plot_hmm_stock_analysis

May 3, 2016

0.1 Hidden Markov Models on stock data by Hongyu Zhu

This script shows how to use HMMs (Discrete HMMs, Gaussian HMMs and Kalman filter) on stock price data from Yahoo! finance. The main python packages are:

hmmlearn (version 0.1.1): For HMMs. Note new version 0.2.1 uses a slightly different API. If pip is installed, do

```
pip install 'hmmlearn==0.1.1' --force-reinstall
```

to install this older version. Or you can download source files to install.

The reason I use this package is that hmmlearn is written in the sklearn fashion, which is highly compatible with numpy, cython and pandas. Also installing the python wrapper GHMM is painful on the mac.

pykalman: For Kalman filter.

numpy, scipy, matplotlib: For faster array implementations and plots.

Outline - Sanity check with HW - Gaussian HMMs - Discrete HMMs - Kalman Filter - A case study

```
In [4]: %matplotlib inline
        from __future__ import division
        import datetime
        import numpy as np
        from matplotlib import cm, pyplot as plt
        plt.rcParams['figure.figsize'] = (16, 10)
        from matplotlib.dates import YearLocator, MonthLocator
        import matplotlib.dates as mdates
        import matplotlib.gridspec as gridspec
        from pykalman import KalmanFilter
        import hmmlearn
        assert hmmlearn.__version__== '0.1.1'
        import hmmlearn.hmm as hmm
        from scipy import poly1d
        try:
            from matplotlib.finance import quotes_historical_yahoo_ochl
        except ImportError:
            # For Matplotlib prior to 1.5.
            from matplotlib.finance import \
                quotes_historical_yahoo as quotes_historical_yahoo_ochl

        np.random.seed(36723)  # set seed number
```

To start with, let's first check whether this package is consistent with our results in HW 1.

In HW 1 **Q2**, we know the most likely state for observation 8 is 'A' given

["A" "A" "A" "B" "A" "B" "A" "B" "A" "A" "B" "B" "A" "A" "B"]

Also in **Q4**, the Viterbi algorithm would give state sequence

["B" "B" "B" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"]

given the obervation
["B" "B" "B" "A" "A" "B" "A" "A" "B" "A" "A" "B" "B" "B" "A"]
Let's check:

In [5]:
```
symbols = ['A', 'B']
transmat = np.array([[.8, .2], [.3, .7]])
emitmat = np.array([[.6, .4], [.35, .65]])
startprob = np.array([.5, .5])

h = hmm.MultinomialHMM(n_components=2)
h.startprob_ = startprob
h.transmat_ = transmat
h.emissionprob_ = emitmat

logprob, seq = h.decode([1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0])

print symbols[h.predict([0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1])[7]]
print ", ".join(map(lambda x: symbols[x], seq))
```

```
A
B, B, B, A, A, A, A, A, A, A, A, A, A, A, A
```

The answers match!

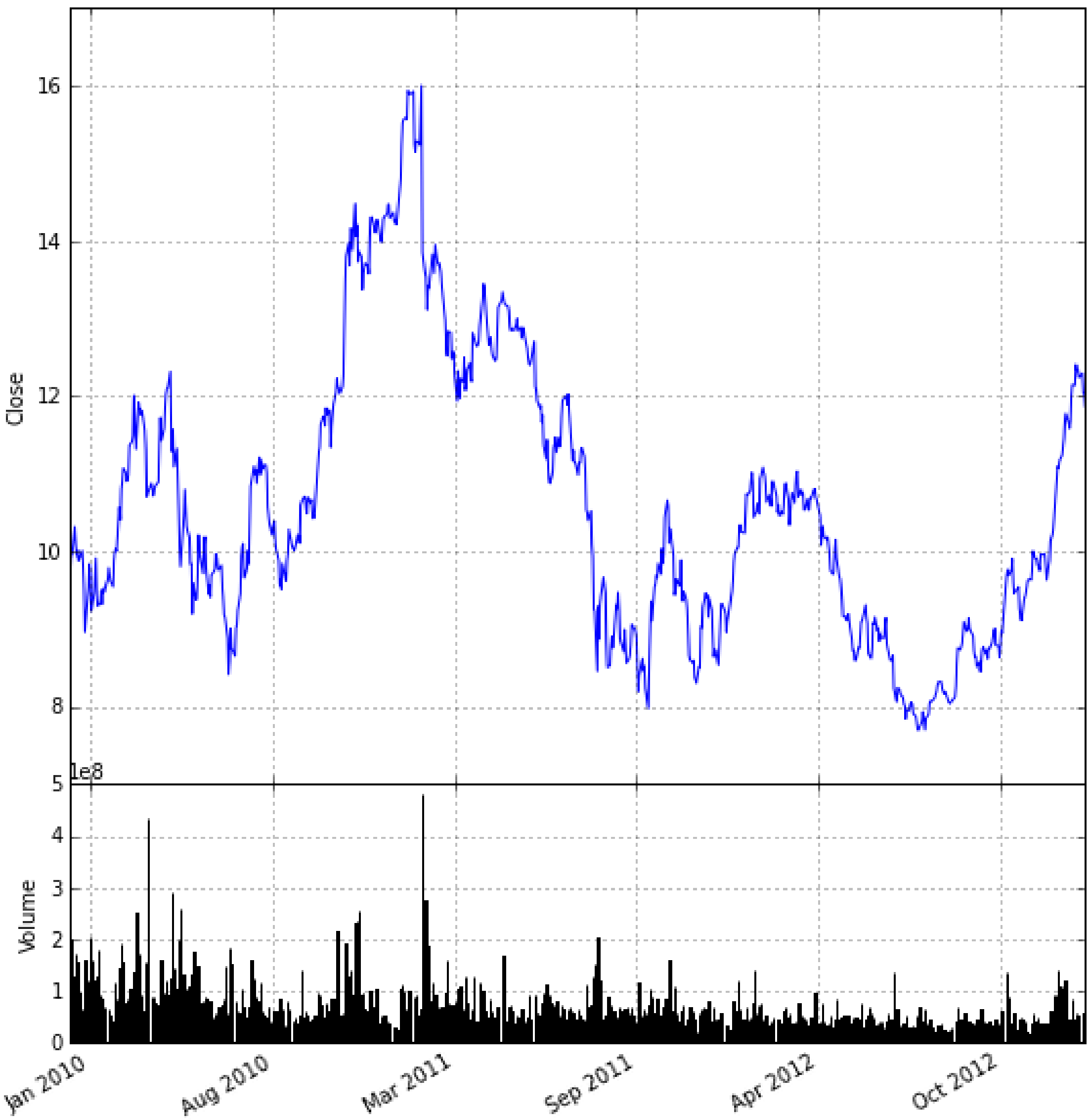
Get quotes from Yahoo! finance (note you don't have to download data manually). The returning values are dates, return and close price of the requested stock).

