In [1]:
```
from imports import *
import avg_clf_train

%matplotlib inline
```

In [2]:
```
check = True

if check == True:
    # check if last day's data is available
    print Quandl.get("YAHOO/HALO", authtoken='DVhizWXNTePyzzy1eHWR').t
```

```
             Open   High    Low  Close   Volume  Adjusted Close
Date
2015-08-13  20.07  21.49  19.93  20.73  1476100           20.73
```

In [7]:
```
download = True

tickers = ticker_list.tickers

print len(tickers), "total tickers\n"

if download == True:
    # download data
    for ticker in tickers:
        try:
            stock_df = Quandl.get("YAHOO/{}".format(ticker), authtoken
            stock_df.to_csv("quandl_data/{}.csv".format(ticker), index
        except:
            print "removed:", ticker
            tickers.remove(ticker)

print "\n", len(tickers), "available tickers:"
print tickers
```

```
56 total tickers

removed: CASI
removed: CHRS

54 available tickers:
['ABIO', 'ACOR', 'ADMA', 'AERI', 'AFFX', 'AGEN', 'APPY', 'ARDM', 'AR
IA', 'ARNA', 'ARWR', 'AXDX', 'AXGN', 'BABY', 'BASI', 'BCLI', 'BCRX',
'BGMD', 'BIIB', 'BLUE', 'BOTA', 'BRKR', 'CBLI', 'CBMG', 'CBMX', 'CBP
O', 'CGEN', 'CLDN', 'CLDX', 'CNMD', 'COHR', 'CPHD', 'CPRX', 'CRIS',
'CUTR', 'CYBX', 'CYNO', 'CYTR', 'DARA', 'ECYT', 'EXAS', 'HALO', 'IDR
A', 'INO', 'LPCN', 'MEIP', 'MNKD', 'OREX', 'PGNX', 'QLTI', 'RMTI',
'SGYP', 'THLD', 'TNXP']
```

In [9]:
```
def modify_columns(ticker, normalize):
    df = pd.read_csv("quandl_data/{}.csv".format(ticker))
    df = df.drop('Adjusted Close', axis=1)

    df['50dravg'] = pd.rolling_mean(df['Close'], window=50)
    df['200dravg'] = pd.rolling_mean(df['Close'], window=200)

    if normalize == True:
        temp_df = df['Volume']
        df = df.drop('Volume', axis=1)
        df = df.std(axis=1, ddof=0)

        df['mean'] = df.mean(axis=1)
        df['std'] = std_df

        df['Open'] = (df['Open'] - df['mean']) / df['std']
        df['High'] = (df['High'] - df['mean']) / df['std']
        df['Low'] = (df['Low'] - df['mean']) / df['std']
        df['Close'] = (df['Close'] - df['mean']) / df['std']

        df['50dravg'] = (df['50dravg'] - df['mean']) / df['std']
        df['200dravg'] = (df['200dravg'] - df['mean']) / df['std']

        df = df.drop(['mean', 'std'], axis=1)

        df['Volume'] = temp_df

    df['OC%'] = (df['Close'] / df['Open']) - 1
    df['HL%'] = (df['High'] / df['Low']) - 1

    df['ticker'] = ticker

    df['label'] = df['OC%'].shift(-1)

    return df #df.loc[500:] # remove first 500 days
```

In [11]:
```
```

```
normalize = False

scale_volume = False

binarize = True

# import data
stock_df = pd.DataFrame()
for ticker in tickers:
    if stock_df.empty:
        stock_df = modify_columns(ticker, normalize)
    else:
        stock_df = stock_df.append(modify_columns(ticker, normalize))
        #stock_df = pd.concat([stock_df, modify_columns(ticker, normali
        #stock_df = pd.concat([stock_df, modify_columns(ticker, normali

# scale volume
if scale_volume == True:
    stock_df['Volume'] = (stock_df['Volume'] - stock_df['Volume'].min()

    # log volume
    #stock_df['Volume'] = stock_df['Volume'].map(lambda x: np.log(x))

#stock_df = stock_df.drop(['Open', 'High', 'Low', 'Close'], axis=1)

stock_df = stock_df.replace([np.inf, -np.inf], np.nan)

prediction_df = stock_df.copy()
stock_df = stock_df.drop('ticker', axis=1)

stock_df = stock_df.dropna()

# remove PPS > 5.0
stock_df = stock_df[stock_df['Open'] <= 5]

# binarize labels
if binarize == True:
    stock_df['label'] = stock_df['label'].map(lambda x: 1 if x >= 0.05

print stock_df.shape
stock_df.tail()
```

```
(88588, 10)
```

