Team Auquan Follow
Dec 12 · 12 min read

Time Series Analysis for Financial Data VI— GARCH model and predicting SPX returns

Download the iPython notebook here

In this mini series on Time Series modelling for Financial Data, so far we've used AR, MA and a combination of these models on asset prices to try and model how our asset behaves. We've found that we were able to model certain time periods well with these models and failed at other times.

This was because of volatility clustering or heteroskedasticity. In this post, we will discuss conditional heteroskedasticity, leading us to our first conditional heteroskedastic model, known as ***ARCH***. Then we will discuss extensions to ARCH, leading us to the famous Generalised Autoregressive Conditional Heteroskedasticity model of order p,q, also known as ***GARCH(p,q)***. GARCH is used extensively within the financial industry as many asset prices are conditional heteroskedastic.

Let's do a quick recap first:

We have considered the following models so far in this series (it is recommended reading the series in order if you have not done so already):

- Discrete White Noise and Random Walks
- Auto Regresssive Models AR(p)
- Moving Average Models MA(q)
- Auto Regresssive Moving Average Models ARMA(p,q) and
- Auto Regresssive Integrated Moving Average Models ARIMA(p,d,q)

Now we are at the final piece of the puzzle. We need a model to examine conditional heteroskedasticity in financial series that exhibit volatility clustering.

What is conditional heteroskedasticity?

Conditional heteroskedasticity exists in finance because asset returns are volatile.

A collection of random variables is **heteroskedastic** if there are subsets of variables within the larger set that have a different variance from the remaining variables.

Consider a day when equities markets undergo a substantial drop. The market gets into panic mode, automated risk management systems start getting of their long positions by selling their positions and all of this leads to a further fall in prices. An increase in variance from the initial price drop

leads to to significant further downward volatility.

That is, an increase in variance is serially correlated to a further increase in variance in such a “sell-off” period. Or looking at it the other way around, a period of of increased variance is conditional on an initial sell-off . Thus we say that such series are **conditional heteroskedastic.**

Conditionally heteroskedastic(CH) series are non stationary since its variance is not constant in time. One of the challenging aspects of conditional heteroskedastic series is ACF plots of a series with volatility might still appear to be a realisation of stationary discrete white noise.

How can we incorporate CH in our model? One way could be to create an AR model for the variance itself—a model that actually accounts for the changes in the variance over time using past values of the variance.

This is the basis of the Autoregressive Conditional Heteroskedastic (ARCH) model.

Autoregressive Conditionally Heteroskedastic Models—ARCH(p)

ARCH(p) model is simply an AR(p) model applied to the variance of a time series.

ARCH(1) is given by:

$$Var(x(t)) = \sigma^2(t) = \alpha * \sigma^2(t-1) + \alpha 1$$

The actual time series is given by:

$$x(t) = w(t) * \sigma(t) = w(t) * \sqrt{(\alpha * \sigma^2(t-1) + \alpha 1)}$$

where w(t) is white noise

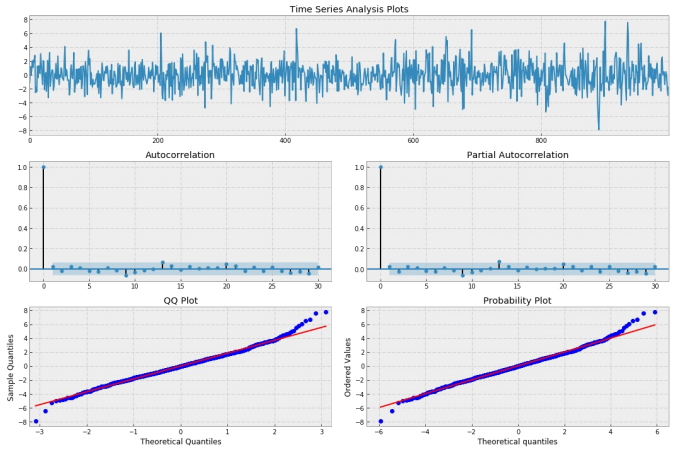
When To Apply ARCH(p)?

Let’s say we fit an AR(p) model and the residuals look almost like white noise but we are concerned about decay of the *p* lag on a ACF plot of the series. If we find that we can apply an AR(p) to the square of residuals as well, then we have an indication that an ARCH(p) process may be appropriate.