In [6]:
```
def quoteYahoo(stock, startdate, enddate):
    """
    Get quotes from Yahoo! finance and plot the close value/volume vs date.

    Parameters
    ----------
    stock: string
        The stock name.
    startdate: datetime.date object
        The request start date.
    enddate: datetime.date object
        The request end date.

    Returns
    -------
    dates: array of int
        The offset of dates based on the definition of the
        \proleptic Gregorian" calendar
    diff: array of double
        Return.
    close_v: array of double
        Close value.
    """

    quotes = quotes_historical_yahoo_ochl(
                                            stock, startdate, enddate)

    # Unpack quotes
    dates = np.array([q[0] for q in quotes], dtype=np.int32)
```

```
    close_v = np.array([q[2] for q in quotes])
    volume = np.array([q[5] for q in quotes])[1:]

    diff = (close_v[1:]-close_v[:-1])/close_v[:-1]
    dates = dates[1:]
    close_v = close_v[1:]

    fig = plt.figure()
    gs = gridspec.GridSpec(2, 1, height_ratios=[3,1])
    gs.update(left=0.05, right=0.48, hspace=0.0)

    ax1 = plt.subplot(gs[0])
    ax1.plot(dates, close_v)
    ax1.set_xlim([dates.min(), dates.max()])
    ax1.set_ylabel('Close')
    ax1.grid(True)
    ax1.set_xticklabels([])

    ax2 = plt.subplot(gs[1])
    ax2.bar(dates, volume)
    ax2.set_xlim([dates.min(), dates.max()])
    ax2.set_ylabel('Volume')
    ax2.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
    ax2.grid(True)

    fig.autofmt_xdate()

    plt.show()

    return dates, diff, close_v
```

Use Ford as an example. The upper panel shows the close value vs time and the lower panel shows the volume vs time.

```
In [7]: stock, startdate, enddate = "F", datetime.date(2010, 1, 1), datetime.date(2013, 1, 27)
        dates, diff, close_v = quoteYahoo(stock, startdate, enddate)
```



Gaussian HMM (GHMM):

```
In [8]: def GaussianHMM_time(dates, query, **kwargs):
            """
            Gaussian HMMs on time series.

            Parameters
            ----------
            dates: array of int
                The offset of dates based on the definition of the
                \proleptic Gregorian" calendar
            query: array of double
                The query on the time series (price or return)
            kwargs: See args of hmm.GaussianHMM
                n_components=1,
                covariance_type='diag',
```

```
        min_covar=0.001,
        startprob_prior=1.0,
        transmat_prior=1.0,
        means_prior=0,
        means_weight=0,
        covars_prior=0.01,
        covars_weight=1,
        algorithm='viterbi',
        random_state=None,
        n_iter=10,
        tol=0.01,
        verbose=False,
        params='stmc',
        init_params='stmc'

    Returns
    -------
    model: hmmlearn.hmm.GaussianHMM object
    hidden_states: array of int
        Hidden states of each time stamp.
    """

    X = np.column_stack([dates, query])
    # Make an HMM instance and execute fit
#     model = hmm.GaussianHMM(n_components=5, covariance_type="diag", n_iter=1000).fit([X])
    model = hmm.GaussianHMM(**kwargs).fit([X])
    # Predict the optimal sequence of internal hidden state
    hidden_states = model.predict(X)

    a, b = model.sample(n=1000)
    ind = np.int32(np.round(a[:,0]))
    samplei, samplev = [], []
    for i in xrange(ind.min(), ind.max()):
        indi = (ind == i)
        if any(indi):
            predict_v = a[indi,1].mean()
            samplei.append(i)
            samplev.append(predict_v)
    sample = np.column_stack((samplei, samplev))

    # Plot
    fig, ax = plt.subplots()
    ax.plot(X[:,0], X[:,1], 'b', label='Market')
    ax.plot(sample[:,0], sample[:,1], 'r', label='Prediction')
    ax.scatter(model.means_[:,0], model.means_[:,1], c='g', s=300, label='Means')
    ax.legend()
    ax.set_xlim([dates.min(), dates.max()])
    ax.grid(True)
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
    fig.autofmt_xdate()
    plt.show()

    return model, hidden_states
```

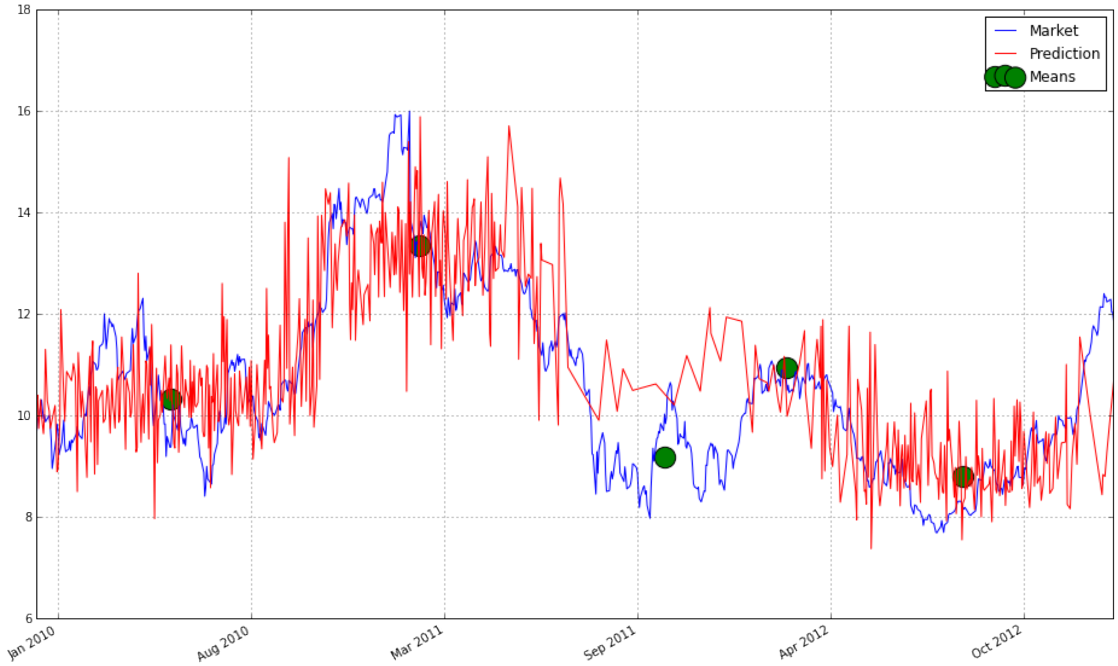
We anaylze the time series on both close values and returns. In the plot, the blue curve is the actually market value and the red curve is our prediction (sampling from the model). The green circles show the means of each hidden states. We can see the means captures the significant properties of the market values. We can also try different number of hidden states here. Right now we are using

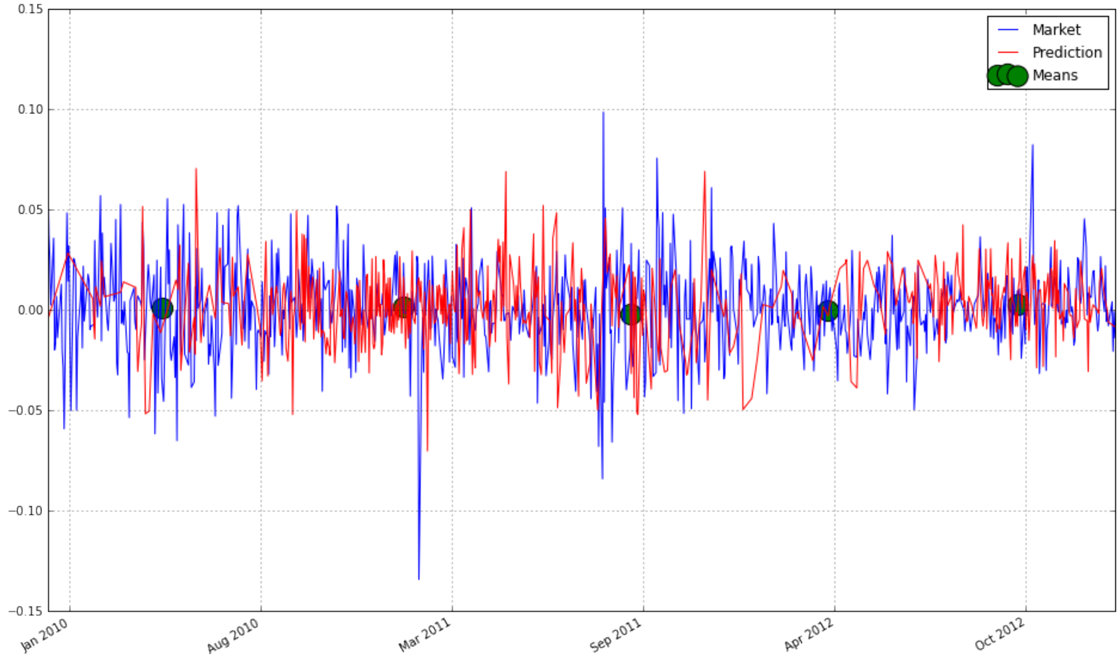
```
n_components=5
```

From the references, these 5 hidden states are typically considered as ['huge increse','small increse','steady','small decrease','huge decrease'].

```
In [9]: close_model, close_hs = GaussianHMM_time(dates, close_v, n_components=5, covariance_type="diag"
        return_model, return_hs = GaussianHMM_time(dates, diff, n_components=5, covariance_type="diag",
```

```
/Users/Hongyu/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/matplotlib/collections.py:
  if self._edgecolors == str('face'):
```





