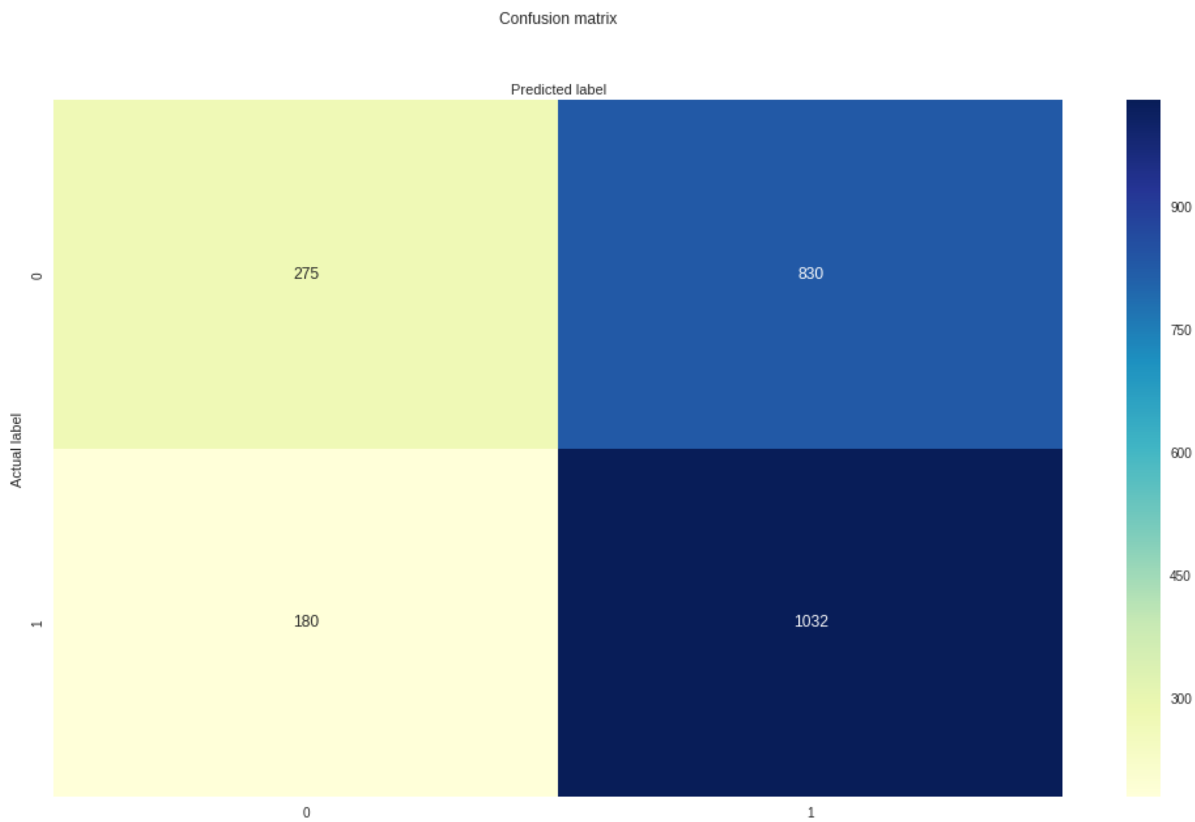
# Determine Threshold

In [23]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, ro
c_auc_score, precision_score

THRESHOLD = 0.50
preds = np.where(logmodel.predict_proba(X_test)[:,1] > THRESHOLD, 1, -1)

pd.DataFrame(data=[accuracy_score(y_test, preds), recall_score(y_test, preds),
                   precision_score(y_test, preds), roc_auc_score(y_test, preds
)],
             index=["accuracy", "recall", "precision", "roc_auc_score"], colum
ns = ["Scores"])
# 0.5 remained the best threshold
```

Out[23]:

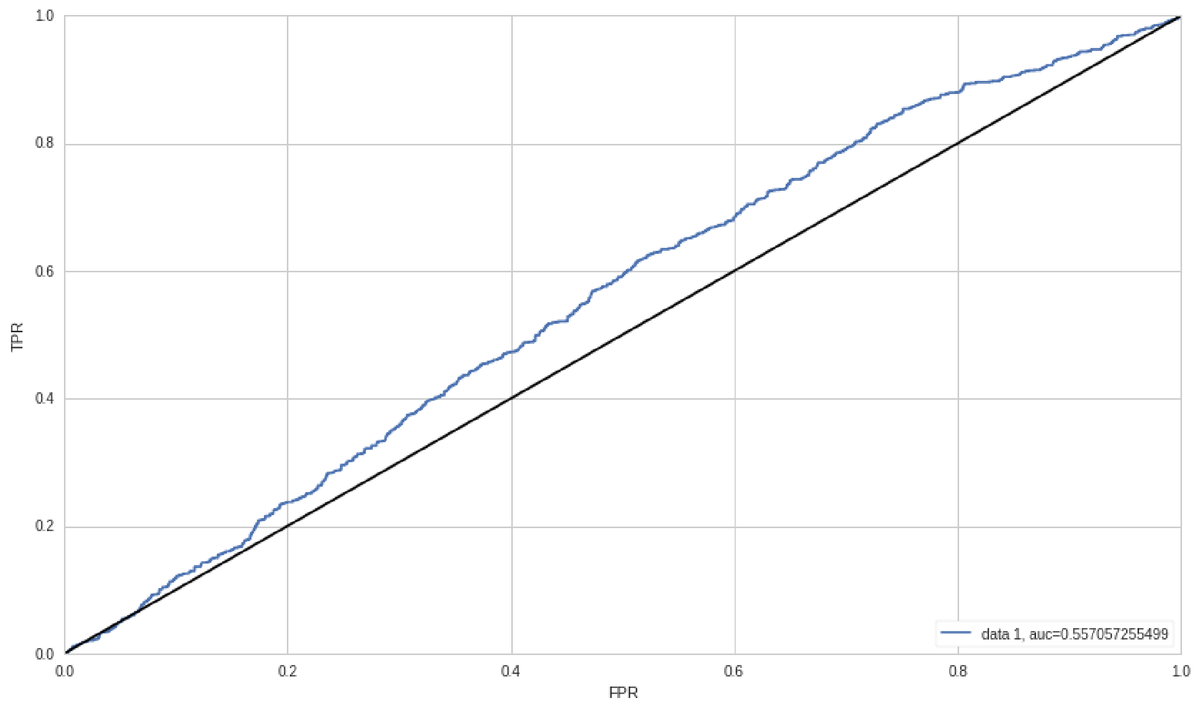
| | Scores |
|---|---|
| **accuracy** | 0.564091 |
| **recall** | 0.851485 |
| **precision** | 0.554243 |
| **roc_auc_score** | 0.550177 |

In [24]:
```
class_names=[-1,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label');
```




AUROC represents the likelihood of the model distinguishing observations from two classes. In other words, if a random selection of an observation from each class is made, what's the probability that your model will be able to "rank" them correctly?

In [25]:
```
y_pred_proba = logmodel.predict_proba(X_test)[::,1]
fpr, tpr, threshold = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
x = np.linspace(0, 1, 100000)
plt.plot(x, x + 0, linestyle='solid',c = 'k')
plt.legend(loc=4)
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```




## Random Forest Classifier

In [15]:
```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=800, max_leaf_nodes=2,max_features=7
, min_samples_split=2)
rfc.fit(X_train, y_train)
```

Out[15]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=7, max_leaf_nodes=2,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=800, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [16]:
```
from sklearn import metrics
rfc_pred = rfc.predict(X_test)
rfc_pred
cnf_matrix = metrics.confusion_matrix(y_test, rfc_pred)
cnf_matrix
```

Out[16]:
```
array([[ 423, 1496],
       [ 314, 1868]])
```

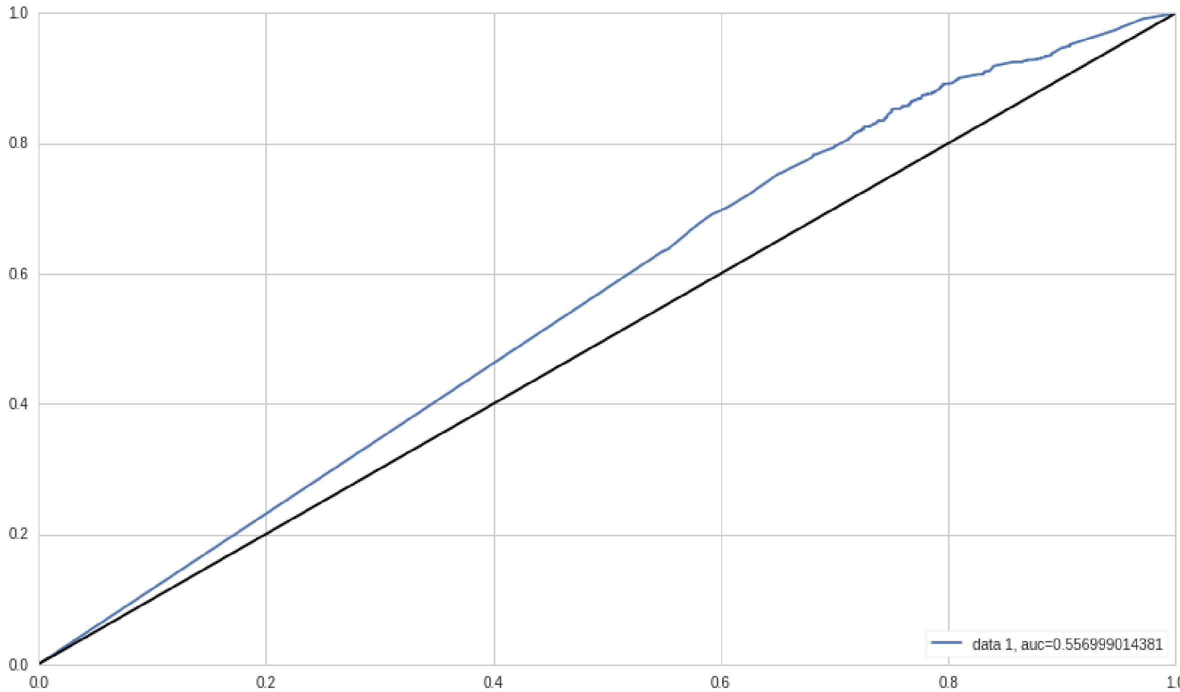
In [17]:
```
print("Accuracy:",metrics.accuracy_score(y_test, rfc_pred))
print("Precision:",metrics.precision_score(y_test, rfc_pred))
print("Recall:",metrics.recall_score(y_test, rfc_pred))
```

```
('Accuracy:', 0.57449402584735432)
('Precision:', 0.57298731937481573)
('Recall:', 0.86741071428571426)
```