Note that ARCH(p) should only ever be applied to a series that has already had an appropriate model fitted sufficient to leave the residuals looking like discrete white noise. Since we can only tell whether ARCH is appropriate or not by squaring the residuals and examining the ACF, we also need to ensure that the mean of the residuals is zero.

ARCH should only ever be applied to series that do not have any trends or seasonal effects, i.e. that has no (evident) serially correlation. ARIMA is often applied to such a series, at which point ARCH may be a good fit.

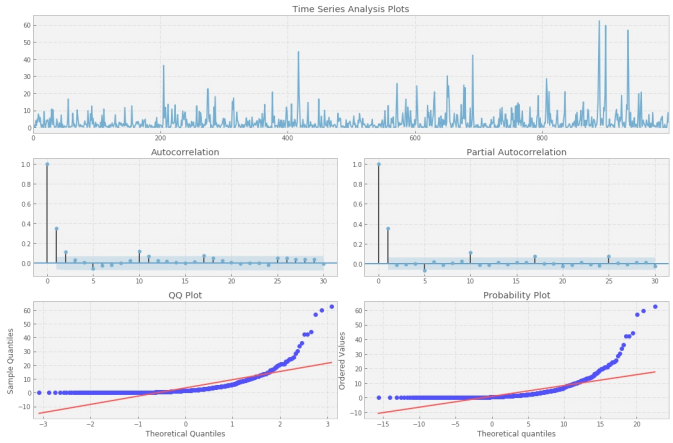
```
# Simulate ARCH(1) series
# Var(yt) = a_0 + a_1*y{t-1}**2
# if a_1 is between 0 and 1 then yt is white noise

np.random.seed(13)

a0 = 2
a1 = .5

y = w = np.random.normal(size=1000)
Y = np.empty_like(y)

for t in range(len(y)):
    y[t] = w[t] * np.sqrt((a0 + a1*y[t-1]**2))

# simulated ARCH(1) series, looks like white noise
tsplot(y, lags=30)
```



ARCH(1) series

Notice the time series looks just like white noise. However, let's see what happens when we plot the square of the series.

```
tsplot(y**2, lags=30)
```



Square of ARCH(1) series

Now the ACF, and PACF seem to show significance at lag 1 indicating an AR(1) model for the variance may be appropriate.

An obvious question to ask at this stage is if we are going to apply an AR(p) process to the variance, why not a Moving Average MA(q) model as well? Or a mixed model such as

ARMA(p,q)?

This is actually the motivation for the Generalised ARCH model, known as GARCH.

Generalized Autoregressive Conditionally Heteroskedastic Models—GARCH(p,q)

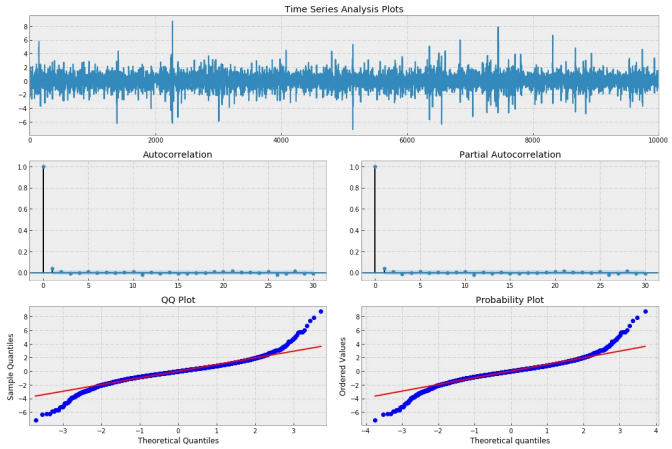
Just like *ARCH(p)* is *AR(p)* applied to the variance of a time series, *GARCH(p, q)* is an *ARMA(p,q)* model applied to the variance of a time series. The AR(p) models the variance of the residuals (squared errors) or simply our time series squared. The MA(q) portion models the variance of the process.

The GARCH(1,1) model is:

$$\sigma^2(t) = a*\sigma^2(t-1) + b*e^2(t-1) + w$$

(a+b) must be less than 1 or the model is unstable. We can simulate a GARCH(1, 1) process below.