Out[11]:

	Open	High	Low	Close	Volume	50dravg	200dravg	OC%	HL%	labe
385	4.27	4.40	4.22	4.25	106000	3.9192	3.81415	-0.004684	0.042654	0
386	4.21	4.29	4.20	4.22	44800	3.9280	3.83350	0.002375	0.021429	0
387	4.24	4.38	4.24	4.31	27600	3.9346	3.85195	0.016509	0.033019	1
388	4.49	5.00	4.40	4.93	261700	3.9586	3.87385	0.097996	0.136364	1
389	4.85	5.18	4.63	5.16	391900	3.9842	3.89765	0.063918	0.118790	0

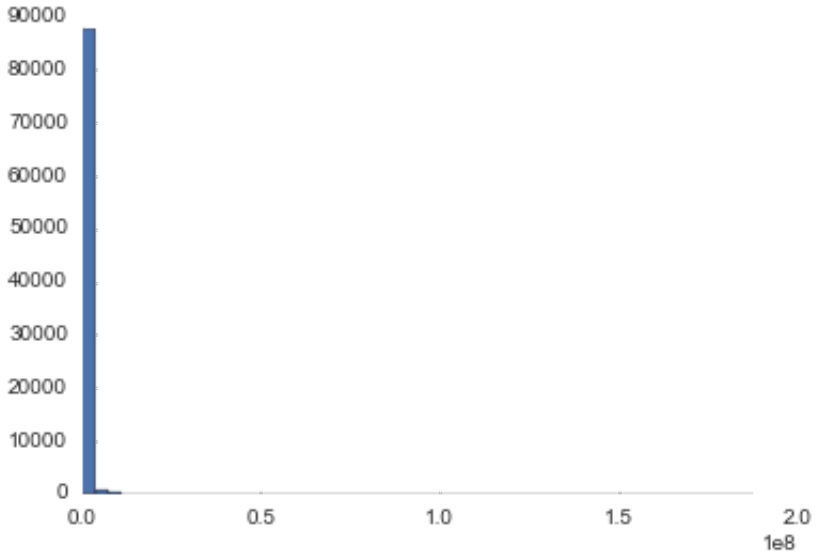
In [12]:
```
stock_df.describe()
```

Out[12]:

	Open	High	Low	Close	Volume	50d
count	88588.000000	88588.000000	88588.000000	88588.000000	8.858800e+04	885
mean	2.273619	2.349181	2.195991	2.271783	2.539001e+05	2.37
std	1.423210	1.466752	1.384322	1.427238	1.536674e+06	1.59
min	0.003003	0.003003	0.003003	0.003003	0.000000e+00	0.00
25%	1.029969	1.062471	0.999960	1.020000	2.400000e+03	1.06
50%	2.159460	2.239940	2.080000	2.150000	1.650000e+04	2.19
75%	3.490000	3.590000	3.360000	3.480000	1.105250e+05	3.53
max	5.000000	9.490000	5.000000	8.600000	1.878141e+08	19.4

In [13]:
```
stock_df.Volume.hist(bins=50)
```

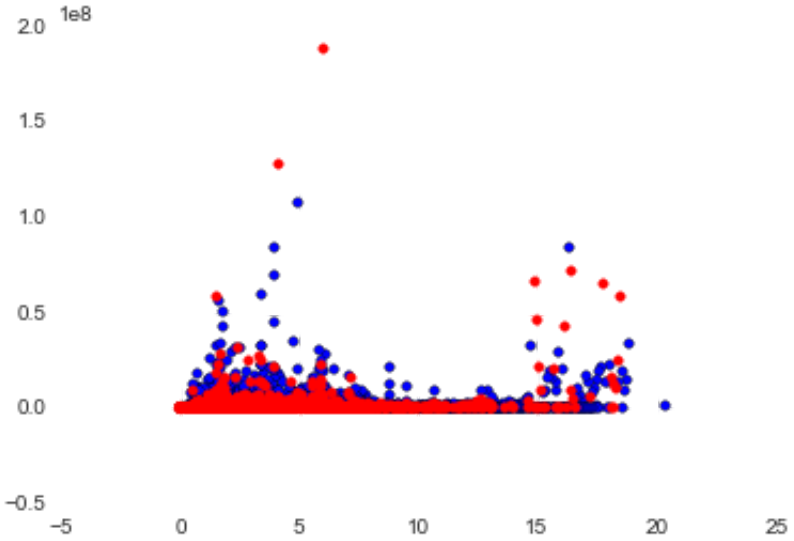
Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x110e32590>
```



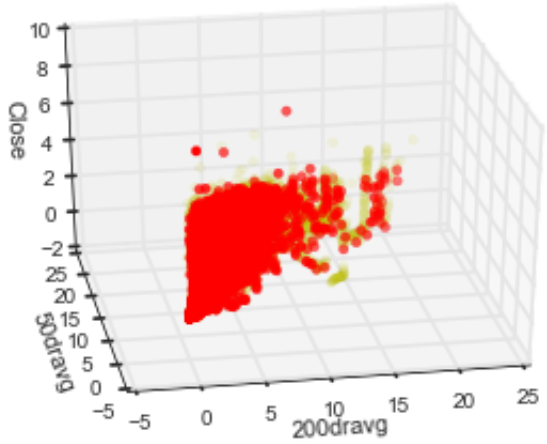
In [14]:
```
for i in xrange(len(stock_df.columns)):
    print i, stock_df.columns[i], stock_df.corr()['label'].values[i]
```

```
0 Open -0.0762917591636
1 High -0.0701949998008
2 Low -0.0806611424378
3 Close -0.0738584927026
4 Volume 0.0102289543641
5 50dravg -0.0597829711614
6 200dravg -0.0379790288168
7 OC% 0.0820010455103
8 HL% 0.119835434254
9 label 1.0
```

In [15]:
```
negative_df = stock_df[stock_df['label'] == 0]
positive_df = stock_df[stock_df['label'] == 1]

plt.scatter(negative_df['200dravg'], negative_df['Volume'])
plt.scatter(positive_df['200dravg'], positive_df['Volume'], color='r')
plt.show()
```



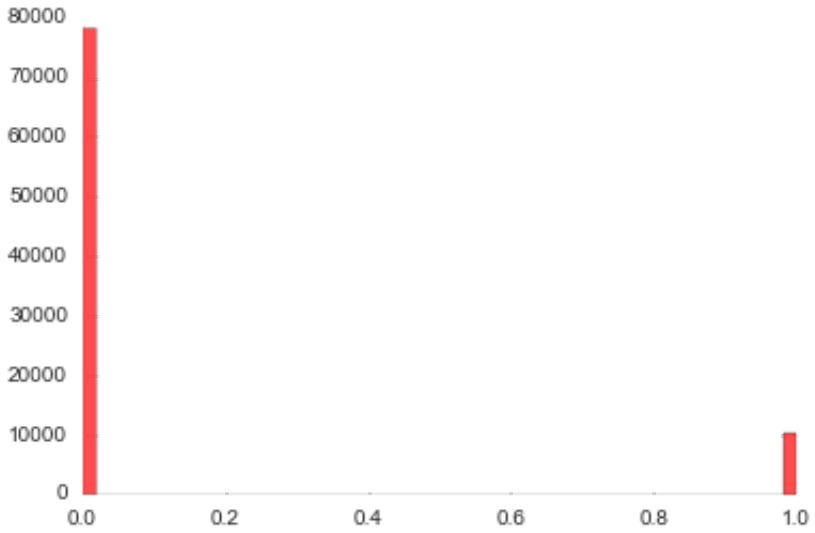
In [16]:
```
x, y, z = '200dravg', '50dravg', 'Close'

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.scatter(negative_df[x], negative_df[y], negative_df[z], alpha=0.1,
ax.scatter(positive_df[x], positive_df[y], positive_df[z], color='r')
ax.set_xlabel(x)
ax.set_ylabel(y)
ax.set_zlabel(z)
ax.view_init(azim=260)
plt.show()
```



In [17]:
```
y = stock_df['label'].values
stock_df = stock_df.drop('label', axis=1)
X = stock_df.values

print X.shape, y.shape
```

```
(88588, 9) (88588,)
```

In [18]:
```
plt.hist(y, bins=50, alpha=0.7, color='r')
plt.show()
```



In [19]:
```
y_values = np.unique(y, return_counts=True)[0]
print y_values.shape, "\n"
print y_values
```

```
(2,)

[0 1]
```

In [20]:
```
num_of_classes = np.unique(y, return_counts=True)[1]
print num_of_classes
print "percent 1: ", np.true_divide(num_of_classes[1],np.sum(num_of_cl
```

```
[78233 10355]
percent 1:  0.116889420689
```

In [21]:
```
classes_to_remove = []
for i in np.where(num_of_classes == 1)[0]:
    classes_to_remove.append(y_values[i])

print len(classes_to_remove)
print classes_to_remove[:5]
print classes_to_remove[-5:]
```

```
0
[]
[]
```

In [22]:
```
print "number of labels: ", np.unique(y, return_counts=True)[0].shape[
```

```
number of labels:  2
```

In [23]:
```
#for i in xrange(X.shape[1]):
#    plt.scatter(X[:,i], y)
#    plt.show()
```

In [24]:
```
#for i in xrange(X.shape[1]):
#    plt.hist(X[:,i])
#    plt.show()
```

In [25]:
```
skb, learners = avg_clf_train.avg_clf_train(X, y, stock_df)
```

```
Open 397.07
High 335.31
Low 443.84
Close 372.89
Volume 2.37
50dravg 249.05
200dravg 101.97
OC% 479.79
HL% 1038.5

GaussianNB()
0.871768822666

confusion matrix:
     FALSE   TRUE
FALSE [15329   303]
TRUE  [1969  117]

             precision    recall  f1-score   support

          0       0.89      0.98      0.93     15632
          1       0.28      0.06      0.09      2086
```

```
avg / total       0.81      0.87      0.83     17718


minutes for learner to run: 0.001

DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
            max_features=None, max_leaf_nodes=None, min_samples_lea
f=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            random_state=None, splitter='best')
0.812450615194

confusion matrix:
     FALSE   TRUE
FALSE [14115  1517]
TRUE  [1806  280]

             precision    recall  f1-score   support

          0       0.89      0.90      0.89     15632
          1       0.16      0.13      0.14      2086

avg / total       0.80      0.81      0.81     17718


minutes for learner to run: 0.009

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, g
amma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
0.883113218196

confusion matrix:
     FALSE   TRUE
FALSE [15630     2]
TRUE  [2069   17]

             precision    recall  f1-score   support

          0       0.88      1.00      0.94     15632
          1       0.89      0.01      0.02      2086

avg / total       0.88      0.88      0.83     17718


minutes for learner to run: 1.418

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr',
          penalty='l2', random_state=None, solver='liblinear', to
```

```
l=0.0001,
          verbose=0)
0.882605260187

confusion matrix:
     FALSE   TRUE
FALSE [15615    17]
TRUE  [2063   23]

             precision    recall  f1-score   support

          0       0.88      1.00      0.94     15632
          1       0.57      0.01      0.02      2086

avg / total       0.85      0.88      0.83     17718


minutes for learner to run: 1.421
```

In [26]:
```
#X_df = pd.DataFrame(X[:,:4])
#X_df = pd.DataFrame(X)
#X_df['labels'] = y
#sns.pairplot(X_df, hue='labels')
#plt.show()
```

In [27]:
```
#plt.hist(y, color='b', alpha=0.7)
#plt.hist(y_pred, color='y', alpha=0.7)
#plt.show()
```

In [28]:
```
#plt.scatter(np.arange(y.shape[0]), y, color='b', alpha=0.7)
#plt.scatter(np.arange(y_pred.shape[0]), y_pred, color='y', alpha=0.7)
#plt.show()
```

In [29]:
```
#y_pred - y
```

In [30]:
```
#np.sum(y)
```

In [31]:
```
#error_count = 0
#for i in xrange(len(y)):
#    if y_pred[i] != y[i]:
#        error_count += 1
#
#print error_count, " / ", len(y)
```

In [32]:
```
pred_df = prediction_df[prediction_df['label'].apply(np.isnan) == True
```

In [33]:
```
pred_X = pred_df.drop(['ticker','label'], axis=1).values
print pred_X.shape
print pred_X[0]
```

```
(54, 9)
[  9.80000000e-01   1.04000000e+00   9.50000000e-01   9.60000000e-01
   1.76100000e+05   1.06320000e+00   8.96050000e-01  -2.04081633e-02
   9.47368421e-02]
```

In [34]:
```
pred_X = skb.transform(pred_X)
print pred_X.shape
```

```
(54, 3)
```

In [35]:
```
y_predictions = []
for learner in learners:
    y_pred = learner.predict(pred_X)
    print y_pred.shape
    y_predictions.append(y_pred)
```

```
(54,)
(54,)
(54,)
(54,)
```

In [36]:
```
y_predictions
```

Out[36]:
```
[array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]),
 array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 1, 0, 0]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]),
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])]
```

In [37]:
```
y_pred_avg = np.mean(y_predictions, axis=1)
print y_pred_avg.shape
```

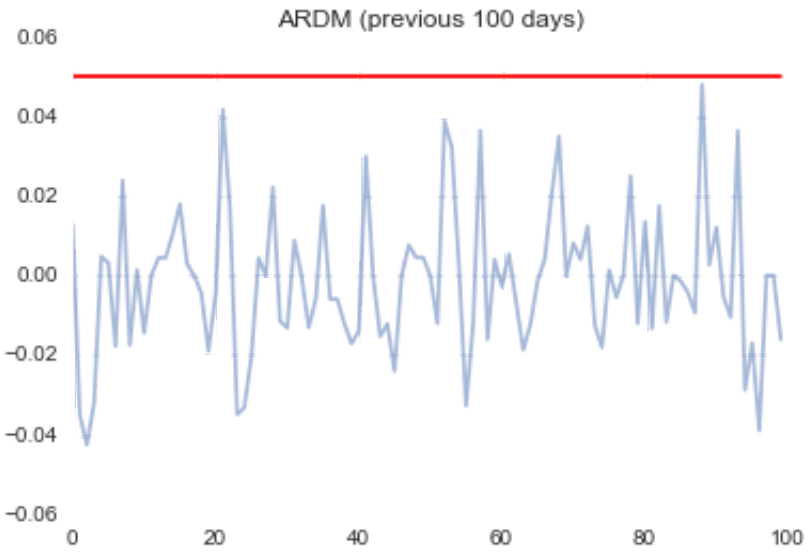
```
(4,)
```

In [44]:
```
positive_tickers = []
for i in xrange(len(tickers)):
    print i, tickers[i], y_predictions[1][i]
    if y_predictions[1][i] == 1:
        positive_tickers.append(tickers[i])
```

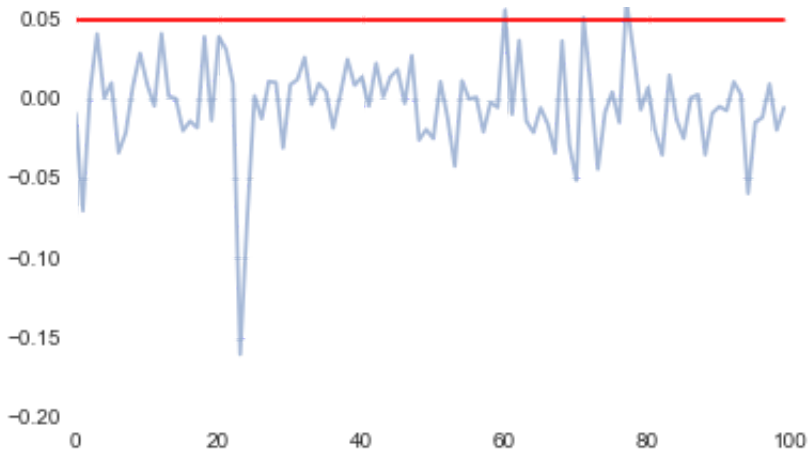
```
0 ABIO 0
1 ACOR 0
2 ADMA 0
3 AERI 0
4 AFFX 0
5 AGEN 0
6 APPY 0
7 ARDM 1
8 ARIA 0
9 ARNA 0
10 ARWR 0
11 AXDX 0
12 AXGN 0
13 BABY 0
14 BASI 0
15 BCLI 0
16 BCRX 0
17 BGMD 0
18 BIIB 0
19 BLUE 0
20 BOTA 0
21 BRKR 0
22 CBLI 0
23 CBMG 0
24 CBMX 0
25 CBPO 0
26 CGEN 0
27 CLDN 0
28 CLDX 1
29 CNMD 0
30 COHR 0
31 CPHD 0
32 CPRX 0
33 CRIS 0
34 CUTR 0
35 CYBX 0
36 CYNO 0
37 CYTR 0
38 DARA 0
39 ECYT 0
40 EXAS 0
41 HALO 0
```

```
42 IDRA 0
43 INO 0
44 LPCN 1
45 MEIP 0
46 MNKD 0
47 OREX 0
48 PGNX 0
49 QLTI 0
50 RMTI 0
51 SGYP 1
52 THLD 0
53 TNXP 0
```

In [72]:
```
for ticker in positive_tickers:

    past_days = 100

    oc = prediction_df[prediction_df['ticker'] == ticker]["OC%"][-past

    num_days = oc.shape[0]

    day_range = np.arange(num_days)

    plt.plot(day_range, oc, alpha=0.5)
    plt.plot(day_range, [0.05 for x in day_range], color='r')
    plt.title("{0} (previous {1} days)".format(ticker, num_days))
    plt.show()

    print "\t", ticker, "100-day freq probability:", np.true_divide(np
    print "~"*50, "\n"
```

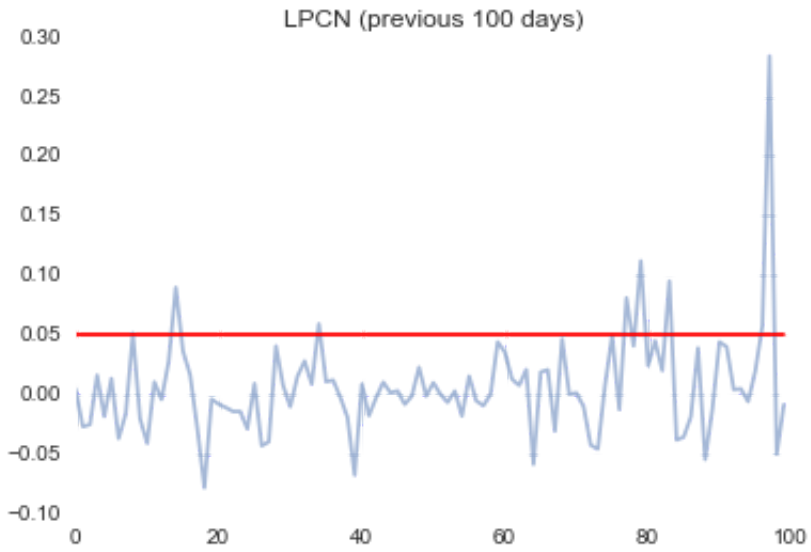



```
        ARDM 100-day freq probability: 0.0
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```



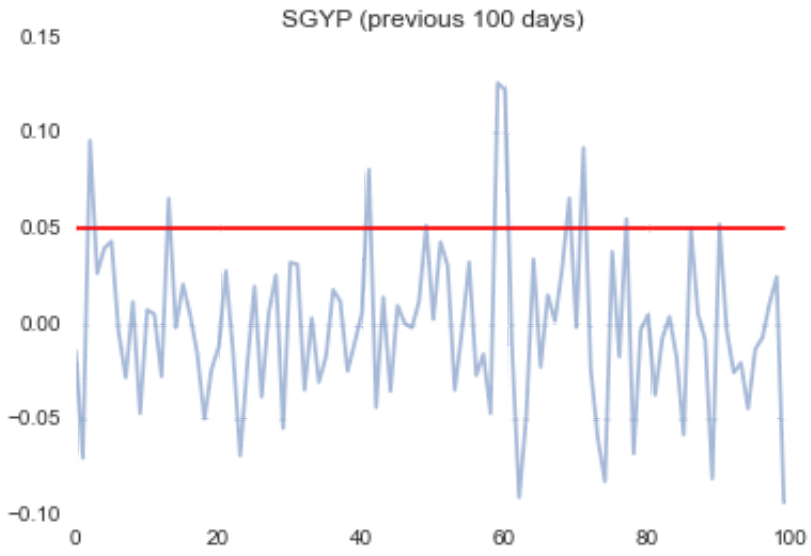
CLDX 100-day freq probability: 0.03

~~~~~~



LPCN 100-day freq probability: 0.08

~~~~~~



SGYP 100-day freq probability: 0.11