Forecasting Volatility ETF Ticker: QABA

Gerson Jimenez

4/18/2023

Preparation

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf
import matplotlib.pyplot as plt
import seaborn as sns
import arch
from arch import arch_model
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
import yfinance as yf
from arch.__future__ import reindexing
```

Data Prep

```
data = yf.download('QABA')
```

```
## [**********************100%**********************]  1 of 1 completed
```

```
returns= data['Adj Close'].ffill()
# Returns are daily
log_returns = 100 * np.log(returns / returns.shift(1))
log_returns.tail()
```

```
## Date
## 2023-04-12   -1.247319
## 2023-04-13    1.151919
## 2023-04-14   -2.000303
## 2023-04-17    2.143368
## 2023-04-18   -2.680350
## Name: Adj Close, dtype: float64
```

```
log_returns = log_returns.dropna()
```

1

Statiscal Properties of the Daily ETF Return

```
summary_stats = log_returns.describe()
# Summary stat for log returns
print(summary_stats)
```

```
## count    3472.000000
## mean        0.027347
## std         1.618529
## min       -13.865144
## 25%        -0.759939
## 50%         0.038924
## 75%         0.851144
## max        12.113358
## Name: Adj Close, dtype: float64
```

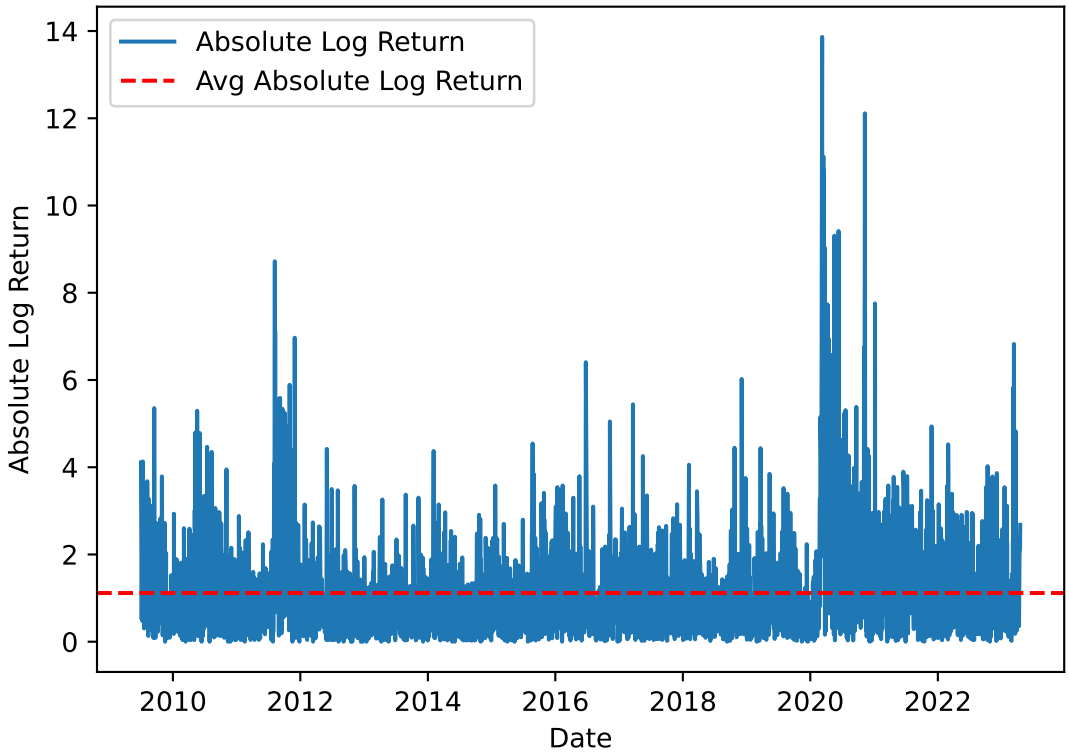
Calculating the summary statistics we are given count which is the number of observations / the amount of dates in which the ETF was on the market. We see the mean log return of the ETF at 0.027347 which means that this is the average daily return through out entirety of the ETF since April 18 2023. We are also given the smallest daily log return and largest daily log return, represented by min and max. smallest return came at -13.865164 and the largest at 12.113327. This and with a standard deviation of 1.618529 suggest the returns are highly volatile. Looking at the quartile summary we see that most of the returns are relatively small. the bottom 25% are -0.759932, 75% at 0.851175.

2