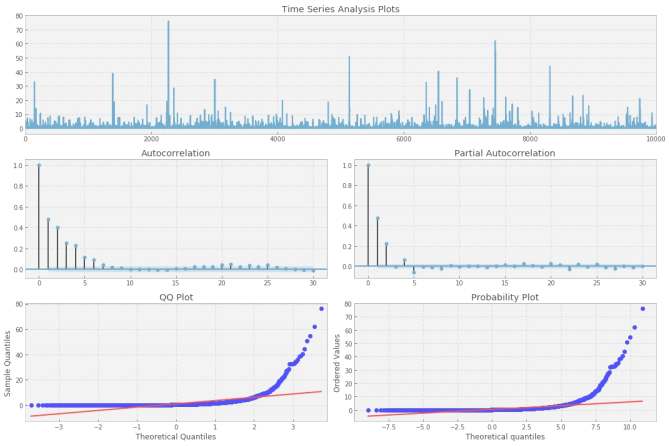
```
# Simulating a GARCH(1, 1) process

np.random.seed(2)

a0 = 0.2
a1 = 0.5
b1 = 0.3

n = 10000
w = np.random.normal(size=n)
eps = np.zeros_like(w)
sigsq = np.zeros_like(w)

for i in range(1, n):
    sigsq[i] = a0 + a1*(eps[i-1]**2) + b1*sigsq[i-1]
    eps[i] = w[i] * np.sqrt(sigsq[i])

_ = tsplot(eps, lags=30)
```



GARCH(1,1) process

Again, notice that overall this process closely resembles white noise, however take a look when we view the squared eps series.

```
_ = tsplot(eps**2, lags=30)
```



Square of GARCH(1,1) process

There is substantial evidence of a conditionally heteroskedastic process via the decay of successive lags. The significance of the lags in both the ACF and PACF indicate we need both AR and MA components for our model. Let's see if we can recover our process parameters using a GARCH(1, 1) model. Here we make use of the arch_model function from the ARCH package.

```
# Fit a GARCH(1, 1) model to our simulated EPS series
# We use the arch_model function from the ARCH package

am = arch_model(eps)
res = am.fit(update_freq=5)
print(res.summary())
```

```
Iteration:     5,   Func. Count:     38,   Neg. LLF: 12311.7950557
Iteration:    10,   Func. Count:     71,   Neg. LLF: 12238.5926559
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 12237.3032673
            Iterations: 13
            Function evaluations: 89
            Gradient evaluations: 13
                  Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                    y   R-squared:                      -0.000
Mean Model:             Constant Mean   Adj. R-squared:                 -0.000
Vol Model:                      GARCH   Log-Likelihood:               -12237.3
Distribution:                  Normal   AIC:                           24482.6
Method:      Maximum Likelihood   BIC:                           24511.4
                                        No. Observations:                10000
Date:          Tue, Feb 28 2017   Df Residuals:                        9996
Time:                  20:52:48   Df Model:                               4
                           Mean Model
==============================================================================
                 coef    std err      t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
mu       -6.7225e-03  6.735e-03 -0.998  0.318 [-1.992e-02,6.478e-03]
                              Volatility Model
==============================================================================
                 coef    std err       t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------------
omega       0.2021  1.043e-02   19.383  1.084e-83 [  0.182,  0.223]
alpha[1]    0.5162  2.016e-02   25.611 1.144e-144 [  0.477,  0.556]
beta[1]     0.2879  1.870e-02   15.395  1.781e-53 [  0.251,  0.325]
```

```
==========================================
=========================
```

```
Covariance estimator: robust
```

We can see that the true parameters all fall within the respective confidence intervals.

Application to Financial Time Series

Now apply the procedure to a financial time series. Here we're going to use SPX returns. The process is as follows:

- Iterate through combinations of ARIMA(p, d, q) models to best fit our time series.
- Pick the GARCH model orders according to the ARIMA model with lowest AIC.
- Fit the GARCH(p, q) model to our time series.
- Examine the model residuals and squared residuals for autocorrelation

Here, we first try to fit SPX return to an ARIMA process and find the best order.

```
import auquanToolbox.dataloader as dl
```

```
end = '2017-01-01'
start = '2010-01-01'
symbols = ['SPX']
data = dl.load_data_nologs('nasdaq', symbols , start, end)['ADJ CLOSE']
# log returns
lrets = np.log(data/data.shift(1)).dropna()
```

```
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: {:6.2f} | order: {}'.format(best_aic, best_order))
    return best_aic, best_order, best_mdl
```