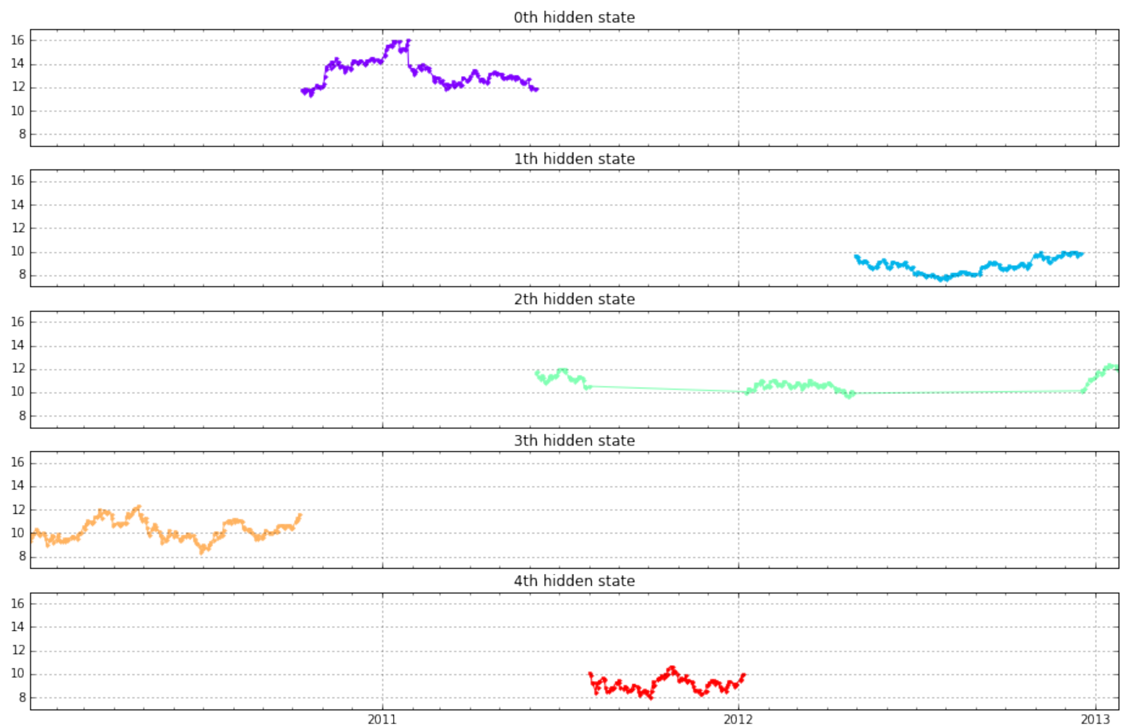
In [10]:
```
def GausianHidden(model, hidden_states):
    """
    Analyze the hidden states of GHMMs.

    Parameters
    ----------
    model: hmmlearn.hmm.GaussianHMM object
    hidden_states: array of int
        Hidden states of each time stamp.
    """
    print "Means and vars of each hidden state"
    for i in range(model.n_components):
        print "{0}th hidden state".format(i)
        print "mean = ", model.means_[i]
        print "var = ", np.diag(model.covars_[i])

    fig, axs = plt.subplots(model.n_components, sharex=True, sharey=True)
    colours = cm.rainbow(np.linspace(0, 1, model.n_components))
    for i, (ax, colour) in enumerate(zip(axs, colours)):
        # Use fancy indexing to plot data in each state.
        mask = hidden_states == i
        ax.plot_date(dates[mask], close_v[mask], ".-", c=colour)
        ax.set_title("{0}th hidden state".format(i))

        # Format the ticks.
        ax.xaxis.set_major_locator(YearLocator())
        ax.xaxis.set_minor_locator(MonthLocator())

        ax.grid(True)
```

```
plt.show()
```

We can also analyze the hidden states for close values and returns. It seems to me that each hidden state governs a specific regime.

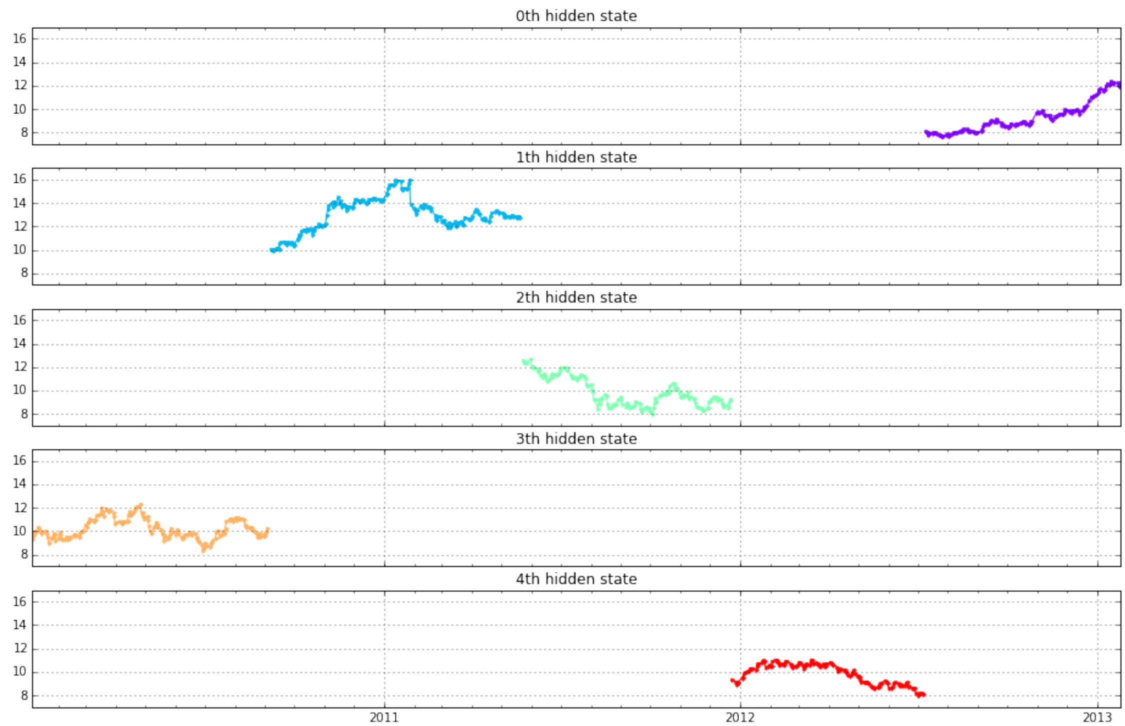
```
In [11]: GausianHidden(close_model, close_hs)
         GausianHidden(return_model, return_hs)
```

```
Means and vars of each hidden state
0th hidden state
mean =  [  7.34175041e+05   1.33375298e+01]
var =  [  4.72924767e+03   1.14725817e+00]
1th hidden state
mean =  [  7.34737201e+05   8.80629808e+00]
var =  [  4.51869917e+03   3.88711831e-01]
2th hidden state
mean =  [  7.34554363e+05   1.09266307e+01]
var =  [  3.42121926e+04   3.85955522e-01]
3th hidden state
mean =  [  7.33916895e+05   1.03189110e+01]
var =  [  6.56234405e+03   6.68369728e-01]
4th hidden state
mean =  [  7.34428942e+05   9.17065254e+00]
var =  [  2.06326985e+03   3.03271669e-01]
```




```
Means and vars of each hidden state
0th hidden state
```

```
mean =  [  7.34791363e+05   2.88167703e-03]
var =  [  3.42597238e+03   2.93305672e-04]
1th hidden state
mean =  [  7.34149028e+05   1.52899416e-03]
var =  [  5.81813842e+03   4.51254234e-04]
2th hidden state
mean =  [  7.34386519e+05  -1.93843868e-03]
var =  [  3.70137764e+03   8.11954243e-04]
3th hidden state
mean =  [  7.33897370e+05   1.09371554e-03]
var =  [  4.87130318e+03   7.62881496e-04]
4th hidden state
mean =  [  7.34592766e+05  -6.11694720e-04]
var =  [  3.27506449e+03   3.55889558e-04]
```