```
# Creating absolute log returns
abs_returns = abs(log_returns)

fig, ax = plt.subplots()
ax.plot(abs_returns.index, abs_returns, label='Absolute Log Return')
# Adding a horizontal line representing the average absolute log return
ax.axhline(y=abs_returns.mean(), color='r', linestyle='--', label='Avg Absolute Log Return')
ax.legend()
ax.set_xlabel('Date')
ax.set_ylabel('Absolute Log Return')

plt.show()
```



After plotting the absolute log return we see evidence of volatility clustering which are returns alternating between periods of high and low volatility, we see during the second half of 2011 the volatility were immensely higher than periods before and after. We see the same pattern take place again in 2020 and early 2021, which is highly due to Covid's effect on the market. There are also many periods in which the absolute returns are much higher to the average returns seen by the red dotted line fixed at a 1.116

3

```
# Plotting ACF (Autocorrelation Function) for log returns
fig, ax = plt.subplots()
plot_acf(log_returns)
plt.title('ACF of Log Returns')
plt.ylim(-0.20, 1)
```

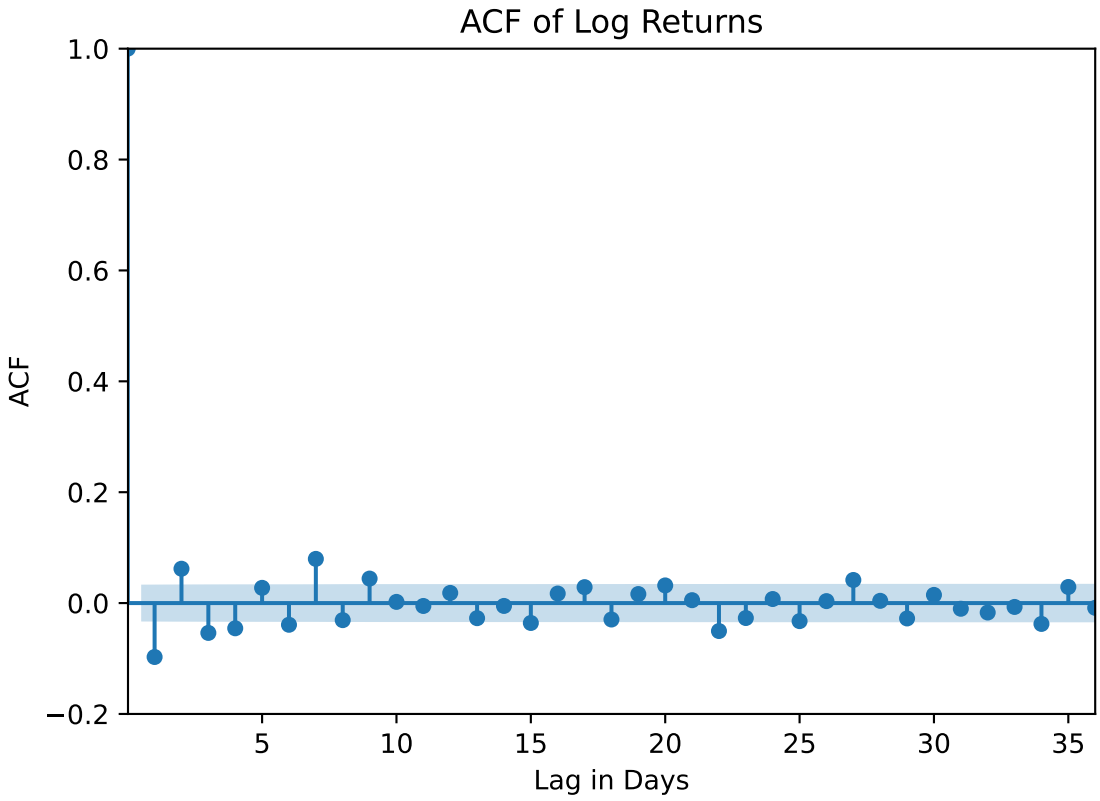
```
## (-0.2, 1.0)
```