In [18]:
```
print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

       -1.0       0.58      0.22      0.32      1861
        1.0       0.57      0.87      0.69      2240

avg / total       0.58      0.57      0.52      4101
```

In [41]:
```
y_pred_proba = rfc.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
x = np.linspace(0, 1, 100000)
plt.plot(x, x + 0, linestyle='solid',c = 'k')
plt.legend(loc=4)
plt.show()
```




# Support Vector Machine (SVMs)

In [17]:
```
from sklearn.svm import SVC
sv_model = SVC(C=1,gamma=0.1)
sv_model.fit(X_train,y_train)
```

Out[17]:
```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.1,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [22]:
```
sv_predictions = sv_model.predict(X_test)
print(metrics.confusion_matrix(y_test,sv_predictions))
print("\n")
print(classification_report(y_test,sv_predictions))
```

```
[[ 411 1539]
 [ 306 1845]]


             precision    recall  f1-score   support

       -1.0       0.57      0.21      0.31      1950
        1.0       0.55      0.86      0.67      2151

avg / total       0.56      0.55      0.50      4101
```

In [22]:
```
#### GRIDSEARCH ####
#Hypertuning parameters
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001]}
from sklearn.grid_search import GridSearchCV
```

In [23]:
```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=0)
```

In [24]:
```
grid.fit(X_train,y_train)
```

Out[24]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, deg
ree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'C': [0.1, 1, 10, 100, 1000], 'gamma': [1, 0.1, 0.01, 0.00
1, 0.0001]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring=None,
       verbose=0)
```

In [25]:
```
print(grid.best_params_)
grid.best_estimator_
```

```
{'C': 1, 'gamma': 0.1}
```

Out[25]:
```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.1,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [26]:
```
grid_predictions = grid.predict(X_test)
print(metrics.confusion_matrix(y_test,grid_predictions))
print("\n")
print(classification_report(y_test,grid_predictions))
```

```
[[ 298  769]
 [ 229 1021]]


             precision    recall  f1-score   support

       -1.0       0.57      0.28      0.37      1067
        1.0       0.57      0.82      0.67      1250

avg / total       0.57      0.57      0.53      2317
```

# K-Nearest Neighbor

***Run it after Pipeliene directly***

In [51]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [52]:
```
scaler.fit(pipeline_output.drop(['close_price', 'BUY', 'return', 'Returns', 'S
HORT'],axis=1))
```

Out[52]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [53]:
```
scaled_features = scaler.transform(pipeline_output.drop(['close_price', 'BUY',
'return', 'Returns', 'SHORT'],axis=1))
```

In [54]:
```
scaled_features
```

Out[54]:
```
array([[-1.92110916, -0.8276163 , -2.12216652, ..., -1.17211062,
        -1.06454103, -1.96366363],
       [-1.92306567, -0.8276163 , -1.49599865, ..., -1.17211062,
        -1.06001677, -1.96366363],
       [ 2.57138894, -2.04815612, -0.24366291, ..., -1.17211062,
        -0.53195792,  0.50925219],
       ...,
       [ 0.3088213 ,  1.44250233,  0.69558889, ..., -1.17211062,
         0.76959902,  0.50925219],
       [ 0.42621986,  1.25449715,  0.06942102, ...,  1.09063874,
         0.22452827,  0.50925219],
       [ 1.15531608,  0.42339293,  0.38250495, ..., -1.17211062,
         0.50079701,  0.50925219]])
```

In [56]:
```
pipeline_scaled = pd.DataFrame(scaled_features, columns=pipeline_output.column
s.drop(['close_price', 'BUY', 'return', 'Returns', 'SHORT']))
```

In [57]:
```
pipeline_scaled.head()
```

Out[57]:

| | MAD | Market Capital. | Sentdex | Sentdex_lag | Stock Classfiaction | sentiment_score | Trading Signal |
|---|---|---|---|---|---|---|---|
| **0** | -1.921109 | -0.827616 | -2.122167 | -1.925463 | -1.172111 | -1.064541 | -1.963664 |
| **1** | -1.923066 | -0.827616 | -1.495999 | -1.925463 | -1.172111 | -1.060017 | -1.963664 |
| **2** | 2.571389 | -2.048156 | -0.243663 | 0.654283 | -1.172111 | -0.531958 | 0.509252 |
| **3** | -1.917113 | -0.827616 | -1.495999 | -1.925463 | -1.172111 | -1.825834 | -1.963664 |
| **4** | 0.494555 | 1.392060 | 0.695589 | 0.654283 | -0.040736 | 0.024551 | 0.509252 |

In [49]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features,pipeline_o
utput['return'],
                                                    test_size=0.30)
# Get dummies for returns first before running
```

In [70]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=36)
```

In [71]:
```
knn.fit(X_train,y_train)
```

Out[71]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_neighbors=36, p=2, weights='uniform')
```

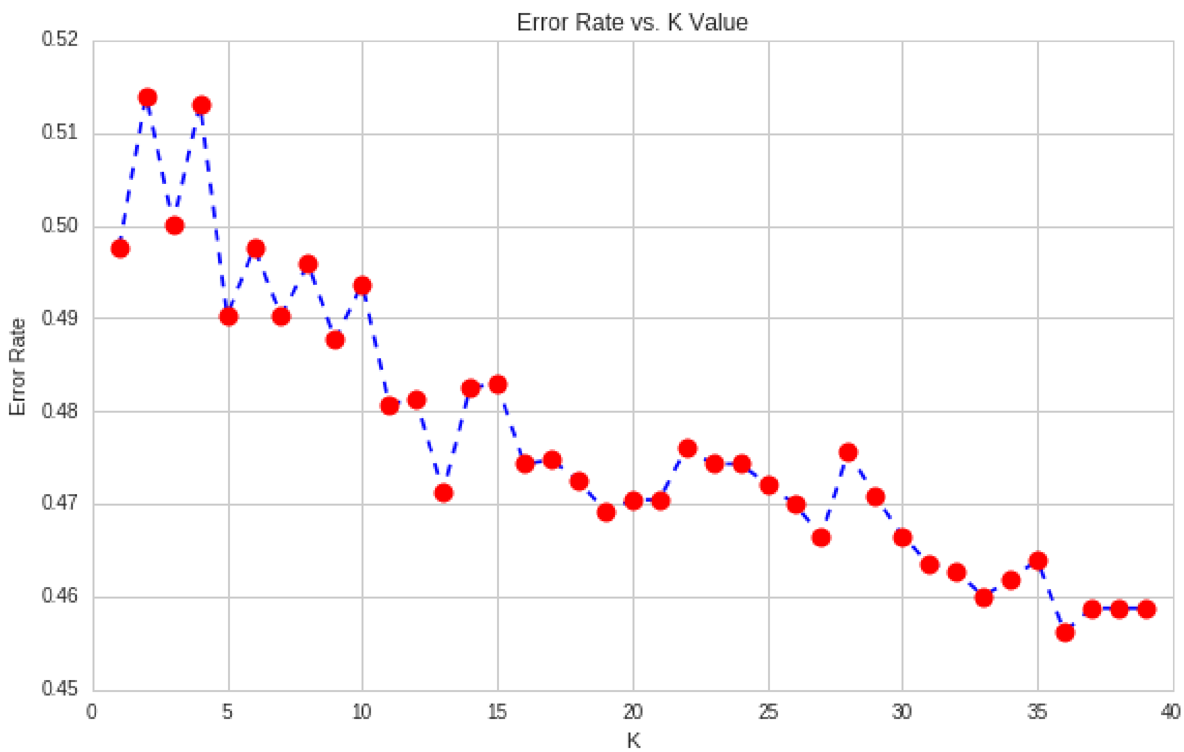
In [72]:
```
pred = knn.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_test,pred))
print("\n")
print(classification_report(y_test,pred))
```

```
[[503 602]
 [455 757]]


             precision    recall  f1-score   support

       -1.0       0.53      0.46      0.49      1105
        1.0       0.56      0.62      0.59      1212

avg / total       0.54      0.54      0.54      2317
```