Discrete (Multinomial) HMMs

```
In [12]: def DiscreteHMM_time(dates, query, **kwargs):
             """
             Discrete(Multinomial) HMMs on time series.

             Parameters
             ----------
             dates: array of int
                 The offset of dates based on the definition of the
                 \proleptic Gregorian" calendar
             query: array of int
                 The binary query on the time series (price or return)
             kwargs: See args of hmm.GaussianHMM
```

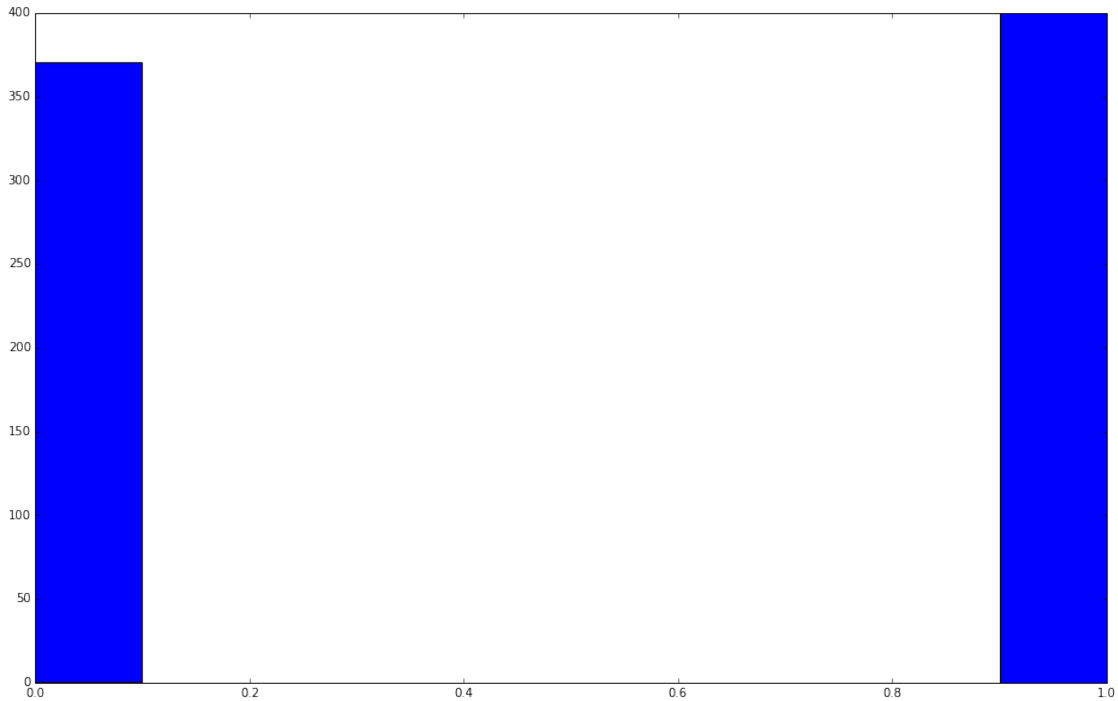
```
                n_components=1,
                covariance_type='diag',
                min_covar=0.001,
                startprob_prior=1.0,
                transmat_prior=1.0,
                means_prior=0,
                means_weight=0,
                covars_prior=0.01,
                covars_weight=1,
                algorithm='viterbi',
                random_state=None,
                n_iter=10,
                tol=0.01,
                verbose=False,
                params='stmc',
                init_params='stmc'

            Returns
            -------
            model: hmmlearn.hmm.GaussianHMM object
            hidden_states: array of int
                Hidden states of each time stamp.
            """
            X = np.int32(query)

            # Make an HMM instance and execute fit
            model = hmm.MultinomialHMM(**kwargs)
            model.fit([X])

            # Predict the optimal sequence of internal hidden state
            hidden_states = model.predict(X)

            a, b = model.sample(n=dates.shape[0])

            sample = np.column_stack((dates, a))
            fig, ax = plt.subplots()
            ax.plot(dates, query, 'b', label='Market')
            ax.plot(sample[:,0], sample[:,1], 'r', label='Prediction')
            ax.legend()
            ax.set_xlim([dates.min(), dates.max()])
            ax.grid(True)
            ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
            fig.autofmt_xdate()
            plt.show()

            return model, b
```

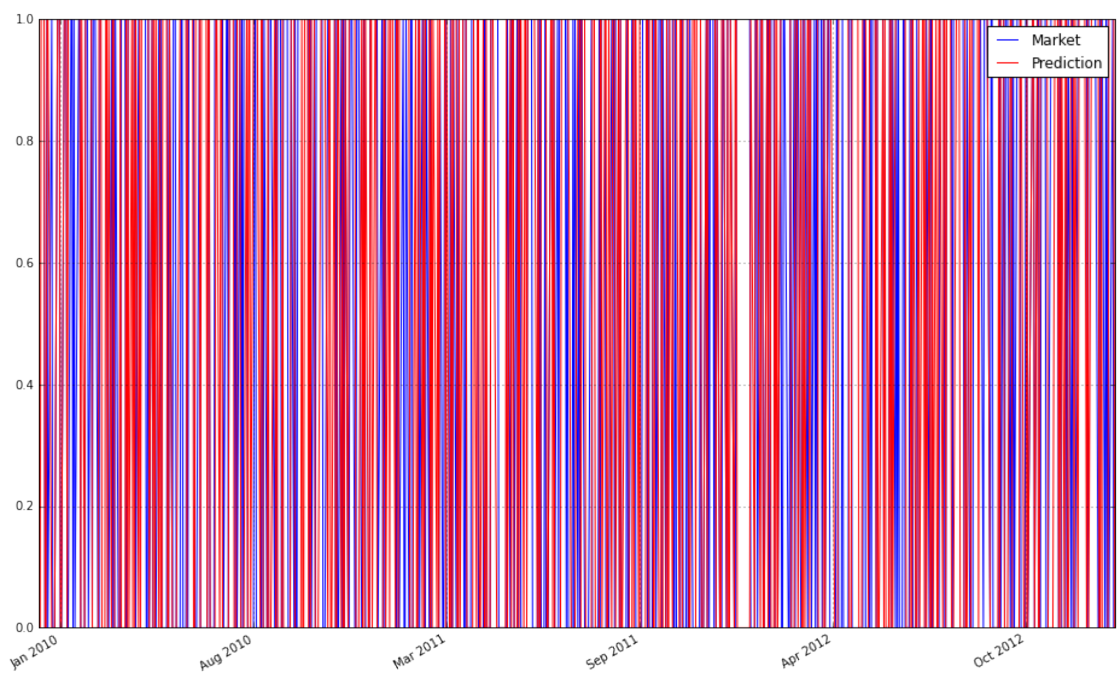
We first turn the returns into discrete integers (in this case, binary 0 if drop and 1 if increase). The histogram shows the distributions of binary returns. Like GHMMs, the number of hidden states can also be adjusted by tuning

```
n_components=5
```

```
In [13]: diff_binary = np.zeros_like(diff)
         for i in xrange(len(diff)):
```

```
    if diff[i] >= 0:
        diff_binary[i] = 1
plt.hist(diff_binary)
returnb_model, returnb_hs = DiscreteHMM_time(dates, diff_binary, n_components=5)
```