```
plt.xlim(0.01, 36)
```

```
## (0.01, 36.0)
```

```
plt.xlabel('Lag in Days')
plt.ylabel('ACF')

plt.show()
```




```
fig, ax = plt.subplots()
plot_acf(abs_returns)
plt.title('ACF of Absolute Log Returns')
plt.ylim(-0.20, 1)
```

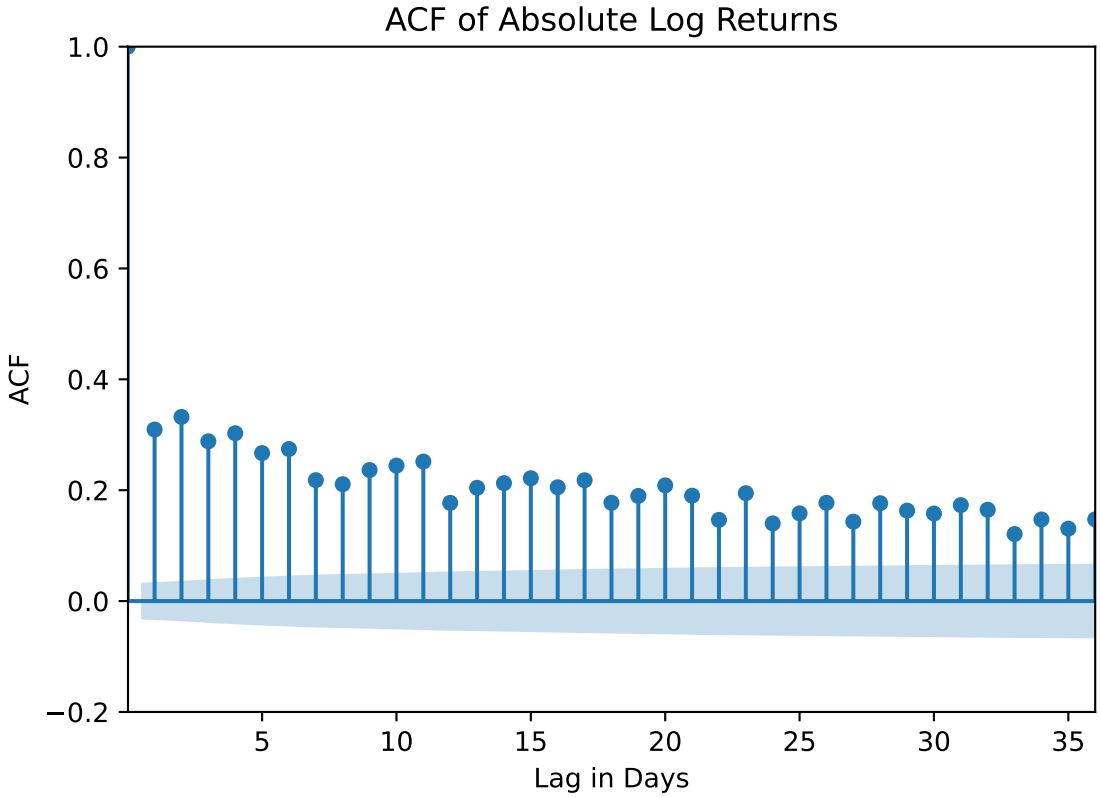
```
## (-0.2, 1.0)
```