In [62]:
```
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [63]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Out[63]: <matplotlib.text.Text at 0x7f1754058790>




In [ ]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
predictions = lm.predict(X_test)
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

## Out of Sample Predictions:

In [74]:
```
pipeline_outsample = run_pipeline(
    make_pipeline(),
    start_date="2019-01-01",
    end_date="2019-06-21"
)
pipeline_outsample.tail(20)
```

**Pipeline Execution Time:** 2 Minutes, 18.30 Seconds

Out[74]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentde |
|---|---|---|---|---|---|---|---|---|
| 2019-01-02 00:00:00+00:00 | Equity(3149 [GE]) | False | 0.571820 | 6.584473e+10 | 0.005984 | True | -3.0 | |
| | Equity(4705 [MKC]) | True | 1.234640 | 1.832881e+10 | 0.002015 | False | 5.0 | |
| | Equity(5029 [MRK]) | True | 1.202823 | 1.986948e+11 | 0.014727 | False | 6.0 | |
| | Equity(13635 [DO]) | False | 0.634035 | 1.297384e+09 | -0.031828 | True | -3.0 | |
| 2019-01-03 00:00:00+00:00 | Equity(337 [AMAT]) | False | 0.723636 | 3.209413e+10 | 0.022283 | True | 2.0 | |
| | Equity(7447 [TIF]) | False | 0.742030 | 9.974845e+09 | 0.016522 | True | -1.0 | |
| | Equity(13635 [DO]) | False | 0.626310 | 1.338614e+09 | 0.032874 | True | -3.0 | |
| | Equity(26143 [NRG]) | True | 1.206547 | 1.104923e+10 | -0.037626 | False | 6.0 | |
| | Equity(51649 [ADT]) | False | 0.783810 | 4.915695e+09 | 0.064007 | True | -3.0 | |
| 2019-01-04 00:00:00+00:00 | Equity(337 [AMAT]) | False | 0.718949 | 3.023444e+10 | -0.058525 | True | 6.0 | |
| | Equity(42950 [FB]) | False | 0.799401 | 3.785928e+11 | -0.026828 | True | -3.0 | |
| 2019-01-07 00:00:00+00:00 | Equity(4705 [MKC]) | True | 1.213979 | 1.798623e+10 | 0.002936 | False | 5.0 | |
| | Equity(9883 [ATVI]) | False | 0.681487 | 3.599310e+10 | 0.040132 | True | -1.0 | |
| | Equity(12909 [LH]) | False | 0.777963 | 1.291116e+10 | 0.034196 | True | 4.0 | |
| | Equity(23269 [WW]) | False | 0.631015 | 2.494360e+09 | 0.021067 | True | -3.0 | |
| | Equity(32660 [SFLY]) | False | 0.573869 | 1.460114e+09 | 0.054598 | True | -1.0 | |
| | Equity(42950 [FB]) | False | 0.799374 | 3.964390e+11 | 0.044532 | True | -3.0 | |
| 2019-01-08 00:00:00+00:00 | Equity(4705 [MKC]) | True | 1.207199 | 1.815031e+10 | 0.008708 | False | 5.0 | |
| | Equity(12909 [LH]) | False | 0.774010 | 1.281228e+10 | -0.007504 | True | 2.0 | |
| | Equity(23269 [WW]) | False | 0.621930 | 2.402793e+09 | -0.037245 | True | -3.0 | |

In [75]:

```
pipeline_outsample['return'] = pipeline_outsample["return"].apply(np.sign)
```

In [76]:
```
pipeline_outsample["Trading Signal"]  = pd.get_dummies(pipeline_outsample['BU
Y'],drop_first=True)
```

In [77]:
```
pipeline_outsample['Market Capital.'] = pipeline_outsample["Market Capital."].
apply(np.log)

pipeline_outsample.head(20)
```

Out[77]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_l |
|---|---|---|---|---|---|---|---|---|
| 2019-01-02 00:00:00+00:00 | Equity(3149 [GE]) | False | 0.571820 | 24.910565 | 0.005984 | True | -3.0 | -3 |
| | Equity(4705 [MKC]) | True | 1.234640 | 23.631740 | 0.002015 | False | 5.0 | 5 |
| | Equity(5029 [MRK]) | True | 1.202823 | 26.015036 | 0.014727 | False | 6.0 | 6 |
| | Equity(13635 [DO]) | False | 0.634035 | 20.983615 | -0.031828 | True | -3.0 | -3 |
| 2019-01-03 00:00:00+00:00 | Equity(337 [AMAT]) | False | 0.723636 | 24.191939 | 0.022283 | True | 2.0 | -3 |
| | Equity(7447 [TIF]) | False | 0.742030 | 23.023332 | 0.016522 | True | -1.0 | -3 |
| | Equity(13635 [DO]) | False | 0.626310 | 21.014901 | 0.032874 | True | -3.0 | -3 |
| | Equity(26143 [NRG]) | True | 1.206547 | 23.125627 | -0.037626 | False | 6.0 | 6 |
| | Equity(51649 [ADT]) | False | 0.783810 | 22.315699 | 0.064007 | True | -3.0 | -3 |
| 2019-01-04 00:00:00+00:00 | Equity(337 [AMAT]) | False | 0.718949 | 24.132247 | -0.058525 | True | 6.0 | -3 |
| | Equity(42950 [FB]) | False | 0.799401 | 26.659727 | -0.026828 | True | -3.0 | -3 |
| 2019-01-07 00:00:00+00:00 | Equity(4705 [MKC]) | True | 1.213979 | 23.612872 | 0.002936 | False | 5.0 | 5 |
| | Equity(9883 [ATVI]) | False | 0.681487 | 24.306593 | 0.040132 | True | -1.0 | -3 |
| | Equity(12909 [LH]) | False | 0.777963 | 23.281358 | 0.034196 | True | 4.0 | -3 |
| | Equity(23269 [WW]) | False | 0.631015 | 21.637298 | 0.021067 | True | -3.0 | -3 |
| | Equity(32660 [SFLY]) | False | 0.573869 | 21.101781 | 0.054598 | True | -1.0 | -3 |
| | Equity(42950 [FB]) | False | 0.799374 | 26.705788 | 0.044532 | True | -3.0 | -3 |
| 2019-01-08 00:00:00+00:00 | Equity(4705 [MKC]) | True | 1.207199 | 23.621954 | 0.008708 | False | 5.0 | 5 |
| | Equity(12909 [LH]) | False | 0.774010 | 23.273670 | -0.007504 | True | 2.0 | -3 |
| | Equity(23269 [WW]) | False | 0.621930 | 21.599898 | -0.037245 | True | -3.0 | -3 |

In [78]:
```
X_ofs17 = pipeline_outsample[feature_cols]
y_ofs17 = pipeline_outsample["return"]
len(X_ofs17)
```

Out[78]:
```
525
```

**Fitting Logestic Model to out of sample data:**

In [79]:
```
y_pred_ofs17 = logmodel.predict(X_ofs17)
y_pred_ofs17
```

Out[79]:
```
array([-1.,  1.,  1., -1., -1., -1., -1.,  1., -1., -1., -1.,  1., -1.,
       -1., -1., -1., -1.,  1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1.,  1., -1.,  1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1.,  1., -1., -1.,
        1., -1., -1., -1., -1., -1., -1.,  1., -1.,  1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,
        1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1., -1., -1.,  1., -1.,
        1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1.,  1.,
       -1., -1.,  1., -1.,  1., -1.,  1.,  1., -1.,  1., -1.,  1., -1.,
        1., -1.,  1., -1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,
        1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1., -1., -1.,  1.,
        1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
        1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1., -1.,  1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1., -1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,
       -1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,
        1., -1.,  1.,  1., -1.,  1., -1., -1., -1.,  1., -1.,  1., -1.,
        1.,  1.,  1., -1., -1., -1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
       -1., -1.,  1.,  1., -1., -1., -1.,  1.,  1.,  1., -1., -1., -1.,
        1., -1.,  1., -1., -1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.,
        1., -1.,  1., -1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1., -1.,
       -1.,  1.,  1., -1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,
        1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
       -1., -1.,  1., -1., -1., -1.,  1., -1., -1.,  1., -1., -1., -1.,
       -1.,  1.,  1., -1., -1., -1., -1.,  1.,  1., -1.,  1., -1., -1.,
        1.,  1., -1.,  1., -1., -1., -1.,  1.,  1.,  1., -1.,  1., -1.,
        1.,  1.,  1., -1., -1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,
       -1.,  1., -1.,  1.,  1., -1., -1., -1.,  1., -1., -1., -1.,  1.,
        1., -1., -1.,  1., -1., -1., -1.,  1.,  1.,  1., -1., -1., -1.,
        1.,  1.,  1., -1.,  1., -1., -1.,  1., -1., -1.,  1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1.,
        1., -1.,  1., -1.,  1.])
```

In [80]:
```
cnf_matrix = metrics.confusion_matrix(y_ofs17, y_pred_ofs17)
print cnf_matrix
print"Accuracy:",metrics.accuracy_score(y_ofs17, y_pred_ofs17)
print"Precision:",metrics.precision_score(y_ofs17, y_pred_ofs17)
print"Recall:",metrics.recall_score(y_ofs17, y_pred_ofs17)
print classification_report(y_ofs17,y_pred_ofs17)
```