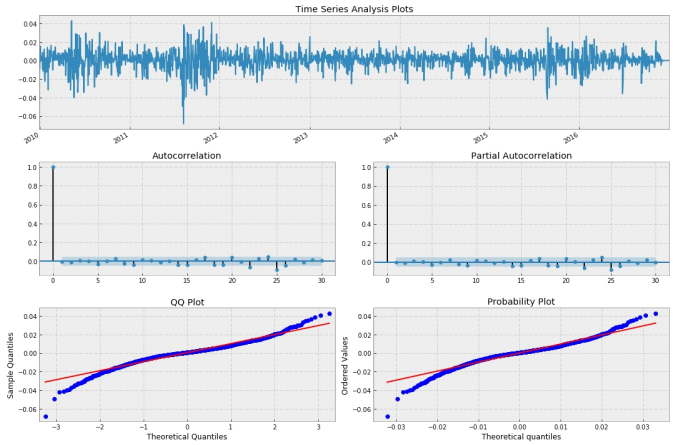
```
TS = lrets.SPX
res_tup = _get_best_model(TS)
```

aic: -11323.07 | order: (3, 0, 3)

```
order = res_tup[1]
model = res_tup[2]
```

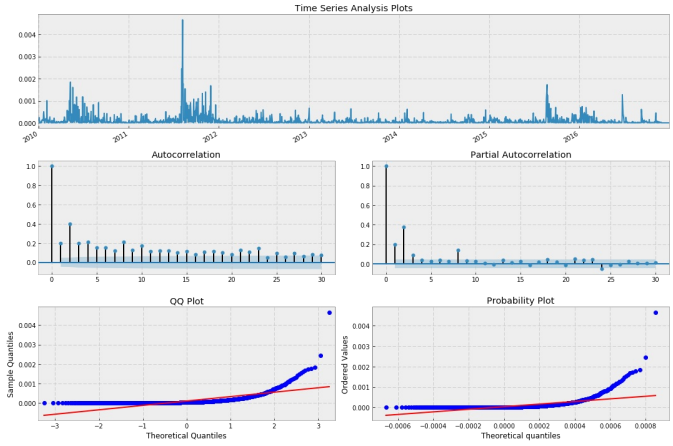
Since we've already taken the log of returns, we should expect our integrated component d to equal zero, which it does. We find the best model is ARIMA(3,0,3). Now we plot the residuals to decide if they possess evidence of conditional heteroskedastic behaviour

```
tsplot(model.resid, lags=30)
```



We find the residuals look like white noise. Let’s look at the square of residuals

```
tsplot(model.resid**2, lags=30)
```



We can see clear evidence of autocorrelation in squared residuals. Let’s fit a GARCH model and see how it does.

```
# Now we can fit the arch model using the best fit arima model parameters
```

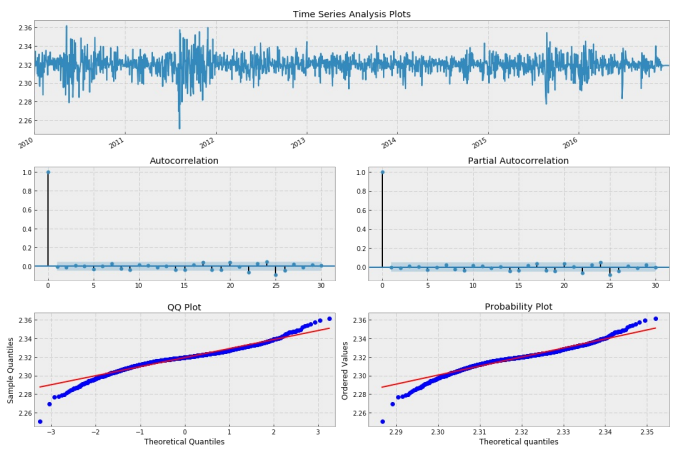
```
p_ = order[0]
o_ = order[1]
q_ = order[2]
```

```
am = arch_model(model.resid, p=p_, 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:      -56917.881
Mean Model:             Constant Mean   Adj. R-squared:  -56917.881
Vol Model:                      GARCH   Log-Likelihood:    -4173.44
Distribution: Standardized Student's t   AIC:               8364.88
Method:           Maximum Likelihood   BIC:               8414.15
                                       No. Observations:     1764
Date:               Tue, Feb 28 2017   Df Residuals:         1755
Time:                       20:53:30   Df Model:                9
                          Mean Model
==============================================================================
             coef    std err       t     P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------------
mu        -2.3189  9.829e-03 -235.934      0.000 [ -2.338, -2.300]
                        Volatility Model
==============================================================================
             coef  std err      t     P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
omega    1.2926e-04 2.212e-04 0.584     0.559 [-3.043e-04,5.628e-04]
alpha[1]    0.0170 1.547e-02 1.099     0.272 [-1.332e-02,4.733e-02]
alpha[2]    0.4638 0.207     2.241 2.500e-02    [5.824e-02,  0.869]
alpha[3]    0.5190 0.213     2.437 1.482e-02     [  0.102,  0.937]
beta[1]  7.9655e-05 0.333 2.394e-04     1.000     [ -0.652,  0.652]
beta[2]  3.8056e-05 0.545 6.980e-05     1.000     [ -1.069,  1.069]
beta[3]  1.6184e-03 0.312 5.194e-03     0.996     [ -0.609,  0.612]
                          Distribution
==============================================================================
             coef    std err       t     P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------------
nu         7.7912     0.362   21.531 8.018e-103 [  7.082,  8.500]
==============================================================================

Covariance estimator: robust
```