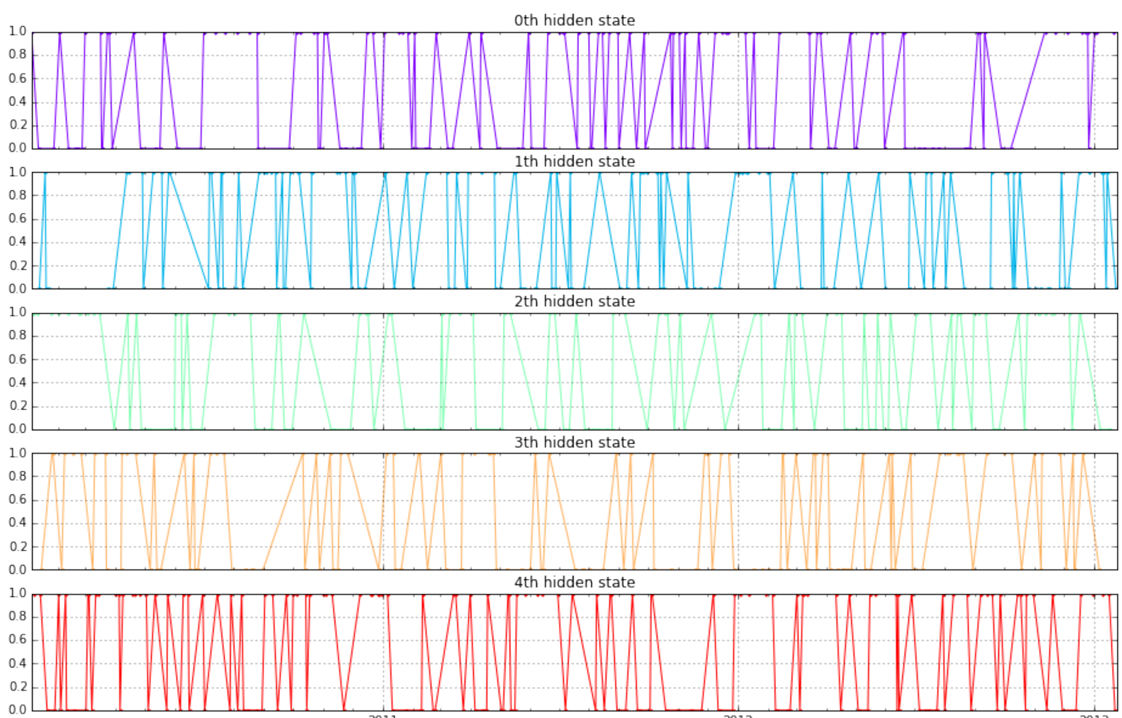




In [14]:
```
def DiscreteHidden(model, hidden_states):
    """
    Analyze the hidden states of DHMMs.

    Parameters
    ----------
    model: hmmlearn.hmm.GaussianHMM object
    hidden_states: array of int
        Hidden states of each time stamp.
    """
    fig, axs = plt.subplots(model.n_components, sharex=True, sharey=True)
    colours = cm.rainbow(np.linspace(0, 1, model.n_components))
    for i, (ax, colour) in enumerate(zip(axs, colours)):
        # Use fancy indexing to plot data in each state.
        mask = hidden_states == i
        ax.plot_date(dates[mask], diff_binary[mask], ".-", c=colour)
        ax.set_title("{0}th hidden state".format(i))

        # Format the ticks.
        ax.xaxis.set_major_locator(YearLocator())
        ax.xaxis.set_minor_locator(MonthLocator())

        ax.grid(True)

    plt.show()
```

The hidden state analysis could be conducted as what we did in GHMMs. Note here the different regimes are interweaved which are not easy to distinguish.

In [15]:
```
DiscreteHidden(returnb_model, returnb_hs)
```

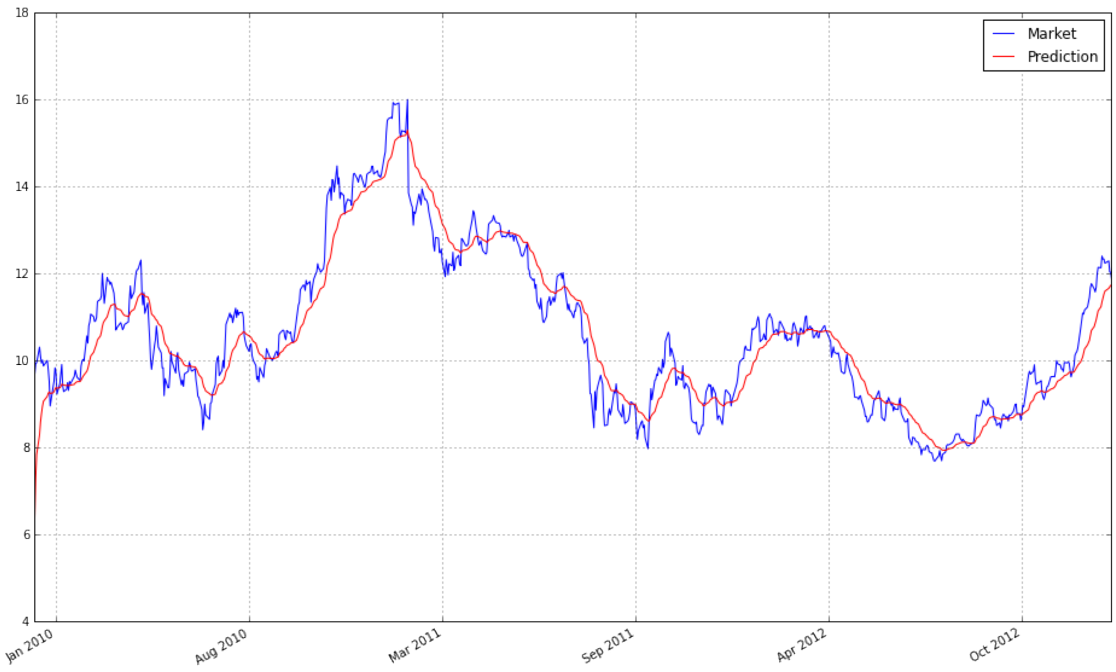


Kalman Filter

```
In [16]: def Kalman_time(dates, query):
             # Construct a Kalman filter, change the params here
             kf = KalmanFilter(transition_matrices = [1],
                               observation_matrices = [1],
                               initial_state_mean = 0,
                               initial_state_covariance = 1,
                               observation_covariance=1,
                               transition_covariance=.01)

             state_means, _ = kf.filter(query)

             fig, ax = plt.subplots()
             ax.plot(dates, query, 'b', label='Market')
             ax.plot(dates, state_means, 'r', label='Prediction')
             ax.legend()
             ax.set_xlim([dates.min(), dates.max()])
             ax.grid(True)
             ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
             fig.autofmt_xdate()
             plt.show()
```

```
In [17]: Kalman_time(dates, close_v)
```