```
[[132  93]
 [143 157]]
Accuracy: 0.550476190476
Precision: 0.628
Recall: 0.523333333333
             precision    recall  f1-score   support

       -1.0       0.48      0.59      0.53       225
        1.0       0.63      0.52      0.57       300

avg / total       0.56      0.55      0.55       525
```

In [ ]:

**Fitting Random Forest Classifier to out of sample data:**

In [81]:
```
y_rfc_pred_ofs_17 = rfc.predict(X_ofs17)
y_rfc_pred_ofs_17
```

Out[81]:
```
array([-1.,  1.,  1., -1., -1., -1., -1.,  1., -1., -1., -1.,  1., -1.,
       -1.,  1., -1.,  1.,  1.,  1., -1., -1., -1., -1., -1., -1.,  1.,
       -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,  1., -1.,  1.,
       -1., -1.,  1., -1., -1., -1.,  1., -1., -1., -1., -1., -1., -1.,
       -1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1.,  1., -1., -1., -1., -1.,  1., -1.,  1.,
       -1., -1.,  1., -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1.,  1., -1., -1.,
        1.,  1., -1., -1., -1.,  1., -1.,  1., -1.,  1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,
        1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1., -1., -1.,  1., -1.,
        1., -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
        1., -1.,  1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,
        1.,  1.,  1.,  1.,  1., -1., -1.,  1., -1.,  1., -1., -1.,  1.,
        1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
       -1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1.,
        1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
        1., -1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,
       -1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,
        1., -1.,  1.,  1., -1., -1., -1., -1., -1.,  1., -1.,  1., -1.,
        1.,  1.,  1., -1., -1., -1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
       -1., -1.,  1.,  1., -1., -1., -1.,  1.,  1.,  1., -1., -1., -1.,
        1., -1.,  1., -1., -1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,
        1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.,
        1., -1.,  1., -1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,
        1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
       -1., -1.,  1., -1., -1., -1.,  1., -1., -1.,  1., -1., -1., -1.,
       -1.,  1.,  1., -1., -1., -1., -1.,  1.,  1., -1.,  1., -1., -1.,
        1.,  1., -1.,  1., -1., -1., -1.,  1.,  1.,  1., -1.,  1., -1.,
        1.,  1.,  1., -1., -1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,
       -1.,  1., -1.,  1.,  1., -1., -1., -1.,  1., -1., -1., -1.,  1.,
        1., -1., -1.,  1., -1., -1., -1.,  1.,  1.,  1., -1., -1., -1.,
        1.,  1.,  1., -1.,  1., -1., -1.,  1., -1., -1.,  1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1.,
        1., -1.,  1., -1.,  1.])
```

In [82]:
```
cnf_matrix_rfc = metrics.confusion_matrix(y_ofs17, y_rfc_pred_ofs_17)
print(cnf_matrix_rfc)
print("Accuracy:",metrics.accuracy_score(y_ofs17, y_rfc_pred_ofs_17))
print("Precision:",metrics.precision_score(y_ofs17, y_rfc_pred_ofs_17))
print("Recall:",metrics.recall_score(y_ofs17, y_rfc_pred_ofs_17))
print classification_report(y_ofs17,y_rfc_pred_ofs_17)
```

```
[[121 104]
 [129 171]]
('Accuracy:', 0.5561904761904762 4)
('Precision:', 0.6218181818181818 3)
('Recall:', 0.5699999999999999 5)
             precision    recall  f1-score   support

       -1.0       0.48      0.54      0.51       225
        1.0       0.62      0.57      0.59       300

avg / total       0.56      0.56      0.56       525
```

In [ ]:
```
### Fitting Support Vector Machines
```

In [84]:
```
y_svm_pred_ofs_17 = sv_model.predict(X_ofs17)
y_svm_pred_ofs_17
```

Out[84]:
```
array([-1.,  1.,  1., -1.,  1., -1., -1.,  1., -1., -1., -1.,  1., -1.,
       -1., -1., -1., -1.,  1.,  1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1.,  1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1.,  1., -1.,  1., -1., -1., -1., -1., -1.,  1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1.,  1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1.,  1., -1., -1.,
        1., -1., -1., -1., -1., -1., -1.,  1., -1.,  1., -1., -1., -1.,
       -1., -1., -1., -1., -1., -1., -1., -1., -1.,  1., -1., -1., -1.,
        1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1., -1., -1.,  1., -1.,
        1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1., -1.,  1.,  1.,
       -1., -1.,  1., -1.,  1., -1.,  1.,  1., -1.,  1., -1.,  1., -1.,
        1., -1.,  1., -1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,
        1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1., -1., -1.,  1.,
        1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
        1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1., -1.,  1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1., -1.,  1., -1.,
        1., -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,
       -1.,  1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,
        1., -1.,  1.,  1., -1.,  1., -1., -1., -1.,  1., -1.,  1., -1.,
        1.,  1.,  1., -1., -1., -1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
       -1., -1.,  1.,  1., -1., -1., -1.,  1.,  1.,  1., -1., -1., -1.,
        1., -1.,  1., -1., -1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.,  1., -1.,  1.,  1.,  1., -1.,  1.,  1., -1.,  1.,
        1., -1.,  1., -1.,  1., -1., -1.,  1.,  1.,  1., -1.,  1., -1.,
       -1.,  1.,  1., -1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,
        1.,  1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,  1.,  1.,
       -1., -1.,  1., -1., -1., -1.,  1., -1., -1.,  1., -1., -1., -1.,
       -1.,  1.,  1., -1., -1., -1., -1.,  1.,  1., -1.,  1., -1., -1.,
        1.,  1., -1.,  1., -1., -1., -1.,  1.,  1.,  1., -1.,  1., -1.,
        1.,  1.,  1., -1., -1., -1.,  1.,  1., -1.,  1.,  1.,  1., -1.,
       -1.,  1., -1.,  1.,  1., -1., -1., -1.,  1., -1., -1., -1.,  1.,
        1., -1., -1.,  1., -1., -1., -1.,  1.,  1.,  1., -1., -1., -1.,
        1.,  1.,  1., -1.,  1., -1., -1.,  1., -1., -1.,  1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1.,  1.,  1., -1., -1.,  1.,  1.,  1.,
       -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1., -1.,  1.,  1.,  1.,
        1., -1.,  1., -1.,  1.])
```

In [86]:
```
cnf_matrix_svm = metrics.confusion_matrix(y_ofs17, y_svm_pred_ofs_17)
print(cnf_matrix_svm)
print("Accuracy:",metrics.accuracy_score(y_ofs17, y_svm_pred_ofs_17))
print("Precision:",metrics.precision_score(y_ofs17, y_svm_pred_ofs_17))
print("Recall:",metrics.recall_score(y_ofs17, y_svm_pred_ofs_17))
print classification_report(y_ofs17,y_svm_pred_ofs_17)
```

```
[[129  96]
 [141 159]]
('Accuracy:', 0.5485714285714286)
('Precision:', 0.6235294117647059)
('Recall:', 0.5300000000000003)
             precision    recall  f1-score   support

       -1.0       0.48      0.57      0.52       225
        1.0       0.62      0.53      0.57       300