```
plt.xlim(0.01,36)
```

```
## (0.01, 36.0)
```

```
plt.xlabel('Lag in Days')
plt.ylabel('ACF')

plt.show()
```



The blue bars are a measurement of statistical significance and correlation to help predict the movement of volatility. Any point within the bars are not statistically significant. In the ACF of log returns we are able to say that lag is a significant measurement to predict volatility. This is not true for every lag value, but it is very significant within the first 5 days and getting smaller and smaller as the numbers of days lags increases. We see the same pattern in the absolute log returns ACF, where the longer the lag value the less correlation occurs but every lag value is significant to measure movement unlike the log returns.

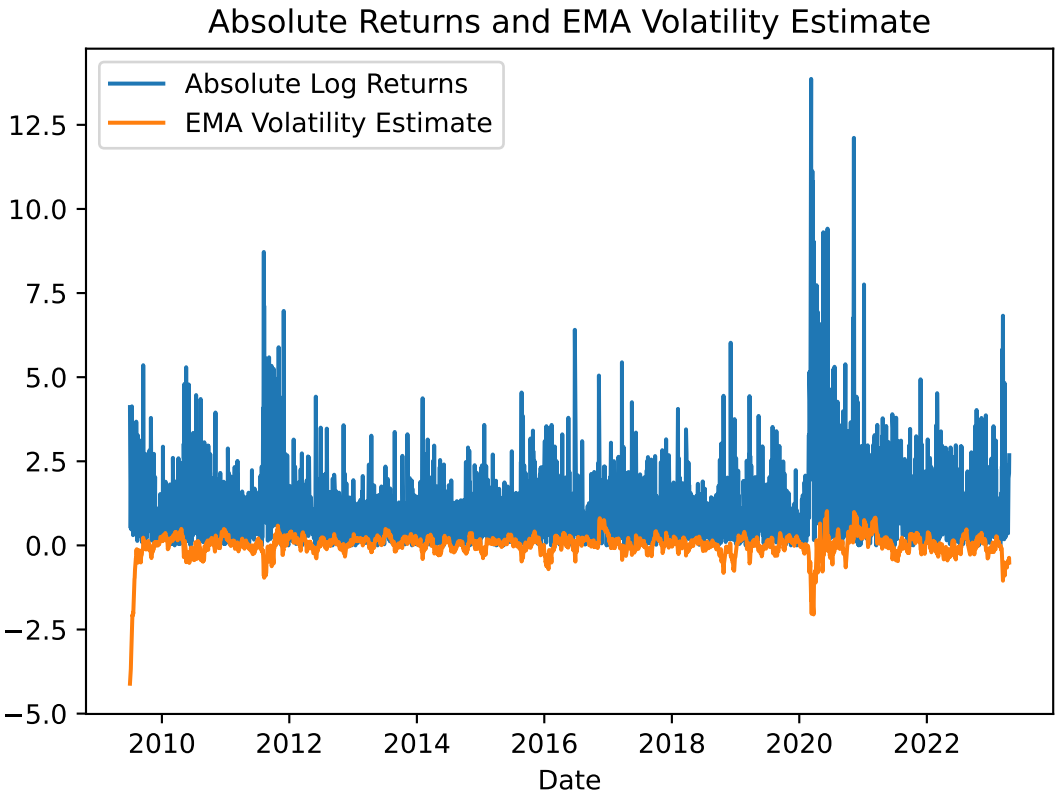
4

Modeling Volatily with EMA

