Instead of rehashing how ARCH and GARCH models work, I thought I'd present a practical application of a GARCH model.

Credits: While researching this project, I came across an amazing Medium article by Quantstart. The concept and the vast majority of the code below were borrowed from this article.

Today, we'll be developing a trading strategy using ARIMA and GARCH. We'll use the models to forecast the next day's return for the SPY ETF in this case (but we could apply this to any instrument).

Basic overview:

- Fit an ARIMA model to log returns
- Fit GARCH model to residuals of ARIMA model using best ARIMA model parameters
- Use GARCH model to forecast the next day's residual. Assuming the forecasted return for the next day is mean zero (given that it's a random walk?), the sign of the residual is a buy/sell signal. Otherwise, forecast would combine ARIMA mean forecast and GARCH volatility forecast

In [53]:
```
import os
import sys

import pandas as pd
import numpy as np

import quandl
import statsmodels.formula.api as smf
import statsmodels.tsa.api as smt
import statsmodels.api as sm
import scipy.stats as scs
import statsmodels.stats as sms
from arch import arch_model

import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

First, we import the historical price data for our instrument, the SPY ETF in this case. There are a lot of sources for this data, but I've use a Yahoo module and Google source data here.

In [72]:
```
from pandas_datareader import data
import fix_yahoo_finance as yf
yf.pdr_override()

symbol = 'SPY'
data_source='google'
start_date = '2000-01-01'
end_date = '2018-05-01'
df = data.get_data_yahoo(symbol, start_date, end_date)
df.tail()
```

```
[*********************100%***********************]  1 of 1 downloaded
```

Out[72]:

Date	Open	High	Low	Close	Adj Close	Volume
2018-04-25	262.910004	264.130005	260.850006	263.630005	263.630005	103840900
2018-04-26	264.790009	267.250000	264.290009	266.309998	266.309998	67731900
2018-04-27	267.000000	267.339996	265.500000	266.559998	266.559998	57053600
2018-04-30	267.260010	267.890015	264.429993	264.510010	264.510010	82182300
2018-05-01	263.869995	265.100006	262.109985	264.980011	264.980011	74203400

In [73]:
```
# log returns
lrets = np.log(df/df.shift(1)).dropna()
```

In [74]:
```
# define function to handle plotting
def tsplot(y, lags=None, figsize=(15, 10), style='bmh'):
    if not isinstance(y, pd.Series):
        y = pd.Series(y)
    with plt.style.context(style):
        fig = plt.figure(figsize=figsize)
        #mpl.rcParams['font.family'] = 'Ubuntu Mono'
        layout = (3, 2)
        ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
        acf_ax = plt.subplot2grid(layout, (1, 0))
        pacf_ax = plt.subplot2grid(layout, (1, 1))
        qq_ax = plt.subplot2grid(layout, (2, 0))
        pp_ax = plt.subplot2grid(layout, (2, 1))

        y.plot(ax=ts_ax)
        ts_ax.set_title('Time Series Analysis Plots')
        smt.graphics.plot_acf(y, lags=lags, ax=acf_ax, alpha=0.05)
        smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax, alpha=0.05)
        sm.qqplot(y, line='s', ax=qq_ax)
        qq_ax.set_title('QQ Plot')
        scs.probplot(y, sparams=(y.mean(), y.std()), plot=pp_ax)