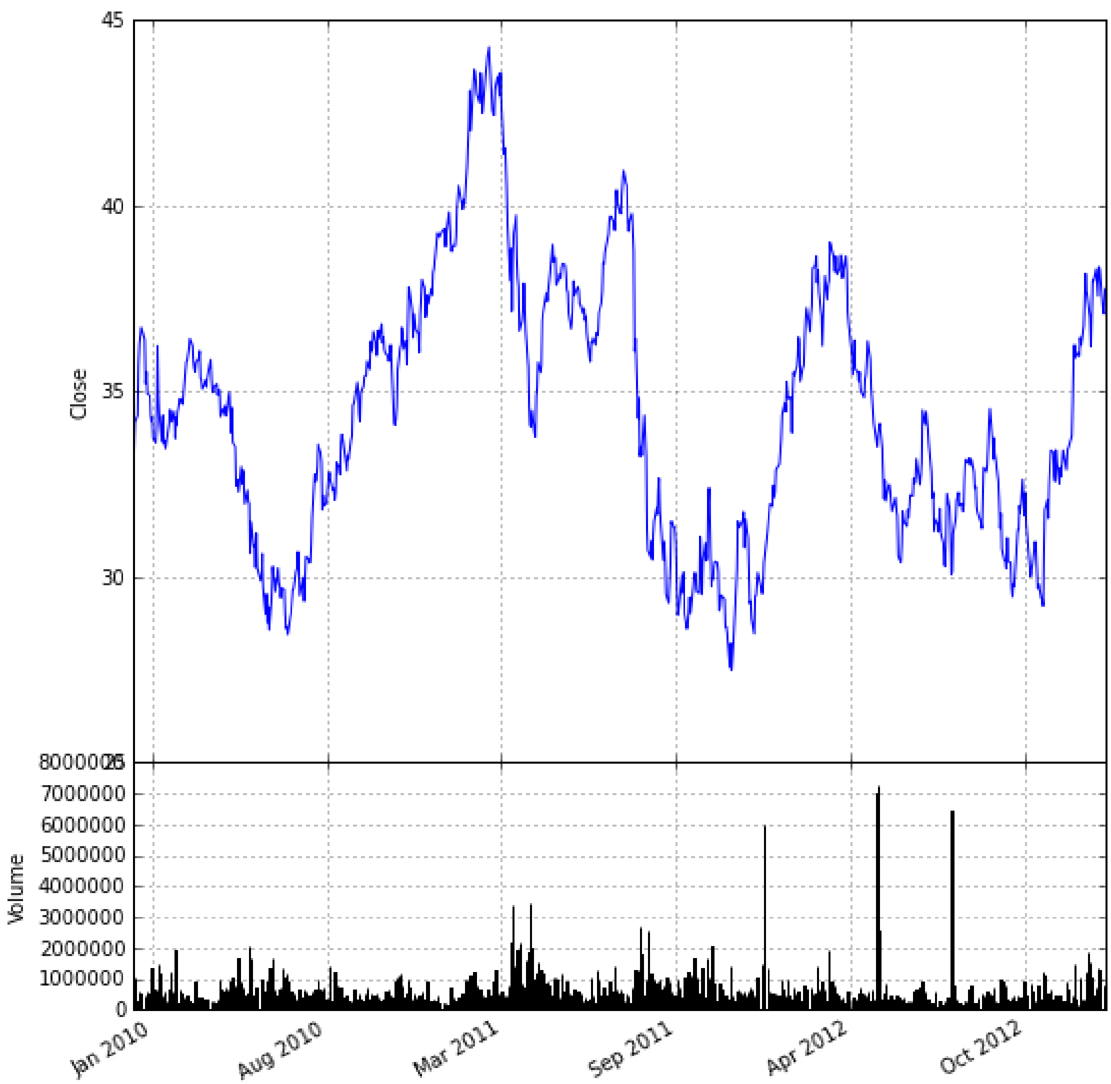
```
/Users/Hongyu/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/scipy/linalg/basic.py:884:
  warnings.warn(mesg, RuntimeWarning)
```