```
ema_returns = log_returns.ewm(alpha=0.06, adjust=False).mean()

fig, ax = plt.subplots()
ax.plot(abs_returns, label='Absolute Log Returns')
ax.plot(ema_returns, label='EMA Volatility Estimate')
ax.legend()
ax.set_title('Absolute Returns and EMA Volatility Estimate')
ax.set_xlabel('Date')

plt.show()
```



Since EMA0.06 is a exponential moving average with smoothing parameter of 0.06, they are not exactly comparable but we could see observe the low persistence of the EMA estimate. Since we know that values presented in EMA are heavily weighted on the previous returns, it speaks to the volatility being around 0. Also we see that negative returns are much more highly present with the EMA estimate, showing large negative spikes with very few equivalent positives.

5

GARCH Modeling

```
# Estimate GARCH(1,1)
# A GARCH(1,1) process has p = 1 and q = 1.
model = arch.arch_model(log_returns.dropna(), p=1, q=1, vol='GARCH', dist = 'normal')
garch = model.fit()
```

```
## Iteration:      1,   Func. Count:      6,   Neg. LLF: 21910.418318209402
## Iteration:      2,   Func. Count:     15,   Neg. LLF: 279803071708.7909
## Iteration:      3,   Func. Count:     23,   Neg. LLF: 7303.9749620299535
## Iteration:      4,   Func. Count:     30,   Neg. LLF: 690917619.9726989
## Iteration:      5,   Func. Count:     36,   Neg. LLF: 6072.22681578124
## Iteration:      6,   Func. Count:     42,   Neg. LLF: 6054.230922510744
## Iteration:      7,   Func. Count:     48,   Neg. LLF: 6053.710043677962
## Iteration:      8,   Func. Count:     54,   Neg. LLF: 6053.69454456868
## Iteration:      9,   Func. Count:     59,   Neg. LLF: 6053.694447499933
## Iteration:     10,   Func. Count:     63,   Neg. LLF: 6053.694447501396
## Optimization terminated successfully    (Exit mode 0)
##             Current function value: 6053.694447499933
```

```
##             Iterations: 10
##             Function evaluations: 63
##             Gradient evaluations: 10
```

```
garch.params
```

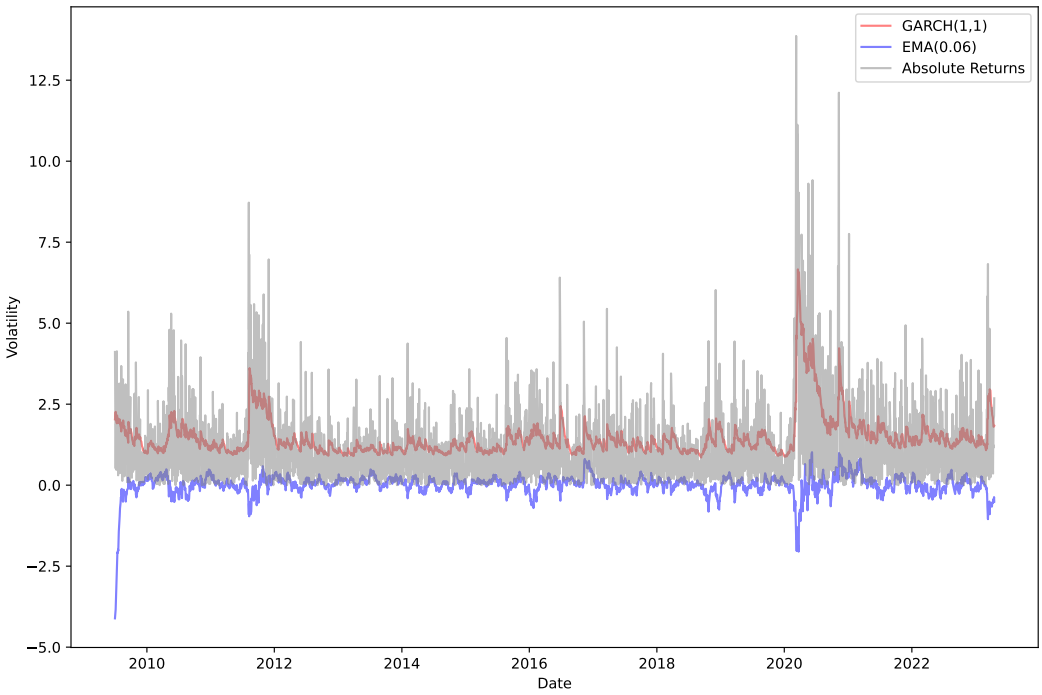
```
## mu          0.052886
## omega       0.052499
## alpha[1]    0.079138
## beta[1]     0.897672
## Name: params, dtype: float64
```

The sum of alpha and beta is 0.97681 Since the sum of alpha and beta is less than 1, the forecast of the conditional variance will converge to the unconditional variance as k(time) decreases. In other words the variance/volatility will go back down to the mean slowly in the long-run. The sum of the alpha and beta can differ based on the smoothing parameter(landa) which is the weight in which past observations are used to estimate data. the higher the landa the more weight is put on past observations and the lower the sum of alpha and beta.

6

```
# Creating a Conditional SD of GARCH(1,1)
garch1 = garch.conditional_volatility
# Plotting
fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(garch1, alpha = 0.5, label='GARCH(1,1)', color = 'r')
ax.plot(ema_returns, alpha= 0.5, label='EMA(0.06)', color = 'b')
ax.plot(abs_returns, alpha=0.5, label='Absolute Returns', color = 'grey')
ax.legend()
ax.set_xlabel('Date')
ax.set_ylabel('Volatility')

plt.show()
```



Comparing GARCH with EMA we see the difference in persistence, due to the GARCH model sum of alpha and beta = 0.97681 we observe high persistence on volatility shown in red, closely following the absolute returns making this much efficient. EMA assumes alpha + beta = 1 meaning volatility is never expected to revert back to the long run mean. EMA is also more likely to be affect by mu unlike GARCH where mu is brought down by the mean reversion term (omega).

7

```
# A GJR-GARCH(1,1) differs in the code with the addition of 0 = '1' before GARCH(1,1) only used two var
gjr_model = arch.arch_model(log_returns.dropna(), p=1, o=1, q=1, vol='GARCH')
gjr_garch = gjr_model.fit()
```

```
## Iteration:      1,   Func. Count:      7,   Neg. LLF: 8674432764092.129
## Iteration:      2,   Func. Count:     17,   Neg. LLF: 44719749275.99568
## Iteration:      3,   Func. Count:     26,   Neg. LLF: 15514.325357006948
## Iteration:      4,   Func. Count:     34,   Neg. LLF: 7463.3573740133015
## Iteration:      5,   Func. Count:     43,   Neg. LLF: 6087.880762674495
## Iteration:      6,   Func. Count:     50,   Neg. LLF: 6037.850274223741
## Iteration:      7,   Func. Count:     57,   Neg. LLF: 6058.625986918654
## Iteration:      8,   Func. Count:     64,   Neg. LLF: 6027.891711420398
## Iteration:      9,   Func. Count:     70,   Neg. LLF: 6027.886097226645
## Iteration:     10,   Func. Count:     76,   Neg. LLF: 6027.8860084060525
## Iteration:     11,   Func. Count:     81,   Neg. LLF: 6027.886008402769
## Optimization terminated successfully    (Exit mode 0)
##             Current function value: 6027.8860084060525
##             Iterations: 11
##             Function evaluations: 81
##             Gradient evaluations: 11
```

```
gjr_garch.summary()
```

```
## <class 'statsmodels.iolib.summary.Summary'>
## """
##                     Constant Mean - GJR-GARCH Model Results
## ==============================================================================
## Dep. Variable:              Adj Close   R-squared:                       0.000
## Mean Model:             Constant Mean   Adj. R-squared:                  0.000
## Vol Model:                  GJR-GARCH   Log-Likelihood:               -6027.89
## Distribution:                  Normal   AIC:                            12065.8
## Method:            Maximum Likelihood   BIC:                            12096.5
##                                         No. Observations:                 3472
## Date:                Tue, Apr 18 2023   Df Residuals:                     3471
## Time:                        21:22:45   Df Model:                            1
##                                  Mean Model
## ==============================================================================
##                  coef    std err          t      P>|t|       95.0% Conf. Int.
## ------------------------------------------------------------------------------
## mu             0.0233  2.118e-02      1.099      0.272 [-1.824e-02,6.480e-02]
##                               Volatility Model
## ==============================================================================
##                  coef    std err          t      P>|t|      95.0% Conf. Int.
## ------------------------------------------------------------------------------
## omega          0.0460  1.741e-02      2.645  8.177e-03 [1.192e-02,8.015e-02]
## alpha[1]       0.0216  9.835e-03      2.198  2.795e-02 [2.341e-03,4.089e-02]
## gamma[1]       0.0874  2.446e-02      3.576  3.494e-04   [3.951e-02,  0.135]
## beta[1]        0.9134  2.236e-02     40.849      0.000     [  0.870,  0.957]
## ==============================================================================
##
## Covariance estimator: robust
## """
```

From the summary we see evidence of an asymmetric effect on positive/negative returns on future volatility given by the fact that gamma > 0 it is also statistically significant at 1% by looking at the p value. This is consistent with the idea of volatility clustering, where periods of high volatility tend to be followed by periods of high volatility and vice versa.

8

```
# Creating two variables using both Garch Models forecasting 250 days
n_test = 250
train = log_returns[:-n_test]
test = log_returns[-n_test:]

model_garch = arch_model(train, vol='GARCH', p=1, q=1)
model_gjrgarch = arch_model(train, vol='GARCH', p=1, q=1, o=1)

fit_garch = model_garch.fit()
```

```
## Iteration:      1,   Func. Count:      6,   Neg. LLF: 19966.980262424506
## Iteration:      2,   Func. Count:     15,   Neg. LLF: 495534726697.76807
## Iteration:      3,   Func. Count:     23,   Neg. LLF: 6683.2083864732
## Iteration:      4,   Func. Count:     30,   Neg. LLF: 597920918.3160948
## Iteration:      5,   Func. Count:     36,   Neg. LLF: 5624.68497546266
```

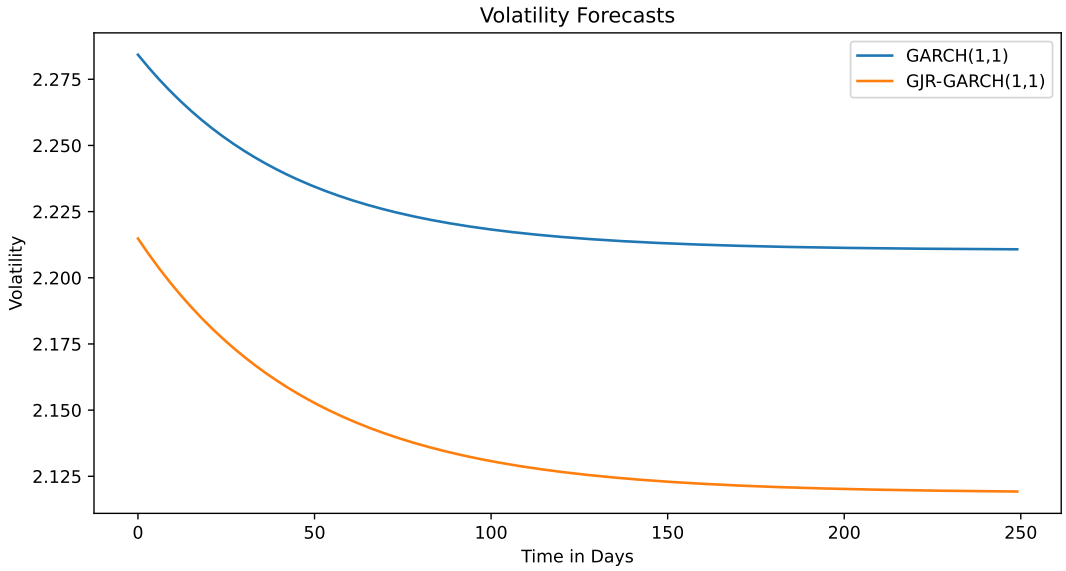
```
## Iteration:      6,   Func. Count:     42,   Neg. LLF: 5589.017917953144
## Iteration:      7,   Func. Count:     48,   Neg. LLF: 5588.133969559601
## Iteration:      8,   Func. Count:     53,   Neg. LLF: 5588.104121939268
## Iteration:      9,   Func. Count:     58,   Neg. LLF: 5588.103985899657
## Iteration:     10,   Func. Count:     63,   Neg. LLF: 5588.1039731496785
## Iteration:     11,   Func. Count:     67,   Neg. LLF: 5588.103973152345
## Optimization terminated successfully    (Exit mode 0)
##             Current function value: 5588.1039731496785
##             Iterations: 11
##             Function evaluations: 67
##             Gradient evaluations: 11
```

```
fit_gjrgarch =model_gjrgarch.fit()
```

```
## Iteration:      1,   Func. Count:      7,   Neg. LLF: 19645.74049116174
## Iteration:      2,   Func. Count:     17,   Neg. LLF: 373279.98671165836
## Iteration:      3,   Func. Count:     27,   Neg. LLF: 7538.330813951919
## Iteration:      4,   Func. Count:     35,   Neg. LLF: 633007020.2961174
## Iteration:      5,   Func. Count:     42,   Neg. LLF: 5579.78168789136
## Iteration:      6,   Func. Count:     49,   Neg. LLF: 5627.396748497842
## Iteration:      7,   Func. Count:     56,   Neg. LLF: 5576.16932284693
## Iteration:      8,   Func. Count:     63,   Neg. LLF: 5594.342151386633
## Iteration:      9,   Func. Count:     70,   Neg. LLF: 5565.654837121423
## Iteration:     10,   Func. Count:     77,   Neg. LLF: 5565.614085574276
## Iteration:     11,   Func. Count:     84,   Neg. LLF: 5565.606999517489
## Iteration:     12,   Func. Count:     90,   Neg. LLF: 5565.6068716284935
## Iteration:     13,   Func. Count:     95,   Neg. LLF: 5565.606871630826
## Optimization terminated successfully    (Exit mode 0)
##             Current function value: 5565.6068716284935
##             Iterations: 13
##             Function evaluations: 95
##             Gradient evaluations: 13
```

```
fc_garch = fit_garch.forecast(horizon = n_test)
fc_gjrgarch = fit_gjrgarch.forecast(horizon = n_test)

# Plotting
plt.figure(figsize=(10, 5))
plt.plot(fc_garch.variance.values[-1:].T, label='GARCH(1,1)')
plt.plot(fc_gjrgarch.variance.values[-1:].T, label='GJR-GARCH(1,1)')
plt.title('Volatility Forecasts')
plt.xlabel('Time in Days')
plt.ylabel('Volatility')
plt.legend()
plt.show()
```



Plotting the volatility forecast for both the GARCH model and the GJR-GARCH model for 250 days, We see them in a nearly parallel form. With the GJR model starting at much lower volatility at day 0 and with the biggest spread between day 0 volatility and day 250 volatility. GJR GARCH implies an asymmetric response to shocks compared to the GARCH model. The GJR-GARCH model adds on to GARCH by introducing an extra term to explain more negative effects on volatility. At the 250 day, GARCH volatility sits right below 2.225 while GJR GARCH is much lower below 2.125.