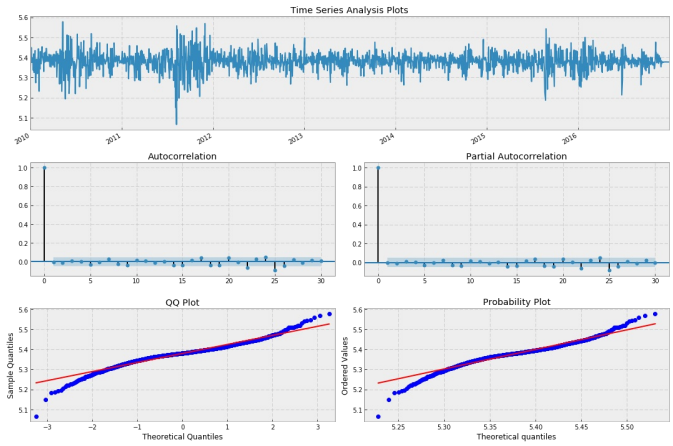
Let’s plot the residuals again

```
tsplot(res.resid, lags=30)
```



The plots looks like a realisation of a discrete white noise process, indicating a good fit. Let's plot a square of residuals to be sure

```
tsplot(res.resid**2, lags=30)
```



We have what looks like a realisation of a discrete white noise process, indicating that we have "explained" the serial correlation present in the squared residuals with an appropriate mixture of ARIMA(p,d,q) and GARCH(p,q).

Next Steps—Sample Trading Strategy

We are now at the point in our time series analysis where we have studied ARIMA and GARCH, allowing us to fit a combination of these models to a stock market index, and to determine if we have achieved a good fit or not.

The next step is to actually produce forecasts of future daily returns values from this combination and use it to create a basic trading strategy for the S&P500.

```
import auquanToolbox.dataloader as dl
```

```
end = '2016-11-30'
start = '2000-01-01'
symbols = ['SPX']
data = dl.load_data_nologs('nasdaq', symbols ,
                    start, end)['ADJ CLOSE']
# log returns
lrets = np.log(data/data.shift(1)).dropna()
```

Strategy Overview

Let's try to create a simple strategy using our knowledge so far about ARIMA and GARCH models. The idea of this strategy is as below:

- Fit an ARIMA and GARCH model everyday on log of S&P 500 returns for previous T days
- Use the combined model to make a prediction for the

next day's return

- If the prediction is positive, buy the stock and if negative, short the stock at today's close
- If the prediction is the same as the previous day then do nothing

Strategy Implementation

Let's start by choosing an appropriate window ***T o***f previous days we are going to use to make a prediction. We are going to use ***T = 252 (1 year)***, but this parameter should be optimised in order to improve performance or reduce drawdown.

```
windowLength = 252
```

We will now attempt to generate a trading signal for *length(data)- T* days

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

To backtest our strategy, let's loop through every day in the trading data and fit an appropriate ARIMA and GARCH model to the rolling window of length 252. We've defined the functions to fit ARIMA and GARCH above (Given that we try 32 separate ARIMA fits and fit a GARCH model, for each day, the indicator can take a long time to generate)

```
for d in range(foreLength):

    # create a rolling window by selecting
    # values between d+1 and d+T of S&P500 returns

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

    # Find the best ARIMA fit
    # 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 that we have our ARIMA fit, we feed this to GARCH model
    p_ = order[0]
    o_ = order[1]
    q_ = order[2]

    am = arch_model(model.resid, p=p_, 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])
```

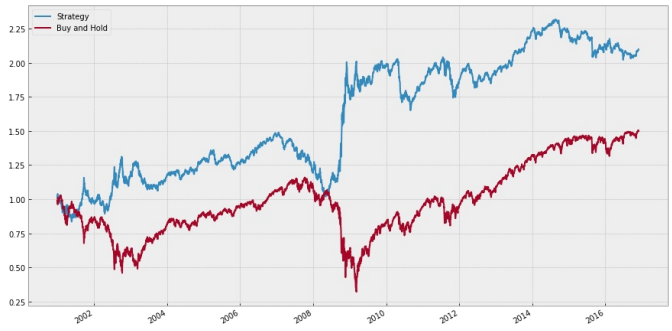
Note: *The backtest is doesn't take commission or slippage into account, hence the performance achieved in a real trading system would be lower than what you see here.*

Strategy Results

Now that we have generated our signals, we need to compare its performance to '*Buy and Hold*': what would our returns be if we simply bought the S&P 500 at the start of our backtest period.

```
returns = pd.DataFrame(index = signal.index,
                columns=['Buy and Hold', 'Strategy'])
returns['Buy and Hold'] = lrets[-foreLength:]
returns['Strategy'] = signal['SPX']*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['Strategy'] = returns['Strategy'].cumsum()+1


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



Long/Short SPX strategy based GARCH + ARIMA model from 2000–2016

We find the model does outperform a naive Buy and Hold strategy. However, the model doesn't perform well all the time, you can see majority of the gains have happened during short durations in 2000–2001 and 2008. It seems there are certain market conditions when the model does exceedingly well.

Long/Short SPX strategy based GARCH + ARIMA model from 2000–2003

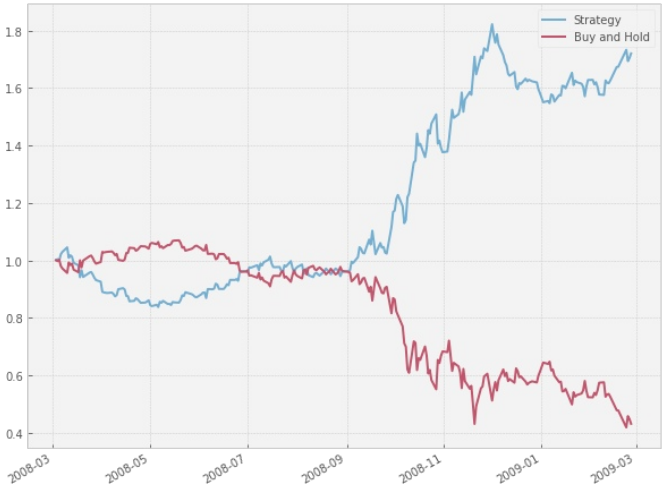
In periods of high volatility, or when S&P 500 had periods of '*sell-off*' *, such as 2000–2002 or the crash of 2008–09,* the

strategy does extremely well, possibly because our GARCH model captures the conditional volatility well. During periods of uptrend in S&P500, such as the bull run from 2002–2007 the model performs on par with S&P 500.

Long/Short SPX strategy based GARCH + ARIMA model from 2003–2007 bull period

In the current bull run from 2009, the model has performed poorly compared to S&P 500. The index behaved like what looks to be more a stochastic trend, the model performance suffered in this duration.

There are some caveats here: We don't account for slippages or trading costs here, which would significantly eat into profits. Also, we've performed a backtest on a stock market index and not a tradeable instrument. Ideally, we should perform the same modelling and backtest on S&P500 futures or a Exchange Traded Fund (ETF) like SPY .



Long/Short SPX strategy based GARCH + ARIMA model during crash of 2008–09

This strategy can be easily applied to other stock market indices, other regions, equities or other asset classes.

Long/Short SPX strategy based GARCH + ARIMA model from 2009-present

You should try researching other instruments, playing with window parameters and see if you can make improvements on the results presented here. Other improvements to the strategy could include buying/selling only if predicted returns are more or less than a certain threshold, incorporating variance of prediction into the strategy etc.

If you do find interesting strategies, participate in our competition, QuantQuest and earn profit shares on your strategies!