        plt.tight_layout()
    return
```

We then fit an ARIMA model to the closing price time series by iterating through various p, d and q ranges.

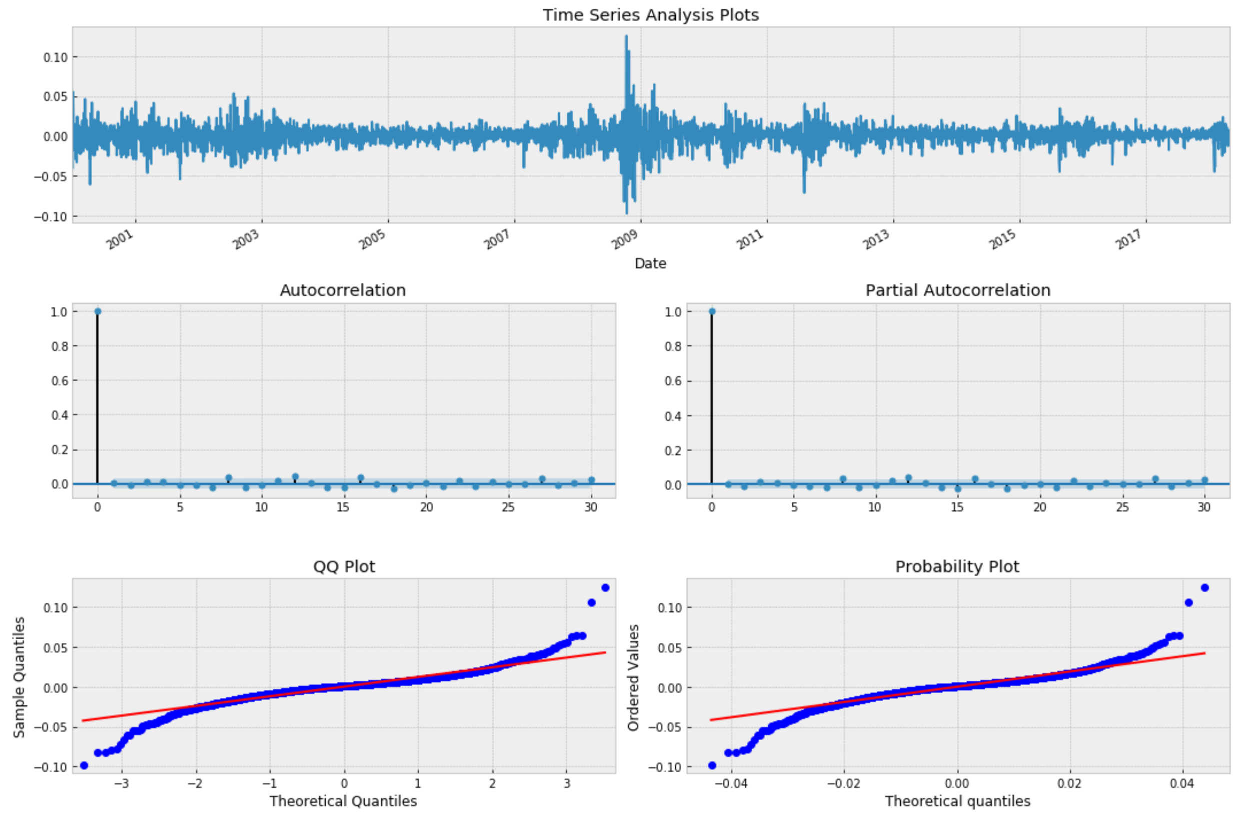
In [75]:
```
# define function to get best arima model
def get_best_model(TS):
    best_aic = np.inf
    best_order = None
    best_mdl = None

    pq_rng = range(5) # [0,1,2,3,4]
    d_rng = range(2) # [0,1]
    for i in pq_rng:
        for d in d_rng:
            for j in pq_rng:
                try:
                    tmp_mdl = smt.ARIMA(TS, order=(i,d,j)).fit(
                        method='mle', trend='nc'
                    )
                    tmp_aic = tmp_mdl.aic
                    if tmp_aic < best_aic:
                        best_aic = tmp_aic
                        best_order = (i, d, j)
                        best_mdl = tmp_mdl
                except: continue
    print('aic: {:6.5f} | order: {}'.format(best_aic, best_order))
    return best_aic, best_order, best_mdl

# Notice I've selected a specific time period to run this analysis
TS = lrets.Close
res_tup = get_best_model(TS)
```

```
aic: -27591.18672 | order: (3, 0, 3)
```

In [76]:
```
order = res_tup[1]
model = res_tup[2]
```

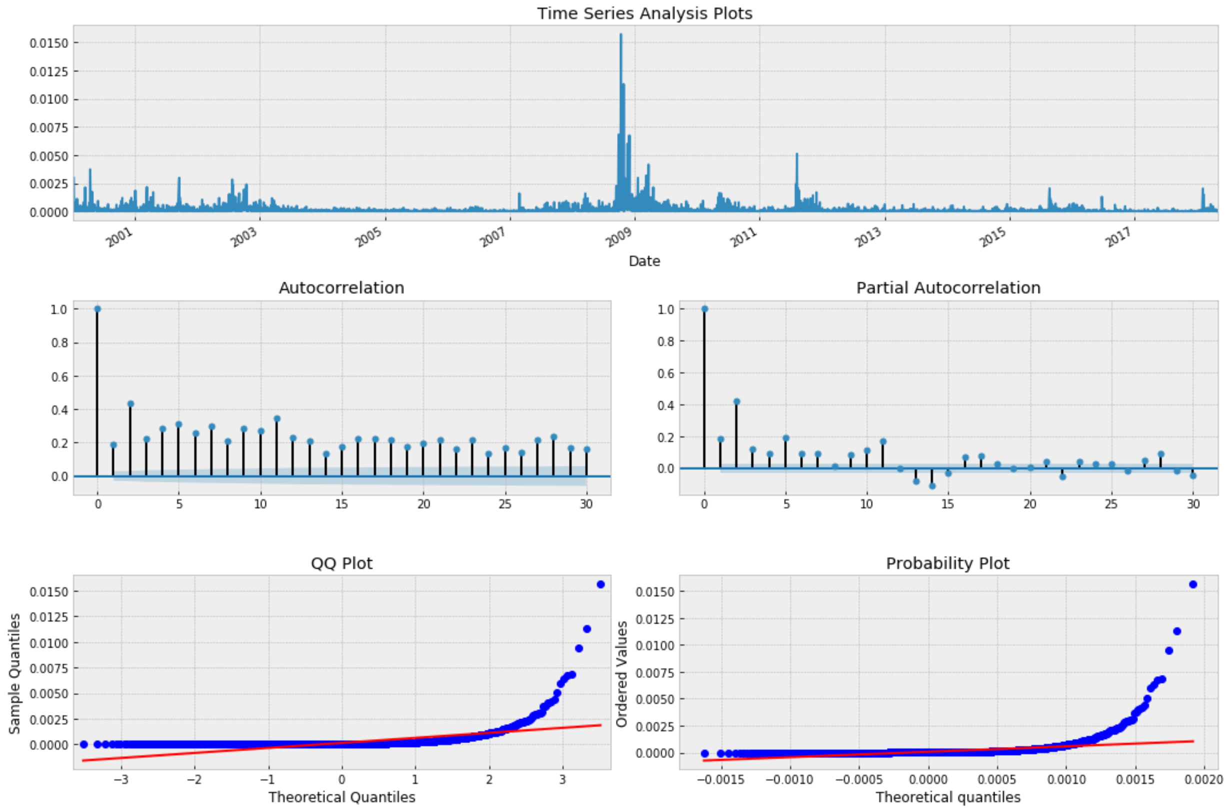
Plotting the residuals of the best fit ARIMA model, we see that the resididuals appeaer to be white noise, which would imply that our model has 'captured' all of the signals within the time series.

In [77]:
```
tsplot(model.resid, lags=30)
```



However, plotting the squared residuals, we see that there are still patterns within the data. From the significant spikes in both the ACF and PACF plots, we see that we can fit an ARMA model to this data, which is the essence of GARCH.

In [78]:
```
tsplot(model.resid**2, lags=30)
```



So we go ahead and fit a GARCH model to the residuals of the ARIMA model using the same p, o and q parameters of the best fit ARIMA model.

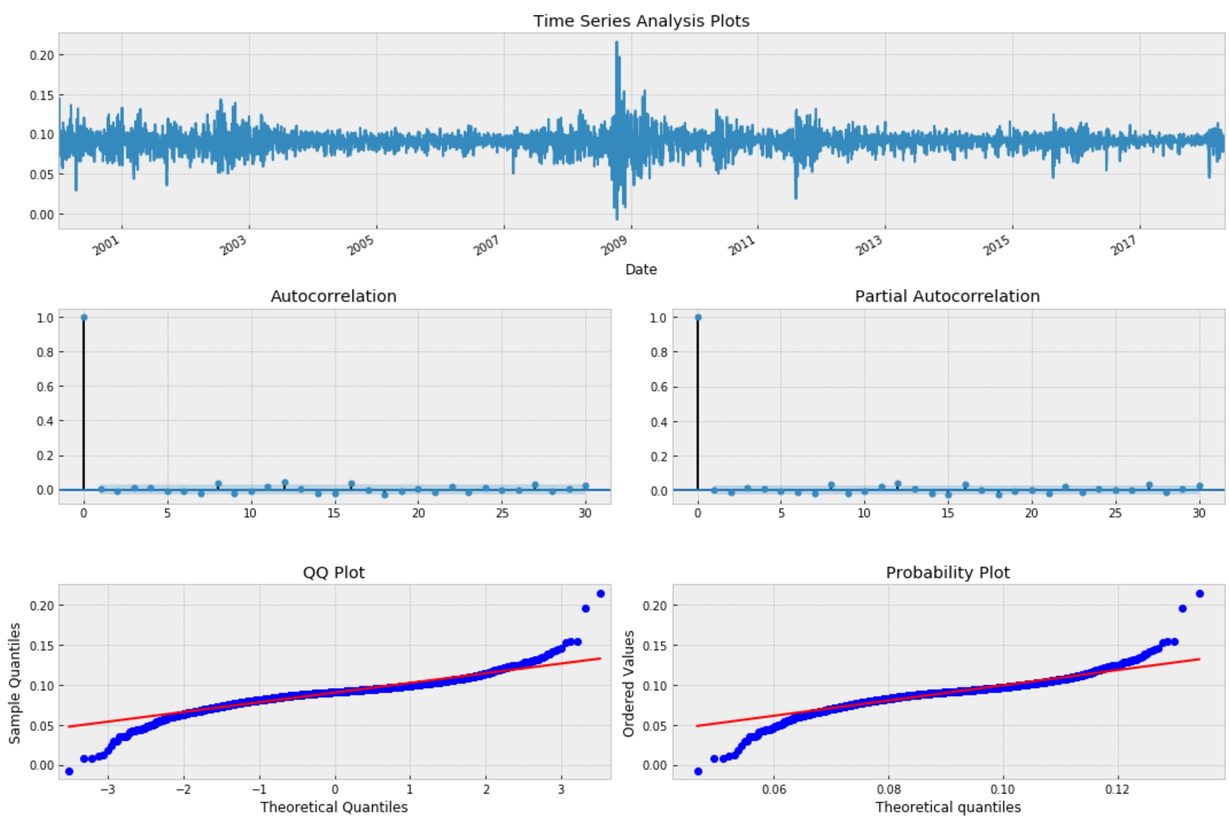
In [85]:
```
# now we can fit the arch model using the best fir arima model parameters

p_ = order[0]
o_ = order[1]
q_ = order[2]

# using a student T distribution usually provides a better fit
# while ensuring p >= 2
am  = arch_model(model.resid, p=max(p_, 2), o=o_, q=q_, dist='StudentsT')
res = am.fit(update_freq=5, disp='off')
print(res.summary())
```

```
                     Constant Mean - GARCH Model Results

=============================================================================
=======
Dep. Variable:                        None   R-squared:
-55.668
Mean Model:                  Constant Mean   Adj. R-squared:
-55.668
Vol Model:                           GARCH   Log-Likelihood:
2357.60
Distribution:      Standardized Student's t   AIC:                       -
4697.21
Method:                 Maximum Likelihood   BIC:                       -
4639.29
                                             No. Observations:
   4610
Date:                     Sun, May 20 2018   Df Residuals:
   4601
Time:                             16:09:26   Df Model:
      9
                               Mean Model
============================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
----------------------------------------------------------------------------
mu            -0.0903  5.545e-03    -16.279  1.402e-59 [ -0.101,-7.939e-02]
                            Volatility Model
=============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
-----------------------------------------------------------------------------
omega      9.4073e-04  4.631e-04      2.031  4.221e-02 [3.308e-05,1.848e-03]
alpha[1]       0.1322      4.279  3.089e-02      0.975     [ -8.254,  8.518]
alpha[2]       0.2556      2.947  8.675e-02      0.931     [ -5.520,  6.032]
alpha[3]   2.1622e-04      1.334  1.621e-04      1.000     [ -2.615,  2.615]
beta[1]        0.0549     24.774  2.215e-03      0.998     [-48.501, 48.611]
beta[2]        0.3405     42.309  8.047e-03      0.994     [-82.583, 83.264]
beta[3]        0.2166     17.462  1.241e-02      0.990     [-34.008, 34.441]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             2.6011  3.246e-02     80.136      0.000 [  2.537,  2.665]
========================================================================

Covariance estimator: robust
```

```
WARNING: The optimizer did not indicate successful convergence. The message w
as
Positive directional derivative for linesearch. See convergence_flag.

C:\Users\bhao1\AppData\Local\Continuum\Anaconda3\lib\site-packages\arch\univari
ate\base.py:524: ConvergenceWarning:
The optimizer returned code 8. The message is:
Positive directional derivative for linesearch
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)
```

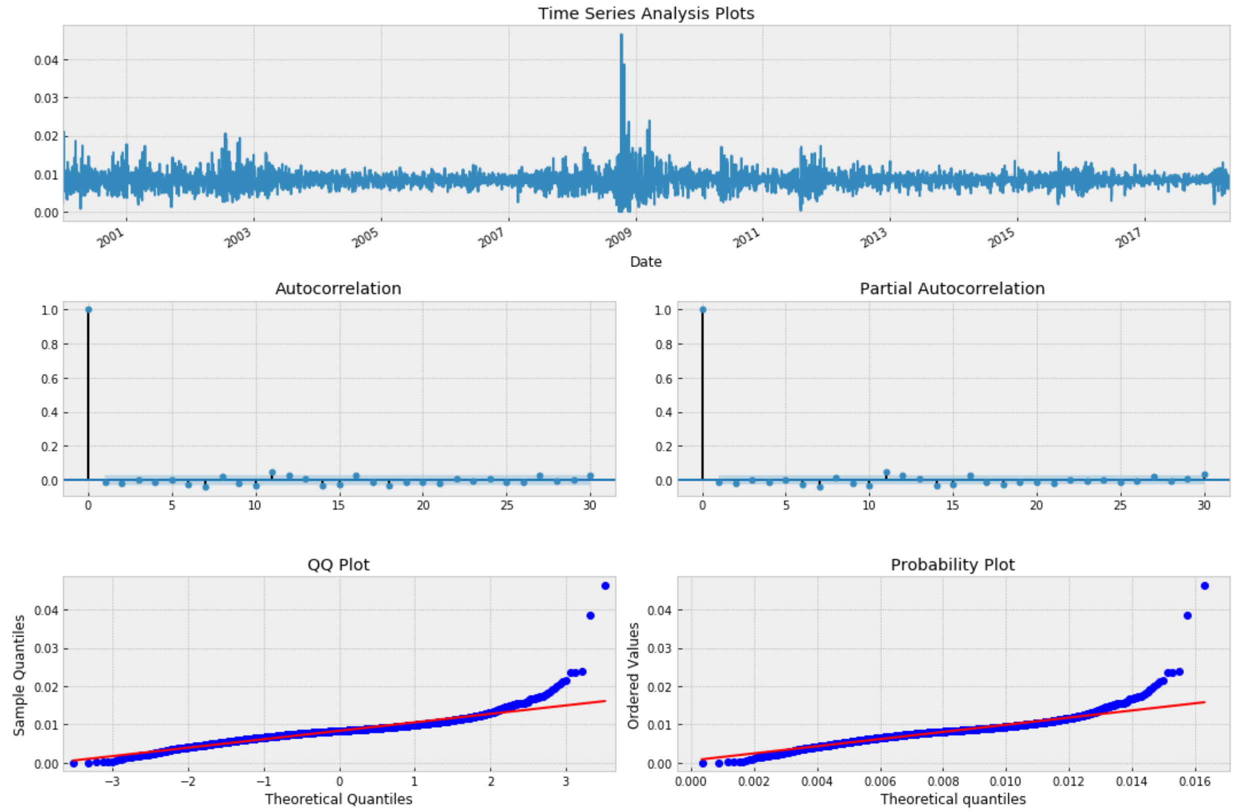
Now plotting the residiuals of the GARCH model, we still see only white noise in the residuals.

In [86]:
```
tsplot(res.resid, lags=30)
```



However, now the squared residuals also appear to be white noise as well, meaning that we've 'explained' the variance with our GARCH model.

In [87]:
```
tsplot(res.resid**2, lags=30)
```



In [83]:
```
out = res.forecast(horizon=1, start=None, align='origin')
print(out.mean.iloc[-1])
print(out.variance.iloc[-1])
```

```
h.1   -0.090257
Name: 2018-05-01 00:00:00, dtype: float64
h.1    0.010412
Name: 2018-05-01 00:00:00, dtype: float64
```

Now we'll back test a simple trading strategy using this approach. We'll use a one-year window to train an ARIMA model, and then we'll fit a GARCH model to the residuals in order to forecast the next day's return. If the forecasted return is positive, we buy, and if it is negative, we short.

In [84]:
```
# simple strategy implementation
# buy when predicted return is positive
# sell when predicted error is posit

windowLength = 252
foreLength = len(lrets) - windowLength
signal = 0*lrets[-foreLength:]
```

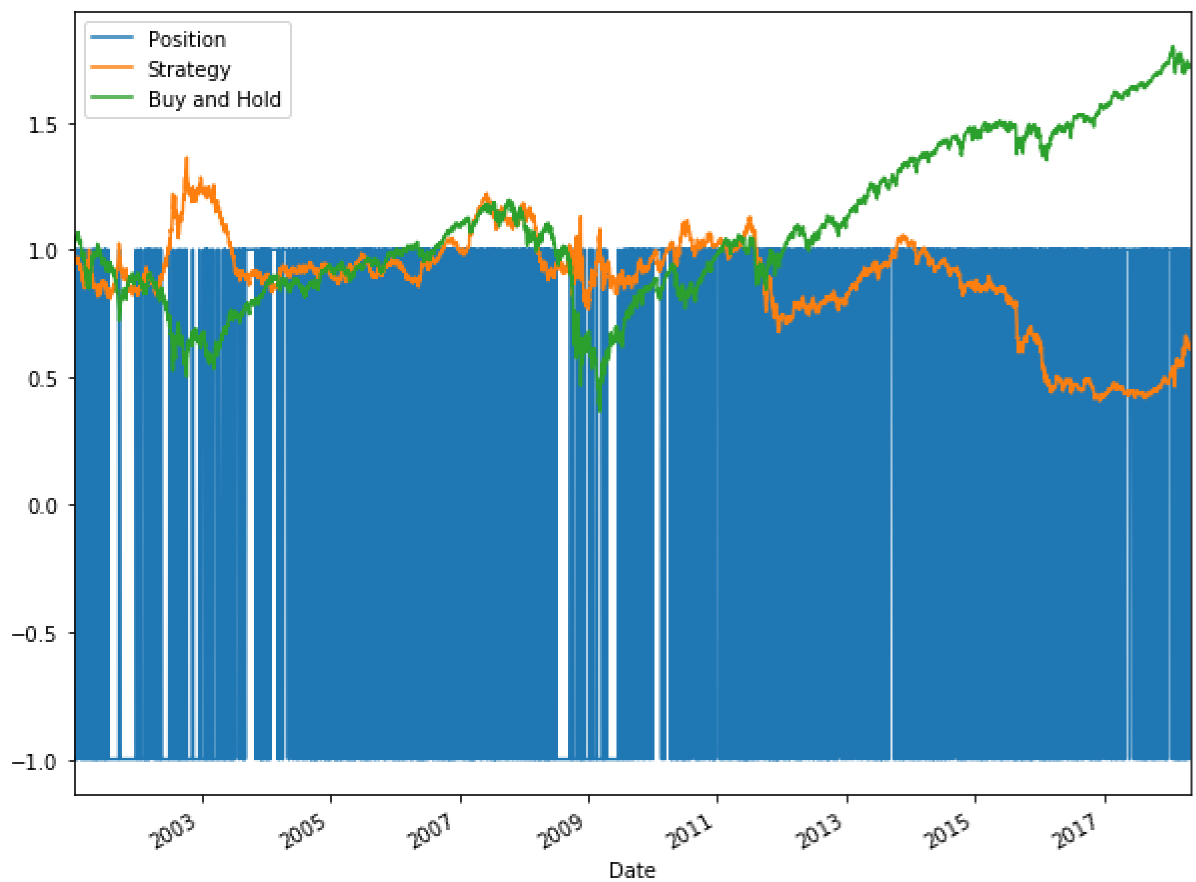
In [88]:
```
for d in range(foreLength):

    # create a rolling window by selecting
    # values between d+1 and d+T of SPY returns

    TS = lrets[(1+d):(windowLength+d)]['Close']

    # find the best fit arima model
    # set d=0 since we've already taken log return of the series
    res_tup = get_best_model(TS)
    order = res_tup[1]
    model = res_tup[2]

    # now with arima model fit, we feed parameters into garch model
    p_ = order[0]
    o_ = order[1]
    q_ = order[2]

    am  = arch_model(model.resid, p=max(p_, 2), o=o_, q=q_, dist='StudentsT')
    res = am.fit(update_freq=5, disp='off')

    # generate a forecast of next day return using our fitted model
    out = res.forecast(horizon=1, start=None, align='origin')

    # set trading signal equal to the sign of forecasted return
    # buy if we expect positive returns, sell if negative

    signal.iloc[d] = np.sign(out.mean['h.1'].iloc[-1])
```

```
aic: -1400.04861 | order: (1, 0, 1)

C:\Users\bhao1\AppData\Local\Continuum\Anaconda3\lib\site-packages\arch\univa
riate\base.py:524: ConvergenceWarning:
The optimizer returned code 8. The message is:
Positive directional derivative for linesearch
See scipy.optimize.fmin_slsqp for code meaning.

  ConvergenceWarning)

aic: -1394.61716 | order: (2, 0, 2)
aic: -1394.93917 | order: (2, 0, 2)
aic: -1405.64890 | order: (4, 0, 2)
aic: -1405.17300 | order: (4, 0, 2)
aic: -1405.18794 | order: (4, 0, 2)

C:\Users\bhao1\AppData\Local\Continuum\Anaconda3\lib\site-packages\arch\univa
riate\base.py:524: ConvergenceWarning:
The optimizer returned code 8. The message is:
Positive directional derivative for linesearch
```

We've run the backtest from 2001 thru the present. To contextualize the results, we compare the trading strategy's performance against a simple buy-and-hold strategy. As you can see, the strategy performed OK up until the most recent bull market run that started in 2009. Since then, the buy-and-hold strategy has significantly outperformed.

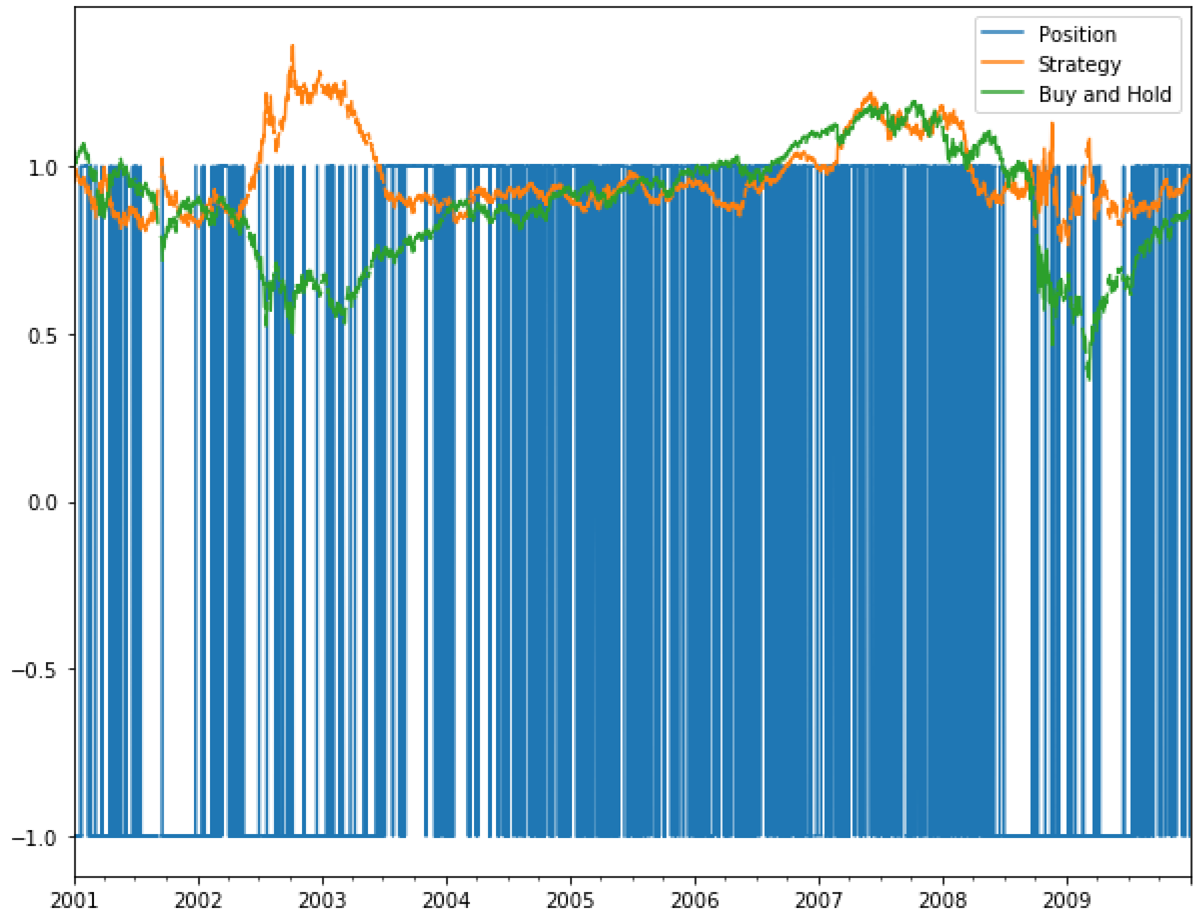
The blue represents the position, i.e. -1 is short one unit and +1 is long one unit. It's hard to see given the time granularity, but the solid blue basically indicates that we are constantly flipping back and forth between being long and short.

In [95]:
```
returns = pd.DataFrame(index   = signal.index,
                       columns = ['Buy and Hold', 'Strategy'])
returns['Buy and Hold'] = lrets[-foreLength:]['Close']
returns['Position'    ] = signal['Close']
returns['Strategy'    ] = signal['Close'] * returns['Buy and Hold']

eqCurves = pd.DataFrame(index   = signal.index,
                        columns = ['Buy and Hold', 'Strategy'])
eqCurves['Buy and Hold'] = returns['Buy and Hold'].cumsum() + 1
eqCurves['Position'    ] = returns['Position'    ]
eqCurves['Strategy'    ] = returns['Strategy'    ].cumsum() + 1

eqCurves['Position'    ].plot(figsize=(10,8))
eqCurves['Strategy'    ].plot()
eqCurves['Buy and Hold'].plot()
plt.legend()
plt.show()
```



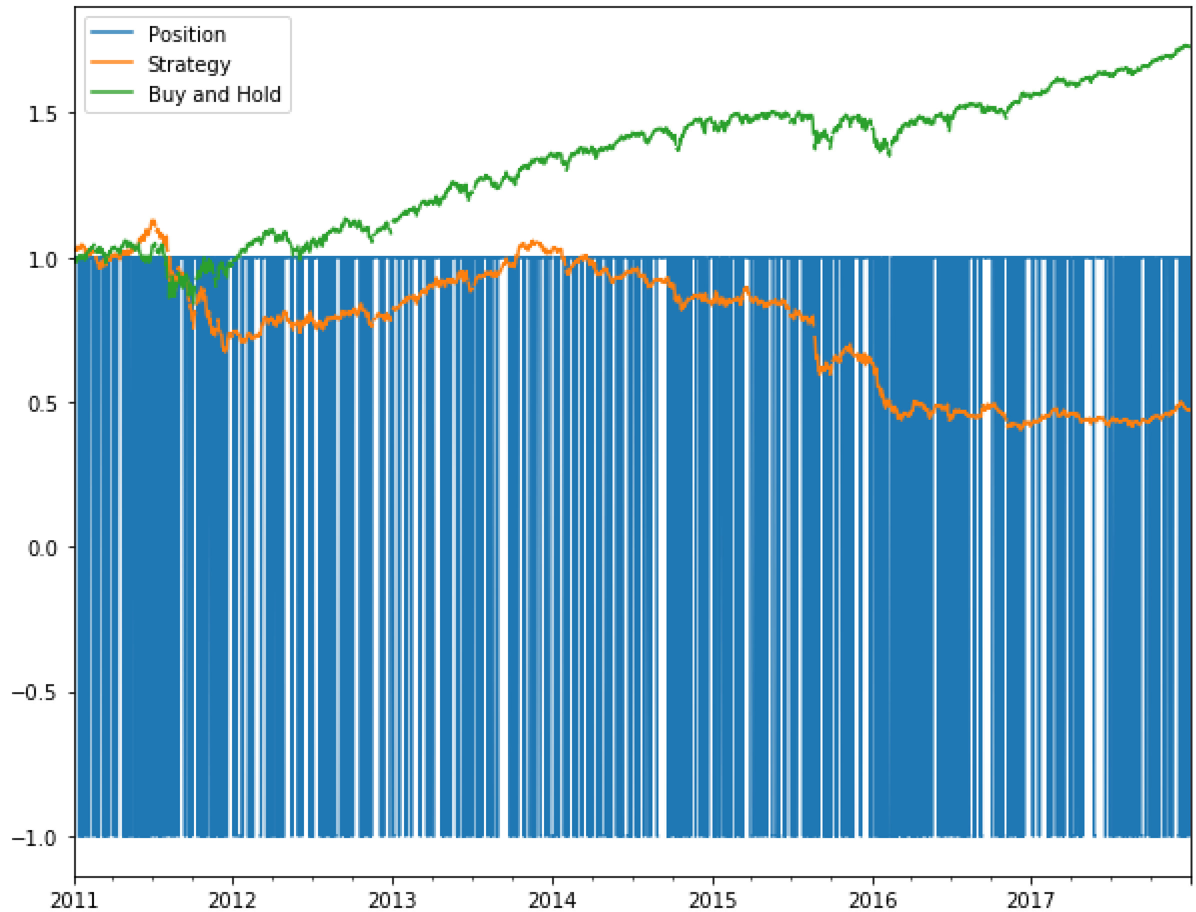
If we look at the first decade of the backtest period, we see that the trading strategy did quite well in the early 2000s after the dot-com bubble burst as well as during the Great Recession in 2008-2009.

In [121]:
```
rng = pd.DatetimeIndex(start='20010101', freq='D', periods=9*365)
eqCurves.loc[rng]['Position'    ].plot(figsize=(10,8))
eqCurves.loc[rng]['Strategy'    ].plot()
eqCurves.loc[rng]['Buy and Hold'].plot()
plt.legend()
plt.show()
```



Again, looking at the recent bull market period, the trading strategy significantly underperforms the buy-and-hold strategy.

In [120]:
```
rng = pd.DatetimeIndex(start='20110101', freq='D', periods=7*365)
eqCurves.loc[rng]['Position'    ].plot(figsize=(10,8))
eqCurves.loc[rng]['Strategy'    ].plot()
eqCurves.loc[rng]['Buy and Hold'].plot()
plt.legend()
plt.show()
```




```
This was a very simple strategy and obviously not a very profitable one.
Furthermore, this backtest does not account for slippage or trading costs, which
would make this trading strategy completely untenable even if the above backtest
was quite profitable.

To productionalize trading strategies in reality, we could do a handful of
things:
- Test different instruments
- Test different training windows
- Apply thresholds to moderate the constant buying and selling (to reduce
trading costs)
- Combine this strategy with other strategies in a portfolio approach

All that said, hopefully this demo illustrates some practical uses of GARCH
modeling.
```