avg / total       0.56      0.55      0.55       525
```

In [87]:
```
pipeline_outsample.tail(20)
```

Out[87]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_la |
|---|---|---|---|---|---|---|---|---|
| 2019-06-19 00:00:00+00:00 | Equity(460 [APD]) | True | 1.240573 | 24.598447 | 0.009592 | False | 6.0 | 6 |
| | Equity(4974 [MSI]) | True | 1.204562 | 24.026651 | 0.007507 | False | 6.0 | 6 |
| | Equity(5773 [PBI]) | False | 0.629607 | 20.428344 | -0.021378 | True | -3.0 | -3 |
| | Equity(8352 [XRAY]) | True | 1.272900 | 23.310976 | 0.003354 | False | 6.0 | 4 |
| | Equity(18221 [VRSN]) | True | 1.207384 | 23.923710 | 0.012737 | False | 5.0 | 5 |
| | Equity(42270 [CPRI]) | False | 0.679690 | 22.378551 | 0.023311 | True | -3.0 | -3 |
| 2019-06-20 00:00:00+00:00 | Equity(460 [APD]) | True | 1.242997 | 24.605139 | 0.006578 | False | 6.0 | 6 |
| | Equity(4974 [MSI]) | True | 1.210011 | 24.035395 | 0.008541 | False | 6.0 | 6 |
| | Equity(5773 [PBI]) | False | 0.627407 | 20.442802 | 0.014563 | True | -3.0 | -3 |
| | Equity(8352 [XRAY]) | True | 1.275001 | 23.326859 | 0.016010 | False | 6.0 | 4 |
| | Equity(18221 [VRSN]) | True | 1.209886 | 23.940468 | 0.016413 | False | 5.0 | 5 |
| | Equity(42270 [CPRI]) | False | 0.676282 | 22.365783 | -0.012399 | True | -3.0 | -3 |
| 2019-06-21 00:00:00+00:00 | Equity(460 [APD]) | True | 1.245247 | 24.612643 | 0.007624 | False | 6.0 | 6 |
| | Equity(939 [BLL]) | True | 1.290730 | 23.837294 | 0.009002 | False | 6.0 | 4 |
| | Equity(3676 [HRS]) | True | 1.209753 | 23.882572 | 0.005192 | False | 4.0 | 6 |
| | Equity(4974 [MSI]) | True | 1.214935 | 24.037315 | 0.002823 | False | 6.0 | 6 |
| | Equity(5773 [PBI]) | False | 0.623981 | 20.413674 | -0.028708 | True | -3.0 | -3 |
| | Equity(8352 [XRAY]) | True | 1.276252 | 23.321302 | -0.005714 | False | 6.0 | 4 |
| | Equity(42270 [CPRI]) | False | 0.672756 | 22.386306 | 0.020146 | True | -3.0 | -3 |
| | Equity(42277 [ZNGA]) | True | 1.357073 | 22.459718 | 0.002473 | False | 4.0 | 4 |

In [124]:
```
pipeline_outsample.iloc[pipeline_outsample.index.levels[0] == '2019-06-21 00:0
0:00+00:00']
```

Out[124]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_lag |
|---|---|---|---|---|---|---|---|---|
| 2019-03-04 00:00:00+00:00 | Equity(939 [BLL]) | True | 1.253408 | 23.638807 | 0.008216 | False | 6.0 | 4.0 |

# Stepping Up a year and refitting both models:

In [44]:
```
pipeline_output_2 = run_pipeline(
    make_pipeline(),
    start_date="2014-01-01",
    end_date="2018-01-01"
)
```

In [45]:
```
pipeline_output_2['return'] = pipeline_output_2["return"].apply(np.sign)
pipeline_output_2['Market Capital.'] = pipeline_output_2["Market Capital."].ap
ply(np.log)
pipeline_output_2["Trading Signal"] = pd.get_dummies(pipeline_output_2['BUY'],
drop_first=True)
```

In [46]:
```
n=float(len(pipeline_output_2[pipeline_output_2["return"]>0]))
m=float(len(pipeline_output_2[pipeline_output_2["Trading Signal"]==1]))
a=float(len(pipeline_output_2[pipeline_output_2["return"]<0]))
b=float(len(pipeline_output_2[pipeline_output_2["Trading Signal"]==0]))
x=float(len(pipeline_output_2[pipeline_output_2["Returns"]>0]))
y=float(len(pipeline_output_2[pipeline_output_2["Returns"]<0]))
z=float(len(pipeline_output))
print"The percentage of positive returns is:", ((n/z)*100),"%"
print"The percentage of BUY Trading Signal is:", ((m/z)*100),"%"
print"The percentage of negative returns is:", ((a/z)*100),"%"
print"The percentage of SELL Trading Signal is:", ((b/z)*100),"%"
```

```
The percentage of positive returns is: 46.2763890688 %
The percentage of BUY Trading Signal is: 63.644605621 %
The percentage of negative returns is: 43.970988214 %
The percentage of SELL Trading Signal is: 26.6027716617 %
```

In [47]:
```
pipeline_output_2.tail(20)
```

Out[47]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_la |
|---|---|---|---|---|---|---|---|---|
| | Equity(24124 [WYNN]) | True | 1.279001 | 23.585159 | -0.010077 | False | 4.0 | 6. |
| | Equity(24811 [GES]) | True | 1.273858 | 21.078006 | -0.011541 | False | 6.0 | 6. |
| | Equity(24832 [RL]) | True | 1.201872 | 22.851096 | -0.002140 | False | 4.0 | 4. |
| 2017-12-29 00:00:00+00:00 | Equity(25920 [MAR]) | True | 1.273396 | 24.628752 | 0.002422 | False | 6.0 | 5. |
| | Equity(27676 [AMP]) | True | 1.234831 | 23.953562 | 0.004271 | False | 4.0 | 4. |
| | Equity(32902 [FSLR]) | True | 1.555720 | 22.695841 | -0.006538 | False | 6.0 | 4. |
| | Equity(41636 [MPC]) | True | 1.214153 | 24.202668 | -0.000301 | False | 6.0 | 6. |
| | Equity(114 [ADBE]) | True | 1.215385 | 25.178645 | -0.001424 | False | 6.0 | 6. |
| | Equity(1267 [CAT]) | True | 1.331272 | 25.263894 | -0.005301 | False | 6.0 | 6. |
| | Equity(1539 [CI]) | True | 1.202831 | 24.636723 | -0.010037 | False | 5.0 | 5. |
| | Equity(1941 [CTAS]) | True | 1.204947 | 23.532140 | -0.004980 | False | 4.0 | 4. |
| | Equity(3321 [GPS]) | True | 1.343680 | 23.306846 | -0.010456 | False | 5.0 | 5. |
| | Equity(6546 [ROST]) | True | 1.221500 | 24.146460 | -0.004094 | False | 6.0 | 6. |
| 2018-01-02 00:00:00+00:00 | Equity(7590 [TROW]) | True | 1.294825 | 23.959022 | -0.003987 | False | 4.0 | 4. |
| | Equity(23269 [WW]) | True | 1.481039 | 21.773177 | -0.064942 | False | 6.0 | 6. |
| | Equity(24124 [WYNN]) | True | 1.280397 | 23.575596 | 0.000831 | False | 4.0 | 6. |
| | Equity(24811 [GES]) | True | 1.272976 | 21.051696 | -0.014594 | False | 6.0 | 6. |
| | Equity(24832 [RL]) | True | 1.206513 | 22.854768 | 0.010624 | False | 4.0 | 4. |
| | Equity(27676 [AMP]) | True | 1.234680 | 23.944926 | -0.012584 | False | 4.0 | 4. |
| | Equity(32902 [FSLR]) | True | 1.559013 | 22.676480 | -0.012723 | False | 6.0 | 6. |

In [48]:
```
X_1 = pipeline_output_2[feature_cols]
y_1 = pipeline_output_2["return"]
```

In [49]:
```
X_train_1,X_test_1,y_train_1,y_test_1=train_test_split(X_1,y_1,test_size=0.2,r
andom_state= None)
```

### Fit New Logestic Model

In [50]:
```
# instantiate the model (using the default parameters)
logmodel_2 = LogisticRegression()
# fit the model with data
logmodel_2.fit(X_train_1,y_train_1)
```

Out[50]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0)
```

In [51]:
```
y_pred_1=logmodel.predict(X_test_1)
y_pred_1
```

Out[51]:
```
array([ 1., -1.,  1., ..., -1.,  1., -1.])
```

In [52]:
```
cnf_matrix_1 = metrics.confusion_matrix(y_test_1, y_pred_1)
print(cnf_matrix_1)
print"Accuracy:",metrics.accuracy_score(y_test_1, y_pred_1)
print"Precision:",metrics.precision_score(y_test_1, y_pred_1)
print"Recall:",metrics.recall_score(y_test_1, y_pred_1)
print classification_report(y_test_1,y_pred_1)
```

```
[[235 458]
 [197 504]]
Accuracy: 0.530129124821
Precision: 0.523908523909
Recall: 0.718972895863
             precision    recall  f1-score   support

       -1.0       0.54      0.34      0.42       693
        1.0       0.52      0.72      0.61       701

avg / total       0.53      0.53      0.51      1394
```

## Fit New Random Forest Classifier Model

In [53]:
```
rfc_2 = RandomForestClassifier(n_estimators=100, max_leaf_nodes=2,max_features
=7, min_samples_split=1)
rfc_2.fit(X_train_1, y_train_1)
```

Out[53]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=7, max_leaf_nodes=2,
            min_samples_leaf=1, min_samples_split=1,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [54]:
```
rfc_pred_2 = rfc.predict(X_test_1)
rfc_pred_2
cnf_matrix_1 = metrics.confusion_matrix(y_test_1, rfc_pred_2)
print cnf_matrix_1
print "Accuracy:",metrics.accuracy_score(y_test_1, rfc_pred_2)
print "Precision:",metrics.precision_score(y_test_1, rfc_pred_2)
print "Recall:",metrics.recall_score(y_test_1, rfc_pred_2)
print classification_report(y_test_1,rfc_pred_2)
```

```
[[246 447]
 [212 489]]
Accuracy: 0.527259684362
Precision: 0.522435897436
Recall: 0.69757489301
             precision    recall  f1-score   support

       -1.0       0.54      0.35      0.43       693
        1.0       0.52      0.70      0.60       701

avg / total       0.53      0.53      0.51      1394
```

## Out of Sample Predictions:

In [55]:
```
pipeline_outsample_2 = run_pipeline(
    make_pipeline(),
    start_date="2018-01-01",
    end_date="2019-01-01"
)
```

In [56]:
```
pipeline_outsample_2['return'] = pipeline_outsample_2["return"].apply(np.sign)
pipeline_outsample_2['Market Capital.'] = pipeline_outsample_2["Market Capita
l."].apply(np.log)

pipeline_outsample_2["Trading Signal"] = pd.get_dummies(pipeline_outsample_2[
'BUY'],drop_first=True)
```