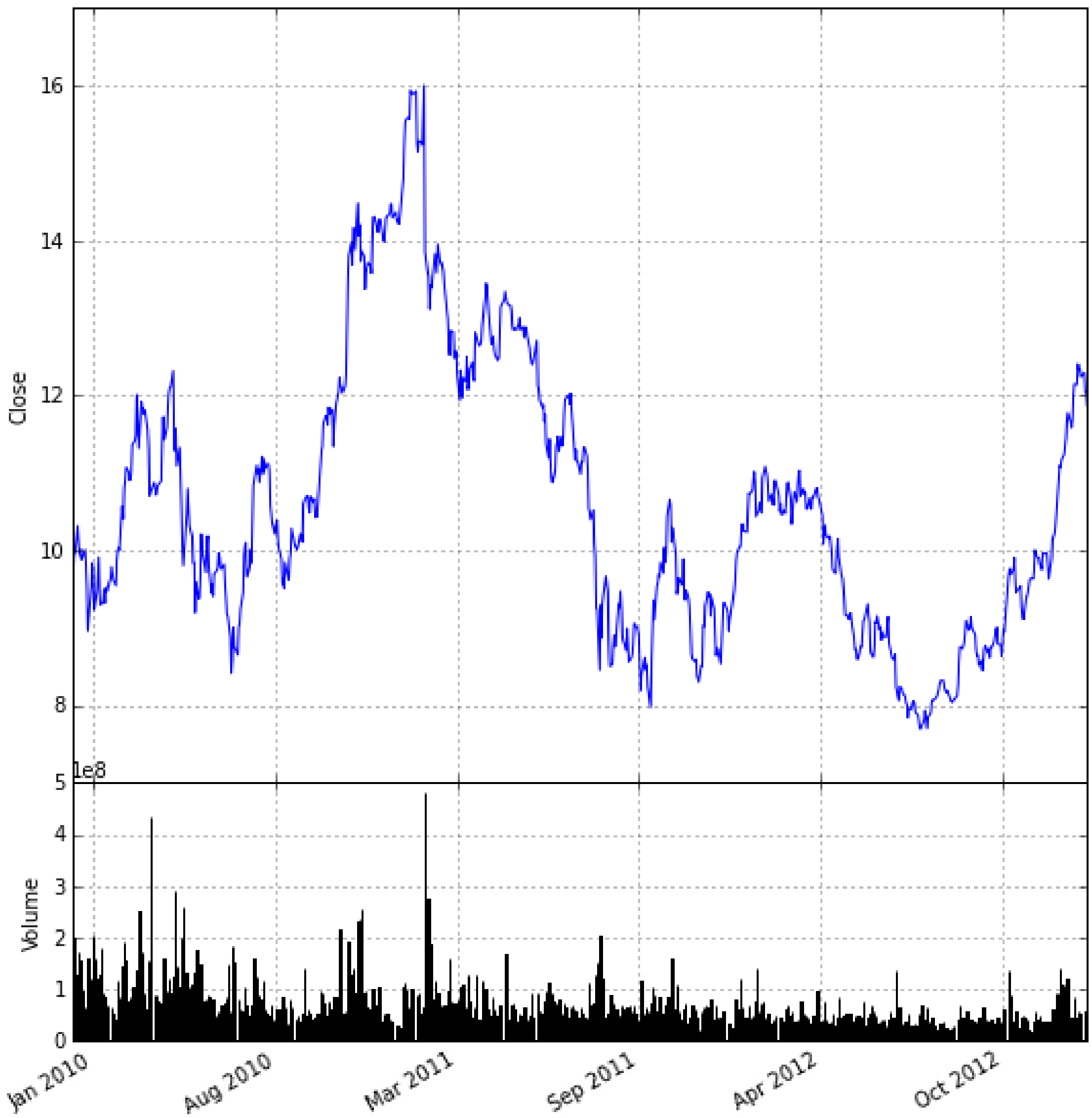


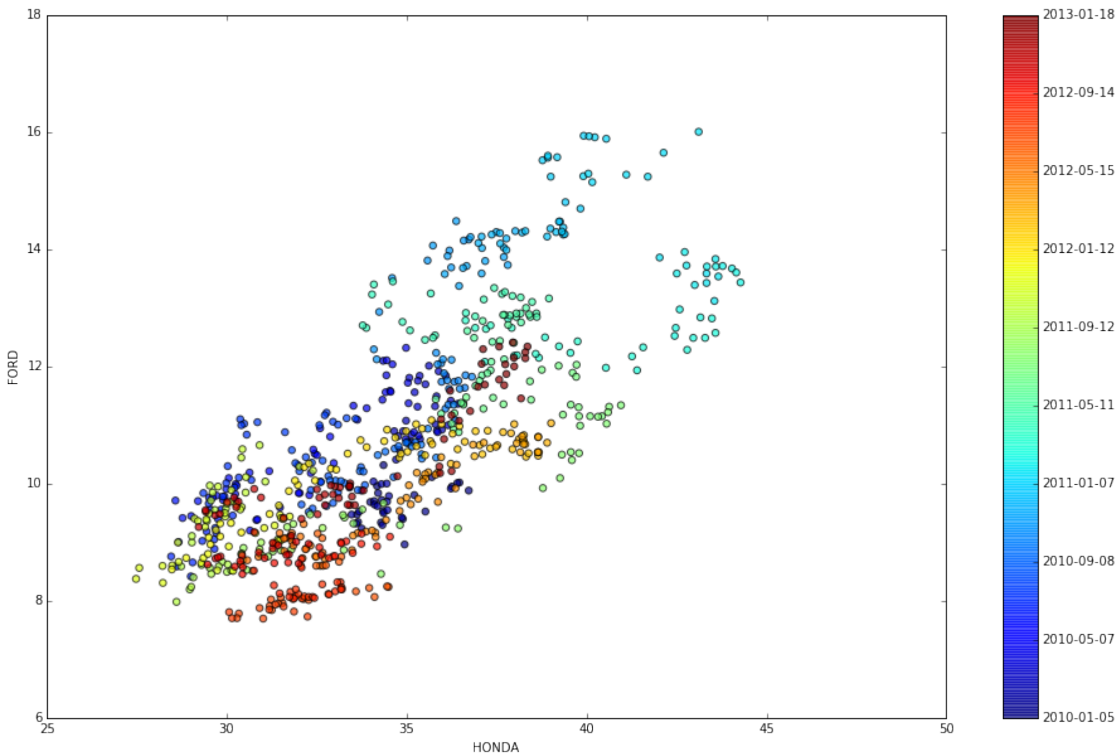
A case study: correlation between Honda and Ford: Adapted from https://www.quantopian.com/posts/quantopian-lecture-series-kalman-filters

Let's firstly get data of Honda and Ford. The third scatter plot roughly shows the correlation of close values between Honda and Ford during each time period. We can see they are positively correlated.

In [18]:
```
startdate, enddate = datetime.date(2010, 1, 1), datetime.date(2013, 1, 27)
dates_hmc, diff_hmc, close_v_hmc = quoteYahoo("HMC", startdate, enddate)
dates_ford, diff_ford, close_v_ford = quoteYahoo("F", startdate, enddate)
# Plot data and use colormap to indicate the date each point corresponds to
cm = plt.get_cmap('jet')
colors = np.linspace(0.1, 1, len(close_v_hmc))
sc = plt.scatter(close_v_hmc, close_v_ford, s=30, c=colors, cmap=cm, edgecolor='k', alpha=0.7)
cb = plt.colorbar(sc)
# cb.ax.yaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
cb.ax.set_yticklabels([datetime.datetime.fromordinal(p).strftime('%Y-%m-%d') for p in dates_hm
plt.xlabel('HONDA')
plt.ylabel('FORD');
```




Close
16
14
12
10
8
1e8
5
4
3
2
1
0
Volume
Jan 2010
Aug 2010
Mar 2011
Sep 2011
Apr 2012
Oct 2012



Build a Kalman Filter. Let's figure out the inputs to our Kalman filter. We'll say that the state of our system is the line that the observations are with parameters α and β. Our inital guesses for these parameters is (0,0), with a covariance matrix (which describes the error of our guess) of all ones.

To get from the state of our system to an observation, we apply a linear model by dotting the state (β, α) with $(x_i, 1)$ to get $\beta x_i + \alpha \approx y_i$, so our observation matrix is just a column of 1s glued to x. We assume that the variance of our observations y is 2. Now we are ready to use our observations of y to evolve our estimates of the parameters α and β.

```
In [19]: delta = 1e-3
         trans_cov = delta / (1 - delta) * np.eye(2) # How much random walk wiggles
         obs_mat = np.expand_dims(np.vstack([[close_v_hmc], [np.ones(len(close_v_hmc))]]).T, axis=1)

         kf = KalmanFilter(n_dim_obs=1, n_dim_state=2, # y is 1-dimensional, (alpha, beta) is 2-dimensi
                           initial_state_mean=[0,0],
                           initial_state_covariance=np.ones((2, 2)),
                           transition_matrices=np.eye(2),
                           observation_matrices=obs_mat,
                           observation_covariance=2,
                           transition_covariance=trans_cov)

         state_means, state_covs = kf.filter(close_v_ford)
```

The slope and intercept are α and β over time.

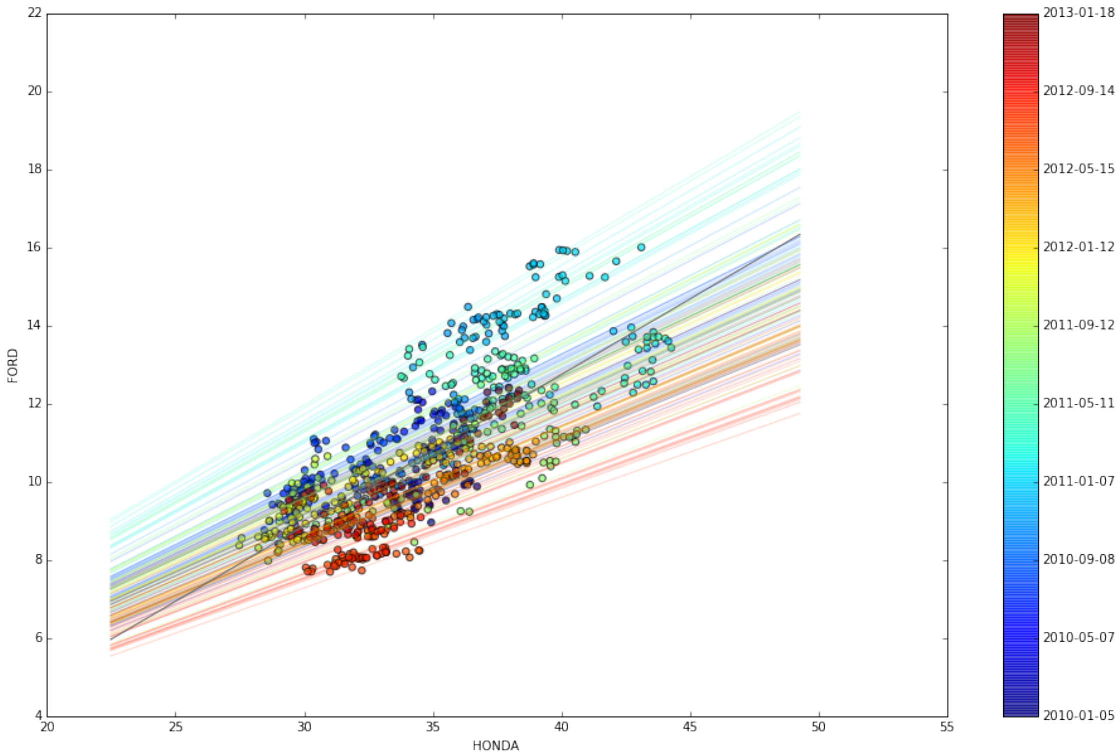
```
In [20]: fig, axarr = plt.subplots(2, sharex=True)
         axarr[0].plot(dates, state_means[:,0], label='slope')
         axarr[0].legend()
         axarr[0].grid(True)
```

```
axarr[1].plot(dates, state_means[:,1], label='intercept')
axarr[1].legend()
axarr[1].set_xlim([dates.min(), dates.max()])
axarr[1].grid(True)
axarr[1].xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
fig.autofmt_xdate()
plt.tight_layout()
plt.show()
```

To visualize how the system evolves through time, we plot every fifth state (linear model α, β) below. The black line is returned by using least-squares regression on the full dataset for comparison.

```
In [21]: # Plot data and use colormap to indicate the date each point corresponds to
         cm = plt.get_cmap('jet')
         colors = np.linspace(0.1, 1, len(close_v_hmc))
         sc = plt.scatter(close_v_hmc, close_v_ford, s=30, c=colors, cmap=cm, edgecolor='k', alpha=0.7)
         cb = plt.colorbar(sc)
         # cb.ax.yaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
         cb.ax.set_yticklabels([datetime.datetime.fromordinal(p).strftime('%Y-%m-%d') for p in dates_hmc

         # Plot every fifth line
         step = 5
         xi = np.linspace(close_v_hmc.min()-5, close_v_hmc.max()+5, 2)
         colors_l = np.linspace(0.1, 1, len(state_means[::step]))
         for i, beta in enumerate(state_means[::step]):
             plt.plot(xi, beta[0] * xi + beta[1], alpha=.2, lw=1, c=cm(colors_l[i]))

         # Plot the OLS regression line
         plt.plot(xi, poly1d(np.polyfit(close_v_hmc, close_v_ford, 1))(xi), '0.4')
```

```
plt.xlabel('HONDA')
plt.ylabel('FORD')

plt.show()
```

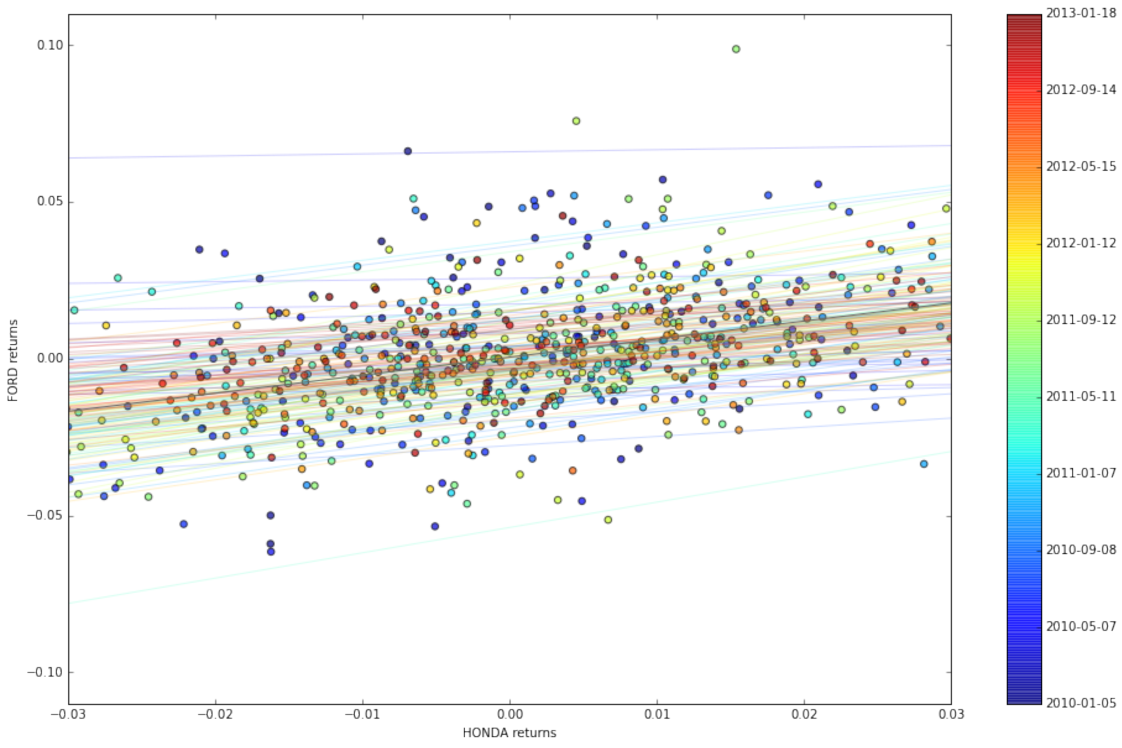


We can apply the same method on returns.

In [22]:
```
# Get returns from pricing data
x_r = diff_hmc
y_r = diff_ford

# Run Kalman filter on returns data
delta_r = 1e-2
trans_cov_r = delta_r / (1 - delta_r) * np.eye(2) # How much random walk wiggles
obs_mat_r = np.expand_dims(np.vstack([[x_r], [np.ones(len(x_r))]]).T, axis=1)
kf_r = KalmanFilter(n_dim_obs=1, n_dim_state=2, # y_r is 1-dimensional, (alpha, beta) is 2-dim
                  initial_state_mean=[0,0],
                  initial_state_covariance=np.ones((2, 2)),
                  transition_matrices=np.eye(2),
                  observation_matrices=obs_mat_r,
                  observation_covariance=.01,
                  transition_covariance=trans_cov_r)
state_means_r, _ = kf_r.filter(y_r)

# Plot data points using colormap
colors_r = np.linspace(0.1, 1, len(x_r))
sc = plt.scatter(x_r, y_r, s=30, c=colors_r, cmap=cm, edgecolor='k', alpha=0.7)
cb = plt.colorbar(sc)
```

```
cb.ax.set_yticklabels([datetime.datetime.fromordinal(p).strftime('%Y-%m-%d') for p in dates_hm

# Plot every fifth line
step = 5
xi = np.linspace(x_r.min()-4, x_r.max()+4, 2)
colors_l = np.linspace(0.1, 1, len(state_means_r[::step]))
for i, beta in enumerate(state_means_r[::step]):
    plt.plot(xi, beta[0] * xi + beta[1], alpha=.2, lw=1, c=cm(colors_l[i]))

# Plot the OLS regression line
plt.plot(xi, poly1d(np.polyfit(x_r, y_r, 1))(xi), '0.4')

# Adjust axes for visibility
plt.axis([-0.03,0.03,-0.11, 0.11])

# Label axes
plt.xlabel('HONDA returns')
plt.ylabel('FORD returns')
```

Out[22]: <matplotlib.text.Text at 0x10c095350>



In [ ]: