```
# Nifty50_GARCH Model
# Hands-on GARCH (ARCH family) volatility modelling for NIFTY 50
# Purpose: download Nifty data, fit GARCH(1,1), forecast volatility,
# and implement a volatility-scaled trading/backtest strategy
%pip install yfinance arch pandas numpy matplotlib statsmodels scipy empyrical
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.11/dist-packages (0.2.65)
Collecting arch
  Downloading arch-7.2.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.11/dist-packages (0.14.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (1.16.1)
Collecting empyrical
  Downloading empyrical-0.5.5.tar.gz (52 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 52.8/52.8 kB 2.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.3.8)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2025.2)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.11/dist-packages (from yfinance) (2.4.6)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.11/dist-packages (from yfinance) (3.18.2)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.11/dist-packages (from yfinance) (4.13.4)
Requirement already satisfied: curl_cffi>=0.7 in /usr/local/lib/python3.11/dist-packages (from yfinance) (0.13.0)
Requirement already satisfied: protobuf>=3.19.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (5.29.5)
Requirement already satisfied: websockets>=13.0 in /usr/local/lib/python3.11/dist-packages (from yfinance) (15.0.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.11/dist-packages (from statsmodels) (1.0.1)
Requirement already satisfied: pandas-datareader>=0.2 in /usr/local/lib/python3.11/dist-packages (from empyrical) (0.10.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.1->yfinance
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.11.
Requirement already satisfied: cffi>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from curl_cffi>=0.7->yfinance) (1.17.1
Requirement already satisfied: certifi>=2024.2.2 in /usr/local/lib/python3.11/dist-packages (from curl_cffi>=0.7->yfinance) (2
Requirement already satisfied: lxml in /usr/local/lib/python3.11/dist-packages (from pandas-datareader>=0.2->empyrical) (5.4.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinar
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.31->yfinance) (
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12.0->curl_cffi>=0.7->yfinan
Downloading arch-7.2.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (985 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 985.3/985.3 kB 16.8 MB/s eta 0:00:00
Building wheels for collected packages: empyrical
  Building wheel for empyrical (setup.py) ... done
  Created wheel for empyrical: filename=empyrical-0.5.5-py3-none-any.whl size=39752 sha256=2f54f51dad61ce9a265cf9baf2d352054a0
  Stored in directory: /root/.cache/pip/wheels/ac/1d/58/a7ae5ef5c8de7c4b769f24c2584f4706564921f031b16b9cb6
Successfully built empyrical
Installing collected packages: empyrical, arch
Successfully installed arch-7.2.0 empyrical-0.5.5
```

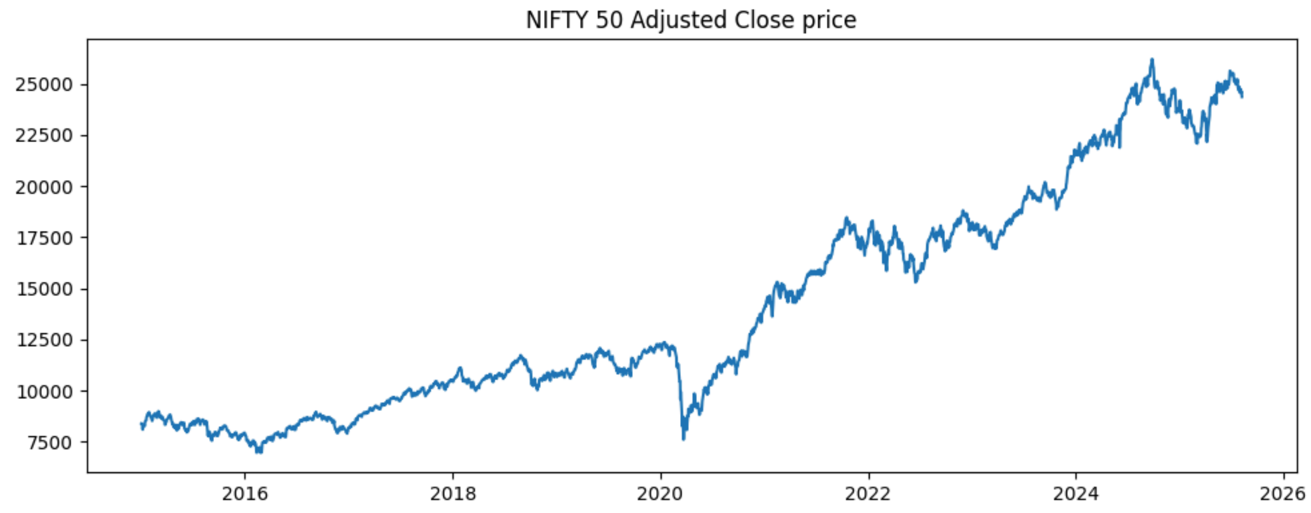
```
# 1) Imports libraries
# -----------------------
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from arch import arch_model
from statsmodels.tsa.stattools import adfuller
from datetime import datetime
import empyrical as emp
```

```
# 2) Parameters define
# -----------------------
TICKER = '^NSEI'           # Nifty 50 Yahoo ticker
START = '2015-01-01'       # change as needed
END = datetime.today().strftime('%Y-%m-%d')
TARGET_ANN_VOL = 0.12      # target annual volatility for position sizing (12%)
TRADING_DAYS = 252
```

```
# 3) Download data and compute returns
# -----------------------
print('Downloading data...')
prices = yf.download(TICKER, start=START, end=END, progress=False)
prices = prices[['Close']] # Select only 'Close'
prices = prices.dropna()
```

```
Downloading data...
/tmp/ipython-input-2603706579.py:4: FutureWarning: YF.download() has changed argument auto_adjust default to True
  prices = yf.download(TICKER, start=START, end=END, progress=False)
```

```
# Use log returns (preferred) or simple pct_change
log_returns = np.log(prices).diff().dropna()
returns = log_returns * 100.0   # percent returns for arch library (optional but common)


# Quick plot of price
plt.figure(figsize=(10,4))
plt.plot(prices)
plt.title('NIFTY 50 Adjusted Close price')
plt.tight_layout()
plt.show()
```

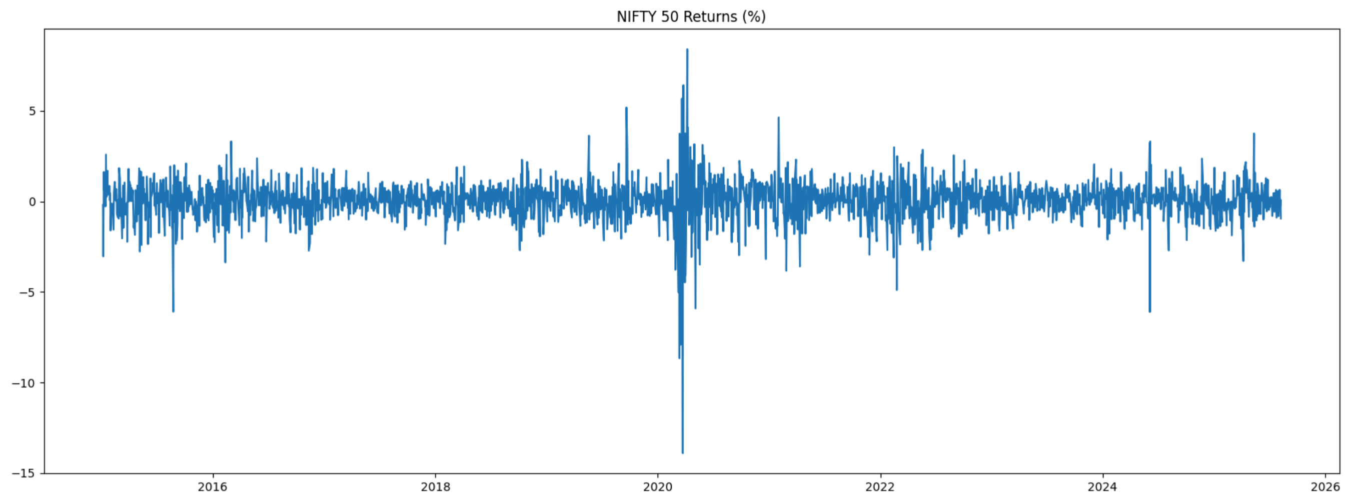


```
# 4) Stationarity check (ADF on log prices and returns)
# -----------------------
print('\nAugmented Dickey-Fuller tests:')
adf_price = adfuller(np.log(prices['Close']).dropna()) # Select 'Close' and dropna() to ensure 1D array
print(f'ADF on log(price):    stat={adf_price[0]:.4f}, p-value={adf_price[1]:.4f}')
adf_ret = adfuller(returns['Close']) # Select 'Close' and dropna() to ensure 1D array
print(f'ADF on returns(%):    stat={adf_ret[0]:.4f}, p-value={adf_ret[1]:.4f}')
```

```
Augmented Dickey-Fuller tests:
ADF on log(price):    stat=-0.2606, p-value=0.9309
ADF on returns(%):    stat=-14.1302, p-value=0.0000
```

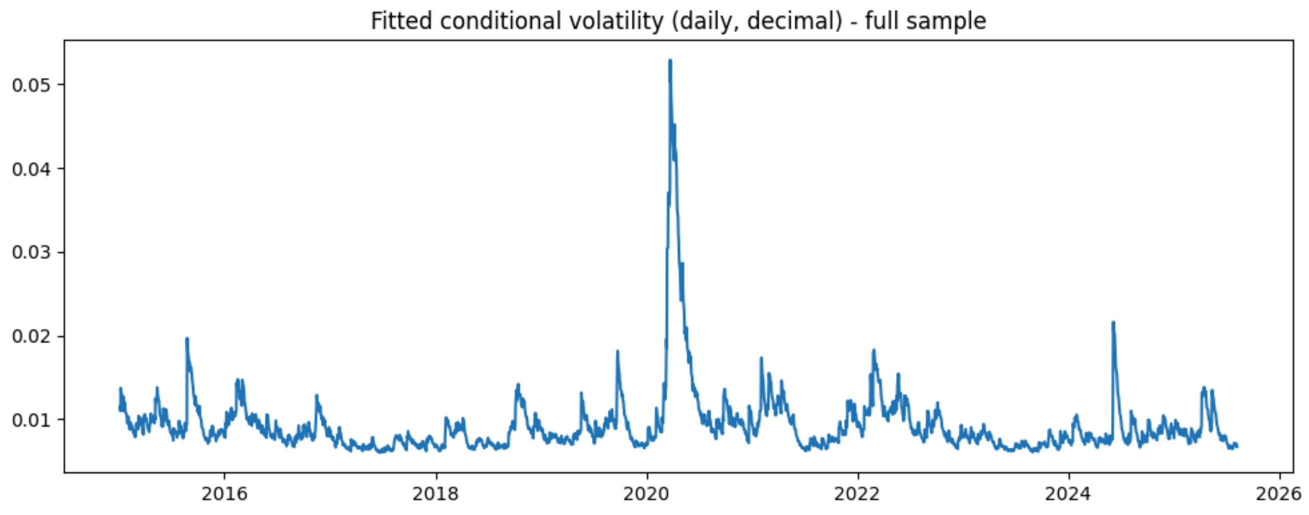
✦ Gemini

```
# 5) Visualize returns
# -----------------------
plt.figure(figsize=(16,6))
plt.plot(returns)
plt.title('NIFTY 50 Returns (%)')
plt.tight_layout()
plt.show()
```



```
#  Fit GARCH(1,1) - full-sample (for understanding)
# -----------------------
# We'll use a mean model with constant mean (you can change to AR terms)
model = arch_model(returns, mean='Constant', vol='GARCH', p=1, q=1, dist='t')
print('\nFitting GARCH(1,1) (Student-t errors) on full sample...')
res = model.fit(disp='off')
print(res.summary())

# Extract fitted conditional volatility (in same units as returns: percent)
cond_vol = res.conditional_volatility  # percent (since returns are percent)

# Convert to daily decimal volatility for plotting/usage: percent -> decimal
cond_vol_decimal = cond_vol / 100.0

plt.figure(figsize=(10,4))
plt.plot(cond_vol_decimal)
plt.title('Fitted conditional volatility (daily, decimal) - full sample')
plt.tight_layout()
plt.show()
```

```
Fitting GARCH(1,1) (Student-t errors) on full sample...
                        Constant Mean - GARCH Model Results
====================================================================================
Dep. Variable:          ('Close', '^NSEI')   R-squared:                       0.000
Mean Model:                  Constant Mean   Adj. R-squared:                  0.000
Vol Model:                           GARCH   Log-Likelihood:               -3313.89
Distribution:      Standardized Student's t   AIC:                          6637.77
Method:                 Maximum Likelihood   BIC:                          6667.11
                                             No. Observations:                2610
Date:                     Sat, Aug 09 2025   Df Residuals:                    2609
Time:                             03:57:23   Df Model:                           1
                                Mean Model
==========================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
--------------------------------------------------------------------------
mu             0.0769  1.554e-02      4.950  7.412e-07 [4.648e-02,  0.107]
                              Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.0295  8.441e-03      3.497  4.700e-04 [1.298e-02,4.607e-02]
alpha[1]       0.0806  1.539e-02      5.239  1.611e-07   [5.047e-02,  0.111]
beta[1]        0.8853  2.104e-02     42.073      0.000     [  0.844,  0.927]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             6.3568      0.760      8.359  6.311e-17 [  4.866,  7.847]
========================================================================

Covariance estimator: robust
```

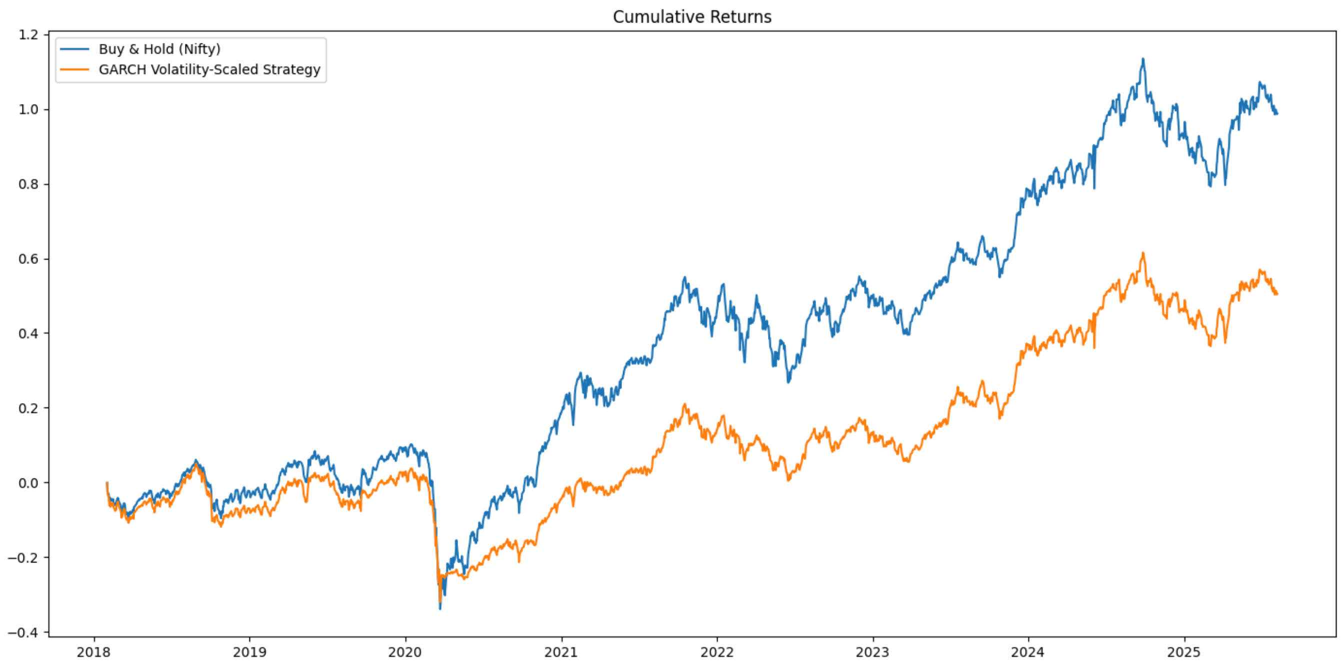


```
# Forecast next N days volatility (example: 5 days ahead)
# ----------------------
horizon = 5
forecast = res.forecast(horizon=horizon, reindex=False)
# forecast.variance is a DataFrame indexed by time with columns 1..horizon
last_var = forecast.variance.iloc[-1]
last_vol = np.sqrt(last_var) / 100.0   # convert percent^2 -> percent -> decimal
print(f'Forecasted daily vol for next {horizon} days (decimal):')
print(last_vol)
```

```
Forecasted daily vol for next 5 days (decimal):
h.1    0.007142
h.2    0.007226
h.3    0.007307
h.4    0.007384
h.5    0.007458
Name: 2025-08-08 00:00:00, dtype: float64
```

```
# ----------------------
# 7) Walk-forward / rolling forecasting + simple volatility scaling strategy
# ----------------------
# Idea: every rebalance period (e.g., monthly), fit GARCH on available history (expanding)
# then forecast next day volatility, and set daily position as:
# position_size = TARGET_ANN_VOL / forecasted_daily_vol_annualized
# daily_vol_annualized = forecasted_daily_vol * sqrt(TRADING_DAYS)
# To avoid extreme leverage, cap position between 0 and max_leverage.

retrain_window_years = 3          # use last 3 years of data for initial fit
retrain_window = int(retrain_window_years * TRADING_DAYS)
rebalance_days = 21               # roughly monthly rebalancing (21 trading days)
```

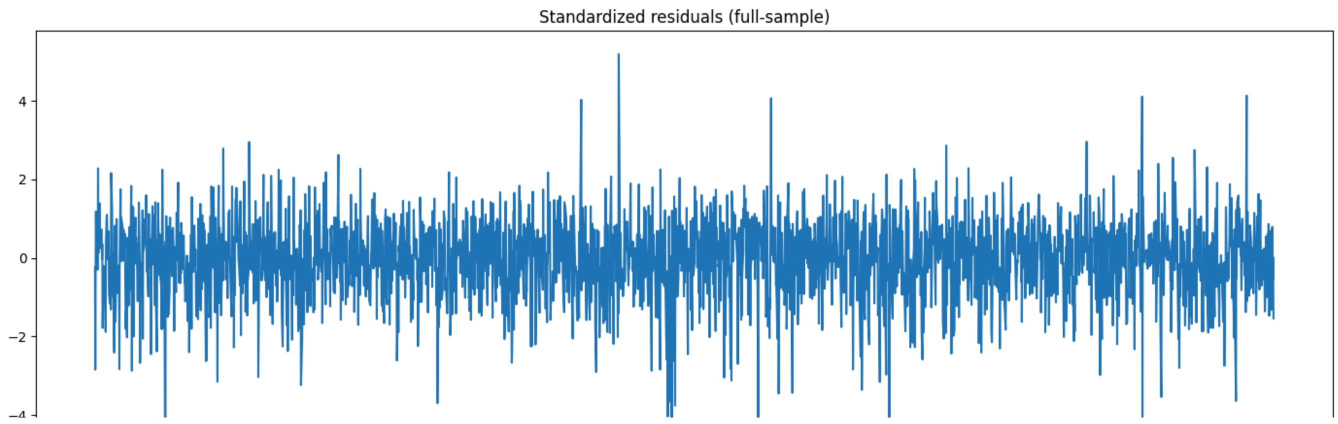
```
max_leverage = 3.0               # cap on position size

# Prepare DataFrame to store results
df = pd.DataFrame(index=returns.index)
df['returns'] = returns / 100.0   # convert back to decimal returns for portfolio math

# We'll perform an expanding fit but re-fit every rebalance_days for speed
start_idx = retrain_window
positions = []
strategy_returns = []
forecasted_daily_vols = []

for i in range(start_idx, len(df)-1):
    # Refit only on rebalance days to save compute
    if (i - start_idx) % rebalance_days == 0:
        train_slice = returns.iloc[:i]  # expanding history up to day i-1
        # Fit GARCH(1,1) quickly (suppress output)
        model_i = arch_model(train_slice, mean='Constant', vol='GARCH', p=1, q=1, dist='t')
        try:
            res_i = model_i.fit(disp='off')
            # One-step ahead forecast (horizon=1)
            f = res_i.forecast(horizon=1, reindex=False)
            var_f = f.variance.iloc[-1, 0]  # variance in percent^2
            vol_f_pct = np.sqrt(var_f)     # percent
            vol_f_dec = vol_f_pct / 100.0  # decimal daily vol
        except Exception as e:
            # In case fitting fails, fallback to historical rolling std
            vol_f_dec = float(train_slice.pct_change().dropna().std())
    # else, keep previous vol_f_dec
    forecasted_daily_vols.append(vol_f_dec)

# Build position sizes aligned to returns index (positions for day i will be used on day i+1)
# Convert to pandas Series
forecasted_daily_vols = pd.Series(forecasted_daily_vols, index=df.index[start_idx:len(df)-1])

# Position sizing
annualized_forecast = forecasted_daily_vols * np.sqrt(TRADING_DAYS)
position_size = TARGET_ANN_VOL / annualized_forecast
position_size = position_size.clip(lower=0.0, upper=max_leverage)  # no shorting here

# Align positions to returns (we use position computed at day t to trade next day t+1)
positions = position_size.shift(0)  # since our loop used returns until i-1 to forecast i's vol

# Compute strategy returns
# strategy_return_t = position_t * market_return_t
strategy_returns = positions * df['returns'].loc[positions.index]

# Put back into df for analysis (we'll align index properly)
df_strategy = pd.DataFrame(index=positions.index)
df_strategy['position'] = positions
# Fill any NaN position with 0
df_strategy['position'] = df_strategy['position'].fillna(0.0)
df_strategy['market_return'] = df['returns'].loc[df_strategy.index]
df_strategy['strategy_return'] = df_strategy['position'] * df_strategy['market_return']

# Compute cumulative returns
cum_market = (1 + df_strategy['market_return']).cumprod() - 1
cum_strategy = (1 + df_strategy['strategy_return']).cumprod() - 1

plt.figure(figsize=(14,7))
plt.plot(cum_market, label='Buy & Hold (Nifty)')
plt.plot(cum_strategy, label='GARCH Volatility-Scaled Strategy')
plt.legend()
plt.title('Cumulative Returns')
plt.tight_layout()
plt.show()
```



```
# Performance metrics
# -----------------------
def perf_stats(returns_series):
    # returns_series expected to be decimal daily returns
    ann_ret = emp.annual_return(returns_series)
    ann_vol = emp.annual_volatility(returns_series)
    sharpe = emp.sharpe_ratio(returns_series, risk_free=0.0)
    max_dd = emp.max_drawdown(returns_series)
    return ann_ret, ann_vol, sharpe, max_dd

market_stats = perf_stats(df_strategy['market_return'].dropna())
strat_stats = perf_stats(df_strategy['strategy_return'].dropna())

print('\nPerformance metrics:')
print('Metric\t\tMarket\t\tStrategy')
print(f'Annual Return\t{market_stats[0]:.2%}\t{strat_stats[0]:.2%}')
print(f'Annual Vol\t{market_stats[1]:.2%}\t{strat_stats[1]:.2%}')
print(f'Sharpe\t\t{market_stats[2]:.2f}\t\t{strat_stats[2]:.2f}')
print(f'Max Drawdown\t{market_stats[3]:.2%}\t{strat_stats[3]:.2%}')
```

```
Performance metrics:
Metric          Market          Strategy
Annual Return   9.79%   5.71%
Annual Vol      17.59%  14.19%
Sharpe          0.62            0.46
Max Drawdown    -40.04% -35.24%
```

```
#  Diagnostics
# -----------------------
# Check standardized residuals and their distribution
std_resid = res.std_resid.dropna()
plt.figure(figsize=(14,6))
plt.plot(std_resid)
plt.title('Standardized residuals (full-sample)')
plt.tight_layout()
plt.show()

# ACF of squared residuals to see remaining ARCH effects
from statsmodels.graphics.tsaplots import plot_acf
plt.figure(figsize=(10,4))
plot_acf(std_resid**2, lags=30)
plt.title('ACF of squared standardized residuals')
plt.tight_layout()
plt.show()
```


Standardized residuals (full-sample)