In [57]:
```
pipeline_outsample_2.head(20)
```

Out[57]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_la |
|---|---|---|---|---|---|---|---|---|
| 2018-01-02 00:00:00+00:00 | Equity(114 [ADBE]) | True | 1.215385 | 25.178645 | -0.001424 | False | 6.0 | 6. |
| | Equity(1267 [CAT]) | True | 1.331272 | 25.263894 | -0.005301 | False | 6.0 | 6. |
| | Equity(1539 [CI]) | True | 1.202831 | 24.636723 | -0.010037 | False | 5.0 | 5. |
| | Equity(1941 [CTAS]) | True | 1.204947 | 23.532140 | -0.004980 | False | 4.0 | 4. |
| | Equity(3321 [GPS]) | True | 1.343680 | 23.306846 | -0.010456 | False | 5.0 | 5. |
| | Equity(6546 [ROST]) | True | 1.221500 | 24.146460 | -0.004094 | False | 6.0 | 6. |
| | Equity(7590 [TROW]) | True | 1.294825 | 23.959022 | -0.003987 | False | 4.0 | 4. |
| | Equity(23269 [WW]) | True | 1.481039 | 21.773177 | -0.064942 | False | 6.0 | 6. |
| | Equity(24124 [WYNN]) | True | 1.280397 | 23.575596 | 0.000831 | False | 4.0 | 6. |
| | Equity(24811 [GES]) | True | 1.272976 | 21.051696 | -0.014594 | False | 6.0 | 6. |
| | Equity(24832 [RL]) | True | 1.206513 | 22.854768 | 0.010624 | False | 4.0 | 4. |
| | Equity(27676 [AMP]) | True | 1.234680 | 23.944926 | -0.012584 | False | 4.0 | 4. |
| | Equity(32902 [FSLR]) | True | 1.559013 | 22.676480 | -0.012723 | False | 6.0 | 6. |
| 2018-01-03 00:00:00+00:00 | Equity(114 [ADBE]) | True | 1.211652 | 25.192585 | 0.013634 | False | 6.0 | 6. |
| | Equity(1267 [CAT]) | True | 1.334898 | 25.260461 | -0.003743 | False | 6.0 | 6. |
| | Equity(1941 [CTAS]) | True | 1.203180 | 23.538664 | 0.006417 | False | 4.0 | 4. |
| | Equity(2127 [DE]) | True | 1.263072 | 24.654567 | 0.009070 | False | 6.0 | 6. |
| | Equity(2298 [DHI]) | True | 1.377207 | 23.678109 | -0.000587 | False | 6.0 | 6. |
| | Equity(3321 [GPS]) | True | 1.344450 | 23.301252 | 0.000591 | False | 5.0 | 5. |
| | Equity(5121 [MU]) | True | 1.326128 | 24.645165 | 0.061512 | False | 6.0 | 6. |

In [58]:
```
X_ofs18 = pipeline_outsample_2[feature_cols]
y_ofs18 = pipeline_outsample_2["return"]
```

In [59]:
```
y_pred_ofs18 = logmodel_2.predict(X_ofs18)
y_pred_ofs18
```

Out[59]:
```
array([ 1.,  1.,  1., ...,  1.,  1., -1.])
```

## Logestic Regression

In [60]:
```
cnf_matrix_ofs2 = metrics.confusion_matrix(y_ofs18, y_pred_ofs18)
print cnf_matrix_ofs2
print "Accuracy:",metrics.accuracy_score(y_ofs18, y_pred_ofs18)
print "Precision:",metrics.precision_score(y_ofs18, y_pred_ofs18)
print "Recall:",metrics.recall_score(y_ofs18, y_pred_ofs18)
print classification_report(y_ofs18,y_pred_ofs18)
```

```
[[ 174 1479]
 [ 129 1841]]
Accuracy: 0.556168920784
Precision: 0.554518072289
Recall: 0.934517766497
             precision    recall  f1-score   support

       -1.0       0.57      0.11      0.18      1653
        1.0       0.55      0.93      0.70      1970

avg / total       0.56      0.56      0.46      3623
```

## Random Forest Classifier

In [61]:
```
y_rfc_pred_ofs18 = rfc_2.predict(X_ofs18)
cnf_matrix_ofs2 = metrics.confusion_matrix(y_ofs18, y_rfc_pred_ofs18)
print cnf_matrix_ofs2
print "Accuracy:",metrics.accuracy_score(y_ofs18, y_rfc_pred_ofs18)
print "Precision:",metrics.precision_score(y_ofs18, y_rfc_pred_ofs18)
print "Recall:",metrics.recall_score(y_ofs18, y_rfc_pred_ofs18)
print classification_report(y_ofs18,y_rfc_pred_ofs18)
```

```
[[ 114 1539]
 [  91 1879]]
Accuracy: 0.550096605023
Precision: 0.549736688122
Recall: 0.953807106599
             precision    recall  f1-score   support

       -1.0       0.56      0.07      0.12      1653
        1.0       0.55      0.95      0.70      1970

avg / total       0.55      0.55      0.44      3623
```

In [62]:
```
print('Number of securities that passed the filter: %d' % len(pipeline_output.index.levels[1].unique()))
print('Number of securities that passed the filter: %d' % len(pipeline_output_2.index.levels[1].unique()))
print('Number of securities that passed the filter: %d' % len(pipeline_outsample.index.levels[1].unique()))
print('Number of securities that passed the filter: %d' % len(pipeline_outsample_2.index.levels[1].unique()))
```

```
Number of securities that passed the filter: 365
Number of securities that passed the filter: 325
Number of securities that passed the filter: 129
Number of securities that passed the filter: 162
```

# Use Models to Predict EOD Return's Direction:

## Date: 05/06/2019

### Wait till 7am ET for sentiment datasets to be updated

#### PsychSignal Trader Mood : Update Frequency: Daily (updated every morning at ~7am ET)

#### Sentdex Sentiment : Update Frequency: Daily (updated every morning at ~7am ET)

#### US Equities Pricing : Update Frequency: Daily (updated overnight after each trading day).

In [18]:
```
# Prices Update Frequency: Daily (updated overnight after each trading day).

predict = run_pipeline(
    make_pipeline(),
    start_date="2019-07-22",
    end_date="2019-07-22"
)
```

**Pipeline Execution Time:** 2 Minutes, 11.41 Seconds

In [19]:
```
predict
```

Out[19]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentde |
|---|---|---|---|---|---|---|---|---|
| **2019-07-22 00:00:00+00:00** | **Equity(460 [APD])** | True | 1.279344 | 4.942636e+10 | -0.004389 | False | 4.0 | |
| | **Equity(4488 [LM])** | True | 1.236497 | 3.298690e+09 | -0.006261 | False | 6.0 | |
| | **Equity(6683 [SBUX])** | True | 1.296297 | 1.093714e+11 | -0.013213 | False | 3.0 | |
| | **Equity(7684 [TSN])** | True | 1.237770 | 2.947701e+10 | -0.001976 | False | 4.0 | |
| | **Equity(7998 [VMC])** | True | 1.202992 | 1.796585e+10 | -0.003005 | False | 6.0 | |
| | **Equity(16511 [KMX])** | True | 1.213953 | 1.404833e+10 | -0.020663 | False | 6.0 | |
| | **Equity(23438 [GME])** | False | 0.452238 | 4.418018e+08 | 0.030952 | True | -1.0 | |
| | **Equity(32902 [FSLR])** | True | 1.247712 | 7.009147e+09 | 0.008489 | False | 6.0 | |
| | **Equity(34395 [LULU])** | True | 1.221689 | 2.448399e+10 | -0.007551 | False | 5.0 | |
| | **Equity(38936 [DG])** | True | 1.207936 | 3.649585e+10 | -0.010504 | False | 5.0 | |

In [20]:
```
predict['Market Capital.'] = predict["Market Capital."].apply(np.log)

predict["Trading Signal"] = pd.get_dummies(predict['BUY'],drop_first=True)

predict
```

Out[20]:

| | | BUY | MAD | Market Capital. | Returns | SHORT | Sentdex | Sentdex_la |
|---|---|---|---|---|---|---|---|---|
| 2019-07-22 00:00:00+00:00 | Equity(460 [APD]) | True | 1.279344 | 24.623750 | -0.004389 | False | 4.0 | 4 |
| | Equity(4488 [LM]) | True | 1.236497 | 21.916791 | -0.006261 | False | 6.0 | 6 |
| | Equity(6683 [SBUX]) | True | 1.296297 | 25.418015 | -0.013213 | False | 3.0 | 6 |
| | Equity(7684 [TSN]) | True | 1.237770 | 24.106876 | -0.001976 | False | 4.0 | 4 |
| | Equity(7998 [VMC]) | True | 1.202992 | 23.611739 | -0.003005 | False | 6.0 | 6 |
| | Equity(16511 [KMX]) | True | 1.213953 | 23.365769 | -0.020663 | False | 6.0 | 4 |
| | Equity(23438 [GME]) | False | 0.452238 | 19.906372 | 0.030952 | True | -1.0 | -3 |
| | Equity(32902 [FSLR]) | True | 1.247712 | 22.670482 | 0.008489 | False | 6.0 | 5 |
| | Equity(34395 [LULU]) | True | 1.221689 | 23.921285 | -0.007551 | False | 5.0 | 5 |
| | Equity(38936 [DG]) | True | 1.207936 | 24.320464 | -0.010504 | False | 5.0 | 5 |

In [ ]:

In [21]:
```
X_live = predict[feature_cols]
print logmodel.predict(X_live)
# print logmodel_2.predict(X_live)
print rfc.predict(X_live)
# print rfc_2.predict(X_live)
print logmodel.predict_proba(X_live)
# print logmodel_2.predict_proba(X_live)
print rfc.predict_proba(X_live)
# print rfc_2.predict_proba(X_live)
print sv_model.predict(X_live)
```

```
[ 1.  1.  1.  1.  1.  1. -1.  1.  1.  1.]
[ 1.  1.  1.  1.  1.  1. -1.  1.  1.  1.]
[[ 0.4393584   0.5606416 ]
 [ 0.46324522  0.53675478]
 [ 0.42302052  0.57697948]
 [ 0.43393543  0.56606457]
 [ 0.45124281  0.54875719]
 [ 0.45998763  0.54001237]
 [ 0.61874917  0.38125083]
 [ 0.46629742  0.53370258]
 [ 0.45695699  0.54304301]
 [ 0.4397027   0.5602973 ]]
[[ 0.43975568  0.56024432]
 [ 0.43975568  0.56024432]
 [ 0.46369274  0.53630726]
 [ 0.43975568  0.56024432]
 [ 0.43975568  0.56024432]
 [ 0.43975568  0.56024432]
 [ 0.58097924  0.41902076]
 [ 0.43975568  0.56024432]
 [ 0.44003627  0.55996373]
 [ 0.43975568  0.56024432]]
[ 1.  1.  1.  1.  1.  1. -1.  1.  1.  1.]
```

In [22]:
```
predict.index.levels[1]
```

Out[22]:
```
Index([   Equity(460 [APD]),    Equity(4488 [LM]),  Equity(6683 [SBUX]),
          Equity(7684 [TSN]),   Equity(7998 [VMC]),  Equity(16511 [KMX]),
        Equity(23438 [GME]), Equity(32902 [FSLR]), Equity(34395 [LULU]),
          Equity(38936 [DG])],
      dtype='object')
```

In [23]:
```
predict.index.levels[1]
predict['Return Predictions LR'] = logmodel.predict(X_live)
predict["Return Predictions RFC"] = rfc.predict(X_live)
predict["Return Predictions SVM"] = sv_model.predict(X_live)
```

In [24]:
```
drop_cols = ["BUY", "Returns","SHORT", "close_price", "return"]
predict.drop(feature_cols, axis = 1, inplace=True)
predict.drop(drop_cols, axis = 1, inplace = True)
```

In [25]:
```
predict["LR Probability (-1)"] = logmodel.predict_proba(X_live)[:,0]
predict["LR Probability  (1)"] = logmodel.predict_proba(X_live)[:,1]
predict["RFC Probability (-1)"] = rfc.predict_proba(X_live)[:,0]
predict["RFC Probability  (1)"] = rfc.predict_proba(X_live)[:,1]
```

In [26]:
```
predict
```

Out[26]:

| | | Return Predictions LR | Return Predictions RFC | Return Predictions SVM | LR Probability (-1) | LR Probability (1) | Prob |
|---|---|---|---|---|---|---|---|
| 2019-07-22 00:00:00+00:00 | Equity(460 [APD]) | 1.0 | 1.0 | 1.0 | 0.439358 | 0.560642 | 0.4 |
| | Equity(4488 [LM]) | 1.0 | 1.0 | 1.0 | 0.463245 | 0.536755 | 0.4 |
| | Equity(6683 [SBUX]) | 1.0 | 1.0 | 1.0 | 0.423021 | 0.576979 | 0.4 |
| | Equity(7684 [TSN]) | 1.0 | 1.0 | 1.0 | 0.433935 | 0.566065 | 0.4 |
| | Equity(7998 [VMC]) | 1.0 | 1.0 | 1.0 | 0.451243 | 0.548757 | 0.4 |
| | Equity(16511 [KMX]) | 1.0 | 1.0 | 1.0 | 0.459988 | 0.540012 | 0.4 |
| | Equity(23438 [GME]) | -1.0 | -1.0 | -1.0 | 0.618749 | 0.381251 | 0.5 |
| | Equity(32902 [FSLR]) | 1.0 | 1.0 | 1.0 | 0.466297 | 0.533703 | 0.4 |
| | Equity(34395 [LULU]) | 1.0 | 1.0 | 1.0 | 0.456957 | 0.543043 | 0.4 |
| | Equity(38936 [DG]) | 1.0 | 1.0 | 1.0 | 0.439703 | 0.560297 | 0.4 |

- **NOTES:** :
  - Causility Test between stock closing prices and lagged sentdex sentiment score. Check how many lags to apply.
  - 1: [https://github.com/statsmodels/statsmodels/blob/master/statsmodels/tsa/stattools.py (https://github.com/statsmodels/statsmodels/blob/master/statsmodels/tsa/stattools.py)](https://github.com/statsmodels/statsmodels/blob/master/statsmodels/tsa/stattools.py)
  - 2: from statsmodels.tsa.stattools import grangercausalitytests

In [70]:
```
def grangercausalitytests(x, maxlag, addconst=True, verbose=True):
    """four tests for granger non causality of 2 timeseries
    all four tests give similar results
    `params_ftest` and `ssr_ftest` are equivalent based on F test which is
    identical to lmtest:grangertest in R
    Parameters
    ----------
    x : array, 2d
        data for test whether the time series in the second column Granger
        causes the time series in the first column
    maxlag : integer
        the Granger causality test results are calculated for all lags up to
        maxlag
    verbose : bool
        print results if true
    Returns
    -------
    results : dictionary
        all test results, dictionary keys are the number of lags. For each
        lag the values are a tuple, with the first element a dictionary with
        teststatistic, pvalues, degrees of freedom, the second element are
        the OLS estimation results for the restricted model, the unrestricted
        model and the restriction (contrast) matrix for the parameter f_test.
    Notes
    -----
    TODO: convert to class and attach results properly
    The Null hypothesis for grangercausalitytests is that the time series in
    the second column, x2, does NOT Granger cause the time series in the first
    column, x1. Grange causality means that past values of x2 have a
    statistically significant effect on the current value of x1, taking past
    values of x1 into account as regressors. We reject the null hypothesis
    that x2 does not Granger cause x1 if the pvalues are below a desired size
    of the test.
    The null hypothesis for all four test is that the coefficients
    corresponding to past values of the second time series are zero.
    'params_ftest', 'ssr_ftest' are based on F distribution
    'ssr_chi2test', 'lrtest' are based on chi-square distribution
    References
    ----------
    http://en.wikipedia.org/wiki/Granger_causality
    Greene: Econometric Analysis
    """
    from scipy import stats

    x = np.asarray(x)

    if x.shape[0] <= 3 * maxlag + int(addconst):
        raise ValueError("Insufficient observations. Maximum allowable "
                         "lag is {0}".format(int((x.shape[0] - int(addconst))
/
                                                 3) - 1))

    resli = {}

    for mlg in range(1, maxlag + 1):
        result = {}
```

```
        if verbose:
            print('\nGranger Causality')
            print('number of lags (no zero)', mlg)
        mxlg = mlg

        # create lagmat of both time series
        dta = lagmat2ds(x, mxlg, trim='both', dropex=1)

        #add constant
        if addconst:
            dtaown = add_constant(dta[:, 1:(mxlg + 1)], prepend=False)
            dtajoint = add_constant(dta[:, 1:], prepend=False)
        else:
            raise NotImplementedError('Not Implemented')
            #dtaown = dta[:, 1:mxlg]
            #dtajoint = dta[:, 1:]

        # Run ols on both models without and with lags of second variable
        res2down = OLS(dta[:, 0], dtaown).fit()
        res2djoint = OLS(dta[:, 0], dtajoint).fit()

        #print results
        #for ssr based tests see:
        #http://support.sas.com/rnd/app/examples/ets/granger/index.htm
        #the other tests are made-up

        # Granger Causality test using ssr (F statistic)
        fgc1 = ((res2down.ssr - res2djoint.ssr) /
                res2djoint.ssr / mxlg * res2djoint.df_resid)
        if verbose:
            print('ssr based F test:         F=%-8.4f, p=%-8.4f, df_denom=%d,'
                   ' df_num=%d' % (fgc1,
                                   stats.f.sf(fgc1, mxlg,
                                              res2djoint.df_resid),
                                   res2djoint.df_resid, mxlg))
        result['ssr_ftest'] = (fgc1,
                              stats.f.sf(fgc1, mxlg, res2djoint.df_resid),
                              res2djoint.df_resid, mxlg)

        # Granger Causality test using ssr (ch2 statistic)
        fgc2 = res2down.nobs * (res2down.ssr - res2djoint.ssr) / res2djoint.ss
r
        if verbose:
            print('ssr based chi2 test:   chi2=%-8.4f, p=%-8.4f, '
                   'df=%d' % (fgc2, stats.chi2.sf(fgc2, mxlg), mxlg))
        result['ssr_chi2test'] = (fgc2, stats.chi2.sf(fgc2, mxlg), mxlg)

        #likelihood ratio test pvalue:
        lr = -2 * (res2down.llf - res2djoint.llf)
        if verbose:
            print('likelihood ratio test: chi2=%-8.4f, p=%-8.4f, df=%d' %
                   (lr, stats.chi2.sf(lr, mxlg), mxlg))
        result['lrtest'] = (lr, stats.chi2.sf(lr, mxlg), mxlg)

        # F test that all lag coefficients of exog are zero
        rconstr = np.column_stack((np.zeros((mxlg, mxlg)),
                                  np.eye(mxlg, mxlg),
```

```
                                np.zeros((mxlg, 1))))
            ftres = res2djoint.f_test(rconstr)
            if verbose:
                print('parameter F test:         F=%-8.4f, p=%-8.4f, df_denom=%d,'
                       ' df_num=%d' % (ftres.fvalue, ftres.pvalue, ftres.df_denom,
                                        ftres.df_num))
            result['params_ftest'] = (np.squeeze(ftres.fvalue)[()],
                                      np.squeeze(ftres.pvalue)[()],
                                      ftres.df_denom, ftres.df_num)

            resli[mxlg] = (result, [res2down, res2djoint, rconstr])

    return resli
```

In [ ]:

In [ ]:

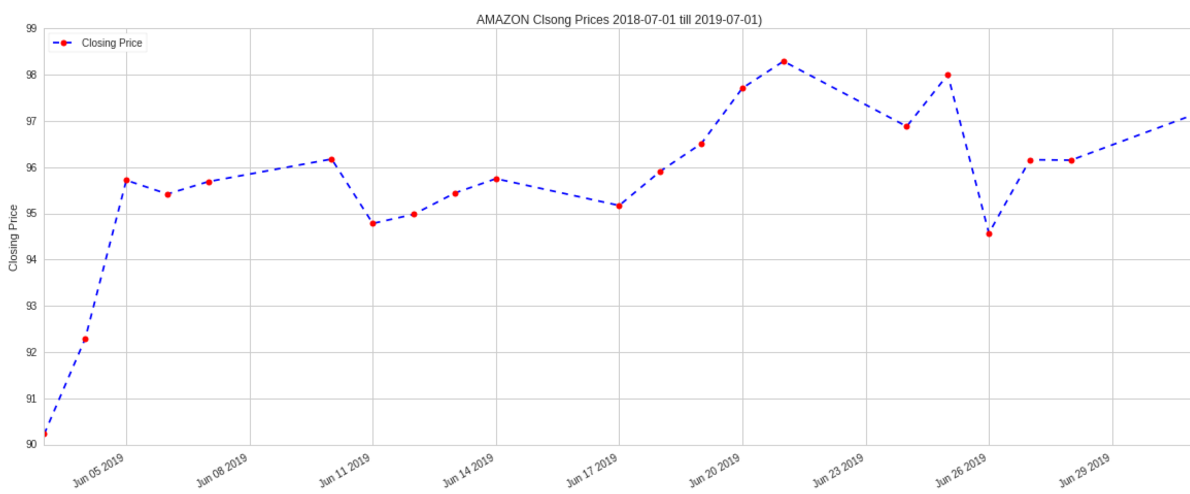
In [ ]:

In [ ]:

In [130]:
```
beg_date = '2019-06-01'
end_date = '2019-07-01'
```

In [131]:
```
stock = get_pricing("NDAQ", start_date = beg_date,
                    end_date = end_date,
                    frequency = 'daily')
```

In [132]:
```
stock['close_price'].plot(label = "Closing Price", figsize = (20,8), c = 'blu
e', marker='o',
         markerfacecolor='red', markersize=6, linestyle = "--")
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.title("AMAZON Clsong Prices 2018-07-01 till 2019-07-01)")
plt.legend(loc = 2);
```

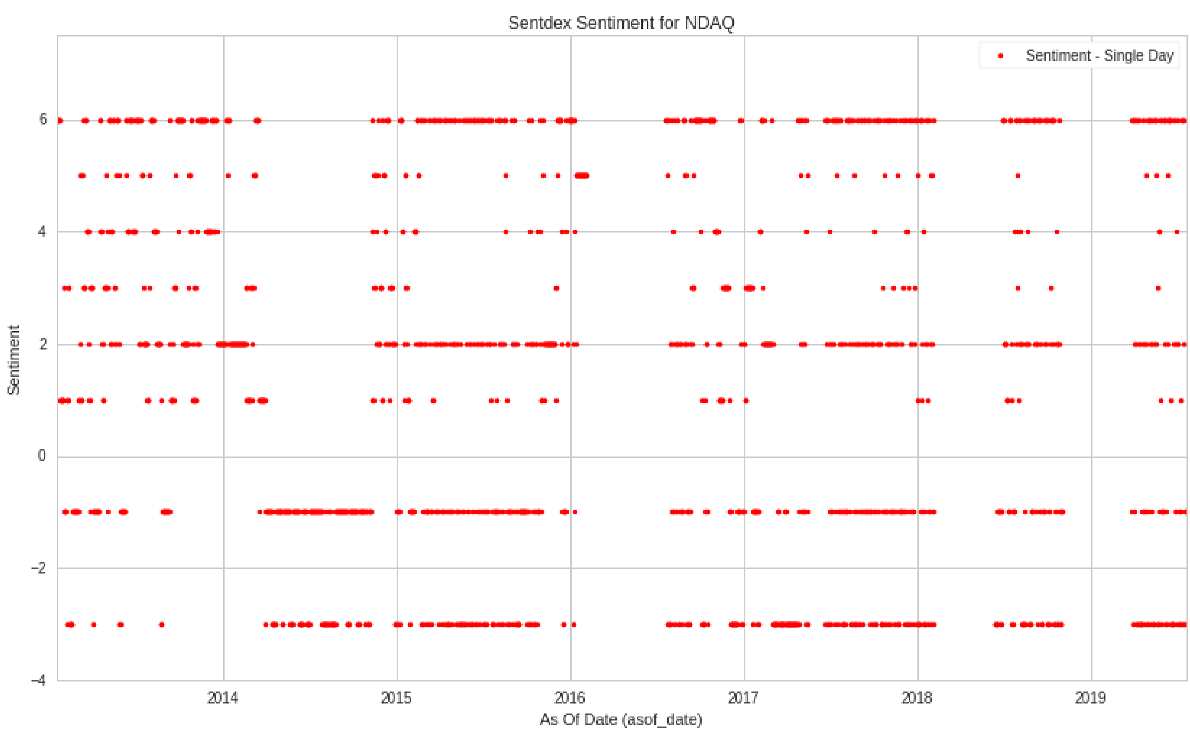


In [133]:
```
from quantopian.interactive.data.sentdex import sentiment as dataset

# import data operations
from odo import odo
```

In [134]:
```
dataset.dshape
```

Out[134]:
```
dshape("""var * {
  symbol: string,
  sentiment_signal: float64,
  sid: int64,
  asof_date: datetime,
  timestamp: datetime
  }""")
```

In [135]:
```
ndaq = dataset[dataset.symbol == "NDAQ"]
ndaq_df = odo(ndaq.sort('asof_date'), pd.DataFrame)
plt.plot(ndaq_df.asof_date, ndaq_df.sentiment_signal, marker='.', linestyle='N
one', color='r')
# plt.plot(ndaq_df.asof_date, pd.rolling_mean(ndaq_df.sentiment_signal, 21))
# plt.plot(ndaq_df.asof_date, pd.rolling_mean(ndaq_df.sentiment_signal, 252))
plt.xlabel("As Of Date (asof_date)")
plt.ylabel("Sentiment")
plt.title("Sentdex Sentiment for NDAQ")
plt.legend(["Sentiment - Single Day"], loc=1)
x1,x2,y1,y2 = plt.axis()
plt.axis((x1,x2,-4,7.5));
```




In [136]:
```
initial = ndaq_df.index[ndaq_df.asof_date == beg_date][0]
end = ndaq_df.index[ndaq_df.asof_date == end_date][0] + 1
```

In [137]:
```
my_test_df = pd.DataFrame()
```

In [138]:
```
my_test_df["ClosePrice"] = stock['close_price']
```

In [139]:
```
my_test_df["SentimentScore"] = ndaq_df['sentiment_signal'][initial:end]
```

In [143]:
```
len(stock['close_price'])
```

Out[143]:
```
21
```

In [144]:
```
ndaq_df.head()
```

Out[144]:

| | symbol | sentiment_signal | sid | asof_date | timestamp |
|---|---|---|---|---|---|
| **0** | NDAQ | 6.0 | 27026 | 2013-01-16 | 2013-01-17 |
| **1** | NDAQ | 6.0 | 27026 | 2013-01-17 | 2013-01-18 |
| **2** | NDAQ | 6.0 | 27026 | 2013-01-18 | 2013-01-19 |
| **3** | NDAQ | 6.0 | 27026 | 2013-01-19 | 2013-01-20 |
| **4** | NDAQ | 6.0 | 27026 | 2013-01-20 | 2013-01-21 |

In [147]:
```
stock.index
```

Out[147]:
```
DatetimeIndex(['2019-06-03', '2019-06-04', '2019-06-05', '2019-06-06',
               '2019-06-07', '2019-06-10', '2019-06-11', '2019-06-12',
               '2019-06-13', '2019-06-14', '2019-06-17', '2019-06-18',
               '2019-06-19', '2019-06-20', '2019-06-21', '2019-06-24',
               '2019-06-25', '2019-06-26', '2019-06-27', '2019-06-28',
               '2019-07-01'],
              dtype='datetime64[ns, UTC]', freq='C')
```

In [ ]: