225-project

December 11, 2025

1 ECE 225A Project: Regime Aware Risk in the S&P 500

Authors: Peter Quawas, Ryan Luo
Course: ECE 225A
Date: December 2025

Abstract: This notebook investigates whether modeling volatility regimes improves risk assessment and portfolio allocation for the S&P 500 during the first half of 2025. We fit a GARCH(1,1) model with Student-t innovations, apply a two state Markov switching model to identify high and low volatility regimes, compute sector level tail risk using Value at Risk (VaR) and Expected Shortfall (ES), and compare equal weight, inverse volatility, and minimum variance portfolios.

1.1 1. Data Description

- **Source:** Kaggle dataset "S&P 500 Stocks Trade Data for First 6 Months of 2025"
- **Contents:** Daily open, close, and volume for 503 S&P 500 constituents
- **Time period:** January 2 to June 30, 2025 (122 trading days)
- **Identifiers:** `company_name` and `ticker` for each stock
- **Supplementary data:** Sector mappings from GitHub `datasets/s-and-p-500-companies`

1.1.1 Question and Motivation

Can regime aware modeling improve risk assessment and portfolio allocation?

Significance: - Financial markets exhibit volatility with distinct calm and turbulent periods - Traditional risk measures such as unconditional VaR may underestimate risk during transitions - Portfolio strategies that adapt to volatility can improve risk adjusted returns - Sector level tail risk varies across sectors, suggesting diversification benefits

We investigate whether identifying high and low volatility regimes via Markov switching leads to: 1. Better understanding of tail risk (VaR and ES) across sectors 2. Improved portfolio allocation (comparing equal weight, inverse volatility, and minimum variance portfolios) 3. More accurate risk assessment during regime transitions

1.2 2. Preprocessing and Log Returns

This section loads the raw CSV data, reshapes it from wide to long format, and computes daily log returns for each stock. Log returns are additive over time and approximate normality better than simple returns.

The data loader checks for the Kaggle input path first. If unavailable, it falls back to `sp500_2025_h1.csv` in the repository root.

- Source: Kaggle dataset "S&P 500 stocks trade data for first 6 months of 2025" (daily open/close/volume per constituent)
- **Structure:** 503 tickers × daily columns (e.g., `02-01-2025_opening`, `02-01-2025_closing`, `02-01-2025_volume`) plus `company_name` and `ticker` identifiers
- **Time span:** 2025-01-03 to 2025-06-30 (121 trading days after differencing)
- **Sector mappings:** GitHub `datasets/s-and-p-500-companies` constituents file

1.2.1 Reshaping and Return Calculation

The code below reshapes the wide format data into a tidy panel (one row per ticker and date), then computes log returns as $r_t = \ln(P_t/P_{t-1})$.

[42]:
```
%pip install -q numpy pandas matplotlib statsmodels scikit-learn scipy arch
```

```
[notice] A new release of pip is
available: 24.3.1 -> 25.3
[notice] To update, run:
python3.11 -m pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

[43]:
```
import os
import random
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.covariance import LedoitWolf
from scipy.stats import norm
from arch import arch_model

plt.style.use("seaborn-v0_8-darkgrid")

random.seed(0)
np.random.seed(0)

kaggle_path = Path("/kaggle/input/
 ↪s-and-p-500-stocks-trade-data-for-first-6-month-2025/sp500_2025_h1.csv")
local_path = Path("sp500_2025_h1.csv")

if kaggle_path.exists():
    data_path = kaggle_path
elif local_path.exists():
    data_path = local_path
```

```
else:
    raise FileNotFoundError("Could not find sp500_2025_h1.csv in Kaggle input␣
 ↪or repo root.")

df = pd.read_csv(data_path)

print("Shape:", df.shape)
print(df.head())
```

```
Shape: (503, 368)
     company_name ticker  02-01-2025_opening  02-01-2025_closing  \
0          Nvidia   NVDA             136.000             138.310
1       Microsoft   MSFT             425.530             418.580
2      Apple Inc.   AAPL             248.657             243.582
3          Amazon   AMZN             222.030             220.220
4  Meta Platforms   META             589.720             599.240

   02-01-2025_volume  03-01-2025_opening  03-01-2025_closing  \
0          198247166             140.010             144.470
1           16896469             421.080             423.350
2           55802016             243.093             243.093
3           33956579             222.505             224.190
4           12682269             604.760             604.630

   03-01-2025_volume  06-01-2025_opening  06-01-2025_closing  …  \
0          229322478             148.590             149.430  …
1           16662943             428.000             427.850  …
2           40288361             244.042             244.731  …
3           27515606             226.780             227.610  …
4           11436784             611.825             630.200  …

   25-06-2025_volume  26-06-2025_opening  26-06-2025_closing  \
0          269146471             155.975              155.02
1           17495099             492.980              497.45
2           39525730             201.430              201.00
3           31755698             213.120              217.12
4            9320436             714.355              726.09

   26-06-2025_volume  27-06-2025_opening  27-06-2025_closing  \
0          198145746             156.040              157.75
1           21578853             497.550              495.94
2           50799121             201.890              201.08
3           50480814             219.920              223.30
4           13964793             726.515              733.63

   27-06-2025_volume  30-06-2025_opening  30-06-2025_closing  \
0          263234539              158.40              157.99
1           34539236              497.04              497.41
```

```
2          73188571              202.01              205.17
3         119217138              223.52              219.39
4          18775735              744.55              738.09

   30-06-2025_volume
0          194580316
1           28368991
2           91912816
3           58887780
4           15402105

[5 rows x 368 columns]
```

The following code performs the wide to long transformation and computes log returns:

```
[44]: id_cols = ["company_name", "ticker"]
all_cols = df.columns.tolist()

# opening/closing/volume columns
open_cols = [c for c in all_cols if c.endswith("_opening")]
close_cols = [c for c in all_cols if c.endswith("_closing")]
volume_cols = [c for c in all_cols if c.endswith("_volume")]

print(f"# opening columns: {len(open_cols)}")
print(f"# closing columns: {len(close_cols)}")
print(f"# volume columns:  {len(volume_cols)}")

# daily close prices into long format
df_close = df.melt(
    id_vars=id_cols,
    value_vars=close_cols,
    var_name="date_col",
    value_name="close"
)
df_close["date_str"] = df_close["date_col"].str.replace("_closing", "",␣
 ↪regex=False)
df_close = df_close.drop(columns=["date_col"])

# daily opening prices into long format
df_open = df.melt(
    id_vars=id_cols,
    value_vars=open_cols,
    var_name="date_col",
    value_name="open"
)
df_open["date_str"] = df_open["date_col"].str.replace("_opening", "",␣
 ↪regex=False)
df_open = df_open.drop(columns=["date_col"])
```

```python
if len(volume_cols) > 0:
    df_vol = df.melt(
        id_vars=id_cols,
        value_vars=volume_cols,
        var_name="date_col",
        value_name="volume"
    )
    df_vol["date_str"] = df_vol["date_col"].str.replace("_volume", "",␣
 ↪regex=False)
    df_vol = df_vol.drop(columns=["date_col"])
else:
    df_vol = None

# merge close and open prices & compute log returns
df_long = df_close.merge(
    df_open,
    on = id_cols + ["date_str"],
    how = "left",
    suffixes = ("_close", "_open")
)
if df_vol is not None:
    df_long = df_long.merge(
        df_vol[id_cols + ["date_str", "volume"]],
        on = id_cols + ["date_str"],
        how = "left"
    )

df_long["date"] = pd.to_datetime(df_long["date_str"], format = "%d-%m-%Y")
df_long = df_long.sort_values(["ticker", "date"])

df_long["log_return"] = (
    df_long.groupby("ticker")["close"].transform(lambda x: np.log(x).diff())
)

# drop rows with missing log returns
df_long = df_long.dropna(subset = ["log_return"])

print(df_long.head())
print("Unique tickers:", df_long["ticker"].nunique())
print("Date range:", df_long["date"].min(), "→", df_long["date"].max())
```

```
# opening columns: 122
# closing columns: 122
# volume columns:  122
              company_name ticker   close    date_str     open   volume  \
753   Agilent Technologies      A  135.69  03-01-2025  133.525  1246919
1256  Agilent Technologies      A  136.43  06-01-2025  135.340  1047034
```

```
1759  Agilent Technologies      A  137.41  07-01-2025  135.980  1056693
2262  Agilent Technologies      A  137.00  08-01-2025  137.220  1684573
2765  Agilent Technologies      A  137.47  10-01-2025  135.195  1369875

           date  log_return
753  2025-01-03    0.016796
1256 2025-01-06    0.005439
1759 2025-01-07    0.007157
2262 2025-01-08   -0.002988
2765 2025-01-10    0.003425
Unique tickers: 503
Date range: 2025-01-03 00:00:00 → 2025-06-30 00:00:00
```

1.3 3. Time Series Diagnostics

Before modeling, we examine the statistical properties of returns to justify our modeling choices. We construct an equal weight pseudo index and inspect:

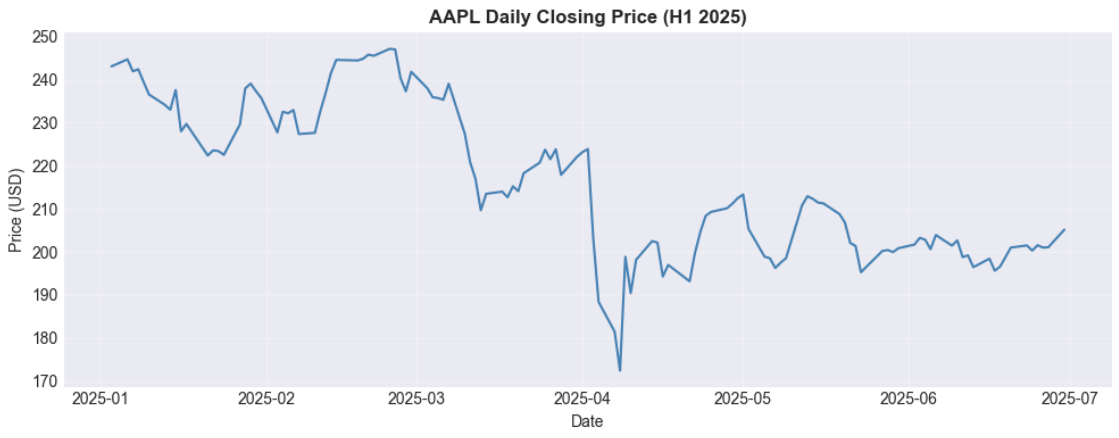
1. **Distribution shape:** Histogram and QQ plot to check for departures from normality (fat tails)
2. **Autocorrelation of returns:** ACF to detect predictable patterns in returns
3. **Autocorrelation of squared returns:** ACF to detect volatility clustering (conditional heteroskedasticity)

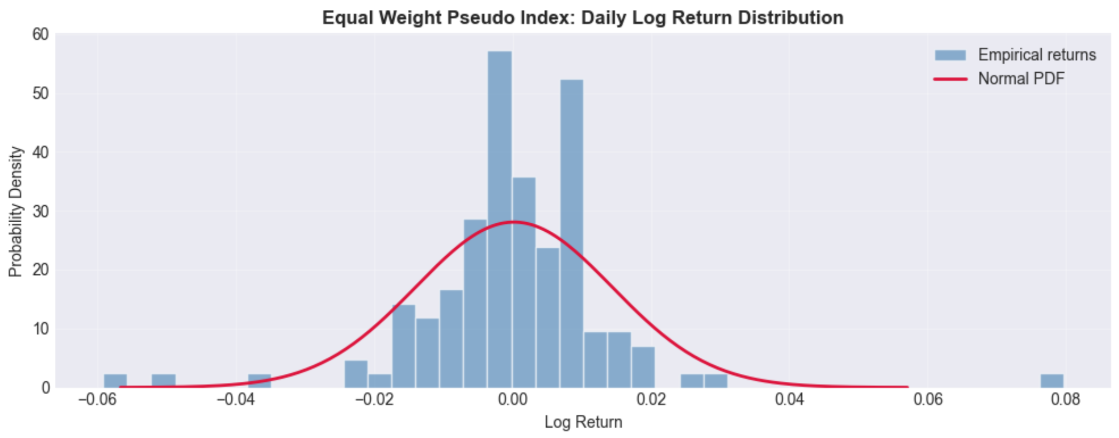
[45]:
```python
# equal weight pseudo index: average log return across all tickers each day
index_returns = (
    df_long.groupby("date")["log_return"]
    .mean()
    .sort_index()
)
index_returns.name = "sp500_eqw"

# sample stock for price path visualization
tickers = df_long["ticker"].unique()
sample_ticker = "AAPL" if "AAPL" in tickers else tickers[0]
print("Sample ticker:", sample_ticker)

sample_df = (
    df_long[df_long["ticker"] == sample_ticker]
    .sort_values("date")
    .set_index("date")
)

# figure 1: sample stock price path
plt.figure(figsize=(10, 4))
plt.plot(sample_df.index, sample_df["close"], color="steelblue", linewidth=1.5)
plt.title(f"{sample_ticker} Daily Closing Price (H1 2025)", fontsize=12,␣
 ↪fontweight="bold")
```

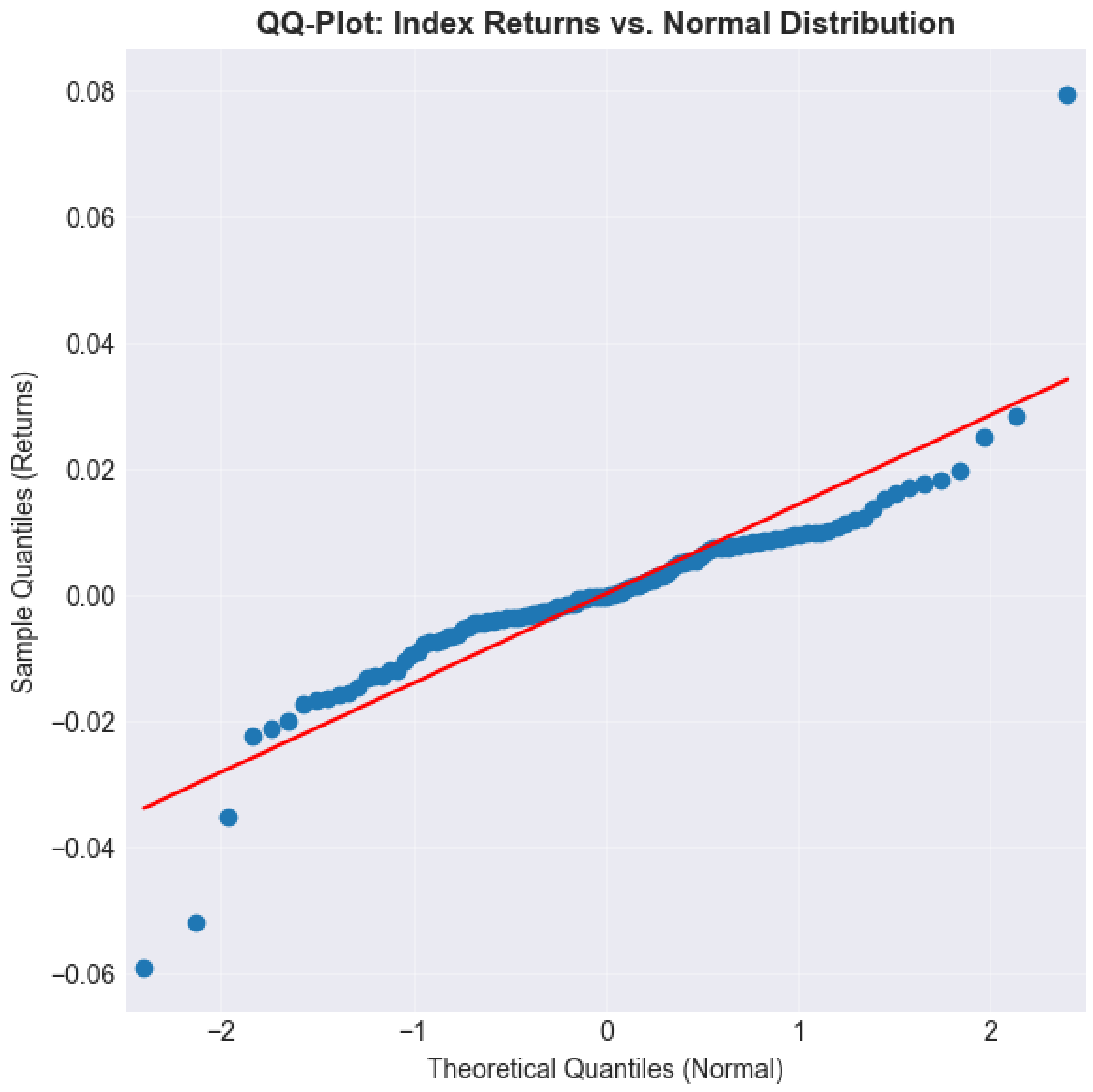
```
plt.xlabel("Date")
plt.ylabel("Price (USD)")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

r = index_returns.dropna()
mu = r.mean()
sigma = r.std()

# figure 2: return distribution vs Normal (distribution has fatter tails,␣
↪excess kurtosis)
plt.figure(figsize=(10, 4))
plt.hist(r, bins=40, density=True, alpha=0.6, color="steelblue",␣
↪edgecolor="white", label="Empirical returns")
x = np.linspace(mu - 4 * sigma, mu + 4 * sigma, 200)
plt.plot(x, norm.pdf(x, mu, sigma), lw=2, color="crimson", label="Normal PDF")
plt.title("Equal Weight Pseudo Index: Daily Log Return Distribution",␣
↪fontsize=12, fontweight="bold")
plt.xlabel("Log Return")
plt.ylabel("Probability Density")
plt.legend(loc="upper right")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# figure 3: QQ plot (points deviate in both tails, confirming fat tails)
fig = sm.qqplot(r, line="s")
fig.set_size_inches(6, 6)
ax = fig.axes[0]
ax.set_title("QQ-Plot: Index Returns vs. Normal Distribution", fontsize=12,␣
↪fontweight="bold")
ax.set_xlabel("Theoretical Quantiles (Normal)")
ax.set_ylabel("Sample Quantiles (Returns)")
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# figure 4: ACF of returns (little autocorrelation, mostly within confidence␣
↪bands)
fig, ax = plt.subplots(figsize=(10, 4))
plot_acf(r, lags=20, ax=ax)
ax.set_title("Autocorrelation Function (ACF) of Daily Returns", fontsize=12,␣
↪fontweight="bold")
ax.set_xlabel("Lag (Days)")
ax.set_ylabel("Autocorrelation")
```

```
plt.tight_layout()
plt.show()

# figure 5: ACF of squared returns (significant autocorrelation, evidence of␣
↪volatility clustering)
fig, ax = plt.subplots(figsize=(10, 4))
plot_acf(r ** 2, lags=20, ax=ax)
ax.set_title("ACF of Squared Returns (Volatility Clustering)", fontsize=12,␣
↪fontweight="bold")
ax.set_xlabel("Lag (Days)")
ax.set_ylabel("Autocorrelation")
plt.tight_layout()
plt.show()
```

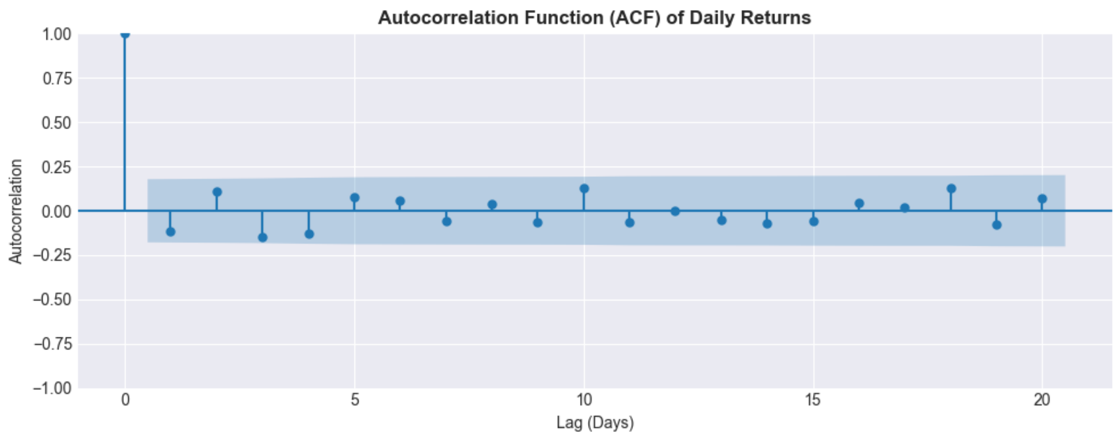
```
Sample ticker: AAPL
```



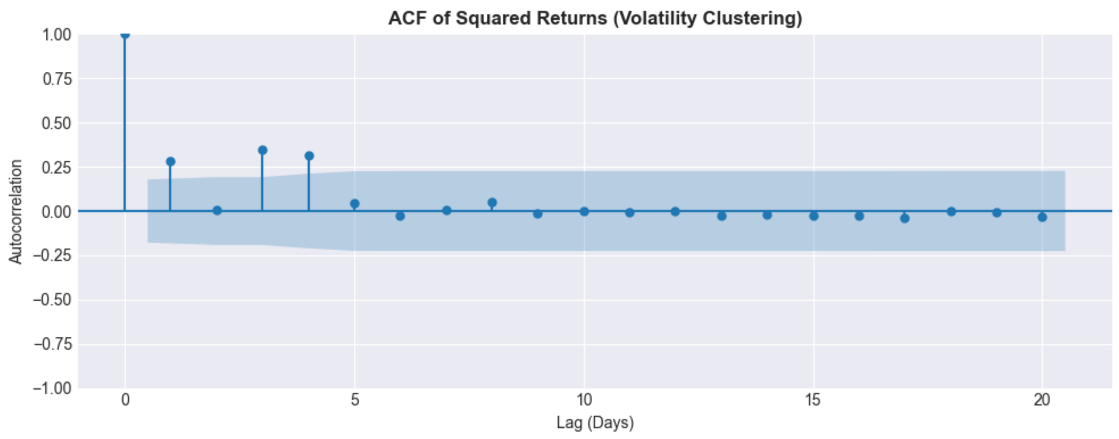




QQ-Plot: Index Returns vs. Normal Distribution
Sample Quantiles (Returns)
Theoretical Quantiles (Normal)
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−2
−1
0
1
2



Autocorrelation Function (ACF) of Daily Returns
Autocorrelation
Lag (Days)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20



1.4 4. Volatility Modeling (GARCH)

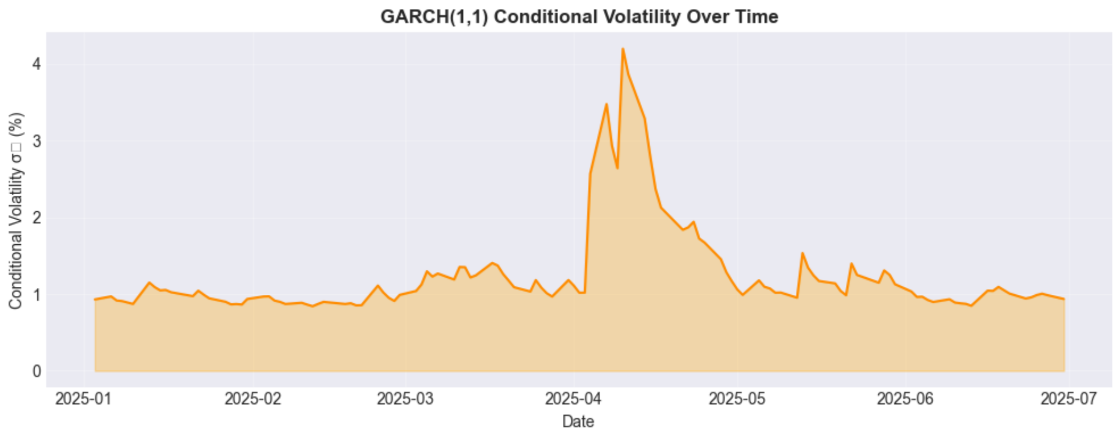
Given the evidence of volatility clustering from Section 3, we fit a GARCH(1,1) model with Student-t innovations to capture time varying conditional volatility. The GARCH(1,1) specification is:

$$\sigma_t^2 = \omega + \alpha\epsilon_{t-1}^2 + \beta\sigma_{t-1}^2$$

where $\alpha + \beta < 1$ ensures stationarity. We use Student-t innovations to accommodate the fat tails observed in the QQ plot.

[46]:
```
# scale to percentage points for typical GARCH parameter magnitudes
index_returns_pct = r * 100

# Student-t GARCH(1,1) for fat tails and volatility clustering
am = arch_model(index_returns_pct, vol="Garch", p=1, q=1, dist="t")
res_garch = am.fit(update_freq=5, disp="off")

print(res_garch.summary())

# extract conditional volatility and align index with returns
cond_vol = res_garch.conditional_volatility
if isinstance(cond_vol, pd.Series):
    cond_vol = cond_vol.reindex(r.index)
else:
    cond_vol = pd.Series(cond_vol, index=r.index, name="cond_vol")

# figure 6: GARCH conditional volatility (spikes during turbulent periods, mean␣
 ↪reverts during calm)
plt.figure(figsize=(10, 4))
plt.plot(cond_vol.index, cond_vol.values, color="darkorange", linewidth=1.5)
plt.fill_between(cond_vol.index, 0, cond_vol.values, alpha=0.3, color="orange")
plt.title("GARCH(1,1) Conditional Volatility Over Time", fontsize=12,␣
  ↪fontweight="bold")
plt.xlabel("Date")
plt.ylabel("Conditional Volatility   (%)")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
                        Constant Mean - GARCH Model Results
================================================================================
====
Dep. Variable:                     sp500_eqw   R-squared:
0.000
Mean Model:                    Constant Mean   Adj. R-squared:
0.000
Vol Model:                             GARCH   Log-Likelihood:
-179.855
Distribution:      Standardized Student's t   AIC:
369.711
Method:                   Maximum Likelihood   BIC:
383.690
                                              No. Observations:
121
Date:                       Thu, Dec 11 2025   Df Residuals:
120
Time:                               21:44:04   Df Model:
1
                                 Mean Model
===========================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
---------------------------------------------------------------------------
mu             0.0936  7.816e-02      1.198      0.231 [-5.955e-02,  0.247]
                              Volatility Model
===========================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
---------------------------------------------------------------------------
omega          0.1933      0.126      1.530      0.126 [-5.426e-02,  0.441]
alpha[1]       0.2045      0.123      1.667  9.544e-02 [-3.590e-02,  0.445]
beta[1]        0.6845      0.117      5.834  5.408e-09    [  0.455,  0.914]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             4.0522      1.388      2.919  3.506e-03 [  1.332,  6.773]
========================================================================
Covariance estimator: robust

/var/folders/nz/c2k7vtjn0fj1yg5s18_dn6hc0000gn/T/ipykernel_55596/259260619.py:25
: UserWarning: Glyph 8348 (\N{LATIN SUBSCRIPT SMALL LETTER T}) missing from
font(s) Arial.
  plt.tight_layout()
/Users/ryanluo/Library/Python/3.11/lib/python/site-
packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 8348 (\N{LATIN
SUBSCRIPT SMALL LETTER T}) missing from font(s) Arial.
  fig.canvas.print_figure(bytes_io, **kw)
```



1.5 5. Regime Switching (Markov Switching Model)

We fit a two state Markov switching model to identify distinct volatility regimes:

- **Low volatility regime:** Calm market conditions with smaller daily moves
- **High volatility regime:** Turbulent periods with large swings

The model estimates regime specific means and variances, plus transition probabilities $P(S_t = j|S_{t-1} = i)$. We extract smoothed regime probabilities to classify each day and visualize how regimes align with market movements.

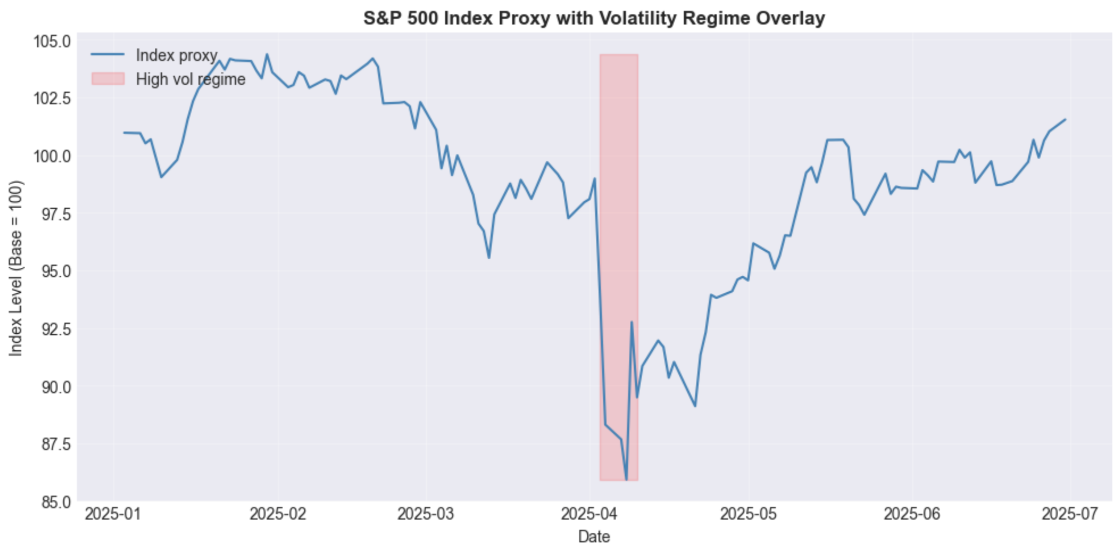
[47]:
```python
from statsmodels.tsa.regime_switching.markov_regression import MarkovRegression

# two state Markov switching on index returns (switching variance)
y = r.values
mod_ms = MarkovRegression(
    y,
    k_regimes=2,
    trend="c",
    switching_variance=True
)
res_ms = mod_ms.fit(em_iter=50, search_reps=20)
print(res_ms.summary())
```

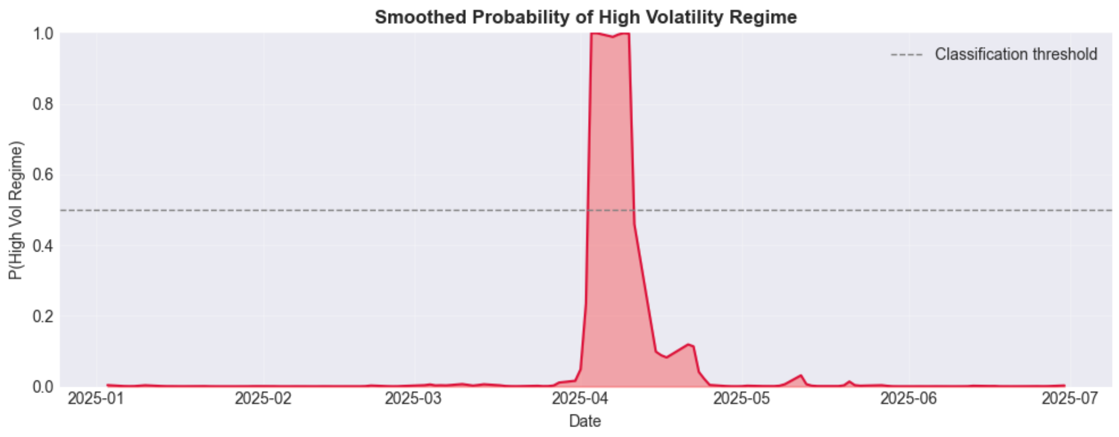
```
# identify high volatility regime by computing empirical variance using␣
 ↪smoothed probabilities
prob_raw = res_ms.smoothed_marginal_probabilities
if hasattr(prob_raw, "values"):
    prob_0 = pd.Series(prob_raw[0].values, index=r.index)
    prob_1 = pd.Series(prob_raw[1].values, index=r.index)
else:
    prob_0 = pd.Series(prob_raw[:, 0], index=r.index)
    prob_1 = pd.Series(prob_raw[:, 1], index=r.index)

# get regime means from params
params = res_ms.params
if isinstance(params, pd.Series):
    const_keys = [k for k in params.index if "const" in str(k).lower()]
    mu_0 = params[const_keys[0]] if len(const_keys) > 0 else 0
    mu_1 = params[const_keys[1]] if len(const_keys) > 1 else 0
else:
    # compute means empirically if params is array
    mu_0 = (r.values * prob_0.values).sum() / prob_0.sum()
    mu_1 = (r.values * prob_1.values).sum() / prob_1.sum()

# compute weighted variance for each regime: E[(r - )²] weighted by regime␣
 ↪probabilities
sigma2_0 = ((r.values - mu_0)**2 * prob_0.values).sum() / prob_0.sum()
sigma2_1 = ((r.values - mu_1)**2 * prob_1.values).sum() / prob_1.sum()

high_vol_regime_idx = 1 if sigma2_1 > sigma2_0 else 0
print(f"\nRegime 0 sigma²: {sigma2_0:.6f}, Regime 1 sigma²: {sigma2_1:.6f}")
print(f"High volatility regime: {high_vol_regime_idx}")

# smoothed probability of the high volatility regime
prob_raw = res_ms.smoothed_marginal_probabilities
if hasattr(prob_raw, "values"):
    prob_high = pd.Series(prob_raw[high_vol_regime_idx].values, index=r.index,␣
 ↪name="prob_high")
else:
    prob_high = pd.Series(prob_raw[:, high_vol_regime_idx], index=r.index,␣
 ↪name="prob_high")
regime_series = (prob_high > 0.5).astype(int)
regime_series.name = "high_vol_regime"

index_level = (1 + r).cumprod() * 100

# figure 7: index level with regime overlay (red shading marks high volatility␣
 ↪periods)
plt.figure(figsize=(10, 5))
```

```python
plt.plot(index_level.index, index_level.values, label="Index proxy",␣
↪color="steelblue", linewidth=1.5)
high_mask = regime_series == 1
plt.fill_between(
    index_level.index,
    index_level.min(),
    index_level.max(),
    where=high_mask,
    color="red",
    alpha=0.15,
    label="High vol regime"
)
plt.title("S&P 500 Index Proxy with Volatility Regime Overlay", fontsize=12,␣
↪fontweight="bold")
plt.xlabel("Date")
plt.ylabel("Index Level (Base = 100)")
plt.legend(loc="upper left")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# figure 8: smoothed regime probabilities (near 1 = high vol, near 0 = low vol)
plt.figure(figsize=(10, 4))
plt.plot(prob_high.index, prob_high.values, color="crimson", linewidth=1.5)
plt.fill_between(prob_high.index, 0, prob_high.values, alpha=0.3, color="red")
plt.axhline(y=0.5, color="gray", linestyle="--", linewidth=1,␣
↪label="Classification threshold")
plt.title("Smoothed Probability of High Volatility Regime", fontsize=12,␣
↪fontweight="bold")
plt.xlabel("Date")
plt.ylabel("P(High Vol Regime)")
plt.ylim(0, 1)
plt.legend(loc="upper right")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
                        Markov Switching Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  121
Model:               MarkovRegression   Log Likelihood                 377.421
Date:                Thu, 11 Dec 2025   AIC                           -742.841
Time:                        21:44:04   BIC                           -726.067
Sample:                             0   HQIC                          -736.028
                                - 121
Covariance Type:               approx
                             Regime 0 parameters
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0010      0.001      1.100      0.272      -0.001       0.003
sigma2      8.449e-05   1.18e-05      7.144      0.000    6.13e-05       0.000
                             Regime 1 parameters
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0105      0.016     -0.656      0.512      -0.042       0.021
sigma2         0.0018      0.001      1.735      0.083      -0.000       0.004
                         Regime transition parameters
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
p[0->0]        0.9896      0.011     92.333      0.000       0.969       1.011
p[1->0]        0.1648      0.152      1.085      0.278      -0.133       0.463
==============================================================================

Warnings:
[1] Covariance matrix calculated using numerical (complex-step) differentiation.

Regime 0 sigma²: 0.000084, Regime 1 sigma²: 0.001775
High volatility regime: 1
```





1.6 6. Sector Risk: Value at Risk and Expected Shortfall

We assess tail risk at the sector level using two standard risk measures:

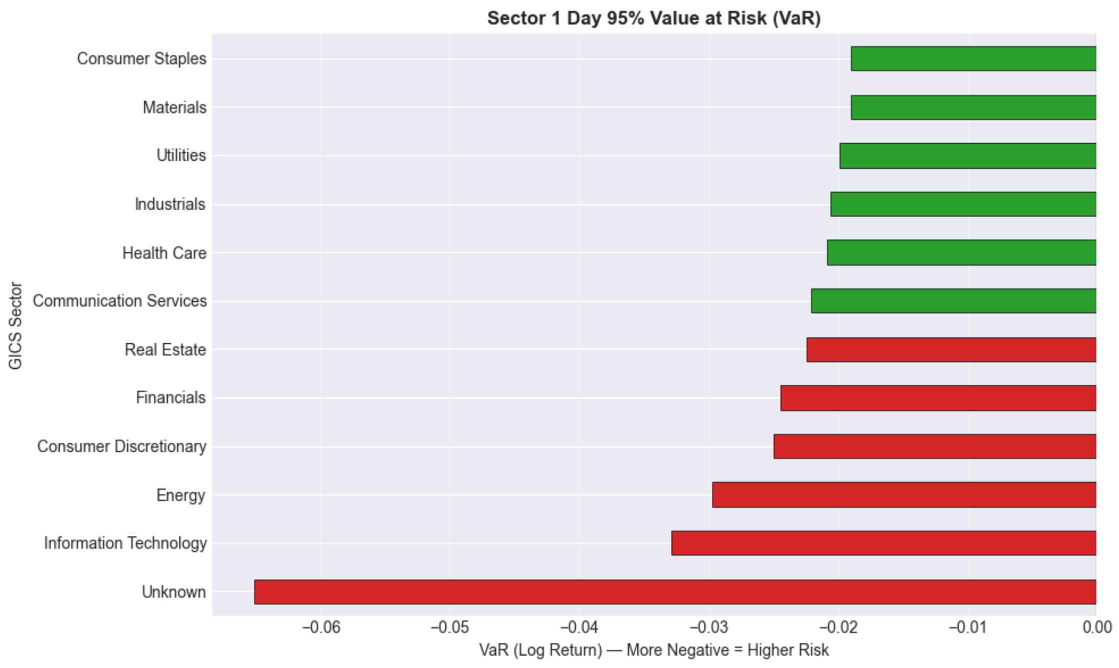
- **Value at Risk (VaR):** The α-quantile loss; at 95% confidence, the loss exceeded only 5% of the time
- **Expected Shortfall (ES):** The average loss conditional on exceeding VaR (also called CVaR)

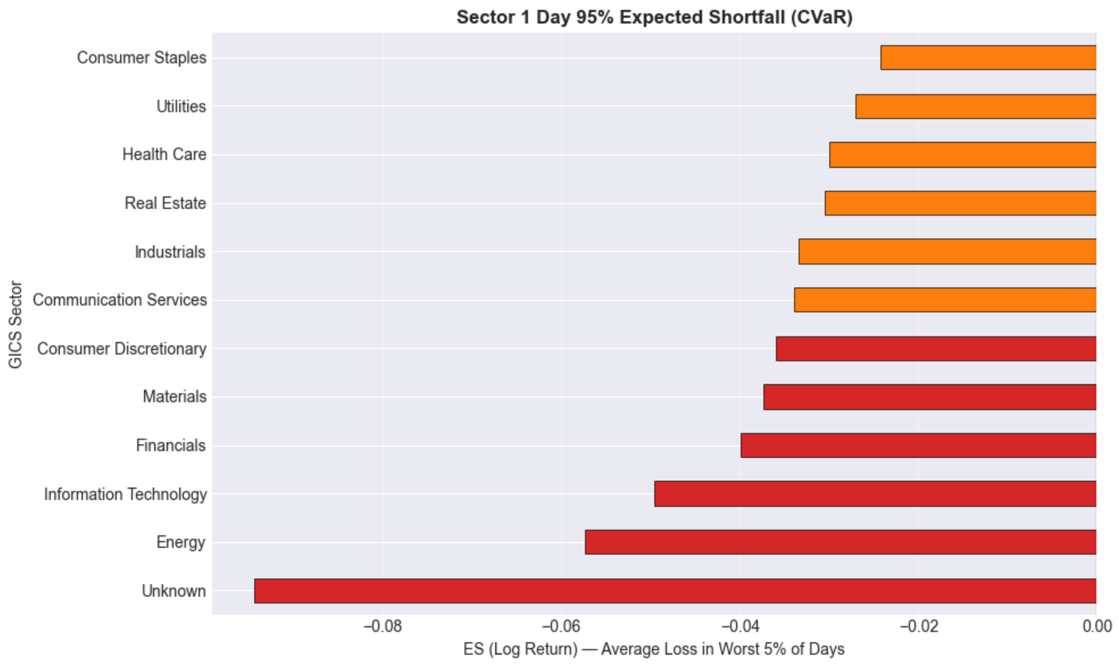
We map tickers to GICS sectors using an external reference file, compute equal weight sector returns, estimate historical VaR and ES, and backtest by checking breach ratios.

[48]:
```
df_long_sect = df_long.copy()

# map tickers to sectors; fall back to Unknown if mapping unavailable
try:
    sector_url = "https://raw.githubusercontent.com/datasets/
↪s-and-p-500-companies/master/data/constituents.csv"
    cons = pd.read_csv(sector_url)
    cons.columns = cons.columns.str.strip().str.lower()
    # detect ticker column
    ticker_col = None
    for c in ["symbol", "ticker"]:
        if c in cons.columns:
            ticker_col = c
            break
    # detect sector column (try multiple common names)
    sector_col = None
    for c in ["sector", "gics sector", "gics_sector", "industry"]:
        if c in cons.columns:
            sector_col = c
            break
    if ticker_col and sector_col:
        cons = cons.rename(columns = {ticker_col: "ticker", sector_col:␣
↪"sector"})
        cons["ticker"] = cons["ticker"].str.upper()
        sector_map = cons.set_index("ticker")["sector"]
        df_long_sect["sector"] = df_long_sect["ticker"].str.upper().
↪map(sector_map)
        missing = df_long_sect["sector"].isna().sum()
        df_long_sect["sector"] = df_long_sect["sector"].fillna("Unknown")
        msg = f"Loaded sector info from GitHub (cols: {ticker_col},␣
↪{sector_col}); {missing} rows mapped to 'Unknown'."
        print(msg)
    else:
        raise ValueError(f"Could not find ticker/sector columns. Available:␣
↪{list(cons.columns)}")
except Exception as e:
    print("Could not load sector info, using 'Unknown' for all. Error:", e)
    df_long_sect["sector"] = "Unknown"

df_long_sect = df_long_sect.dropna(subset = ["log_return"])

sector_returns = (
    df_long_sect.groupby(["date", "sector"])["log_return"]
    .mean()
    .unstack("sector")
    .sort_index()
)

print("Sector returns shape:", sector_returns.shape)
print("Sectors:", sector_returns.columns.tolist())

alpha = 0.95

def compute_var_es(series, alpha=0.95):
    """
    Compute Value-at-Risk (VaR) and Expected Shortfall (ES) for a return series.

    Convention: For log returns, negative values = losses.
    VaR at 95% confidence: 5th percentile (worst 5% of returns).
    ES: Average return conditional on exceeding VaR (mean of tail losses).

    Returns:
        q: VaR (negative value = loss threshold)
        es: ES (average loss in worst 5% of days)
    """
    series = series.dropna()
    # 5th percentile for 95% VaR
    q = series.quantile(1 - alpha)
    # ES: mean of returns in the tail
    es = series[series <= q].mean()
    return q, es

var_sector = {}
es_sector = {}

for sector in sector_returns.columns:
    q, es = compute_var_es(sector_returns[sector], alpha = alpha)
    var_sector[sector] = q
    es_sector[sector] = es

var_sector = pd.Series(var_sector)
es_sector = pd.Series(es_sector)

print("Sector VaR (95%):")
print(var_sector.sort_values())
print("\nSector ES (95%):")
print(es_sector.sort_values())

# figure 9: sector Value at Risk (more negative VaR = higher tail risk)
plt.figure(figsize=(10, 6))
colors = ["#d62728" if v < var_sector.median() else "#2ca02c" for v in␣
 ↪var_sector.sort_values()]
var_sector.sort_values().plot(kind="barh", color=colors, edgecolor="black",␣
 ↪linewidth=0.5)
plt.title("Sector 1 Day 95% Value at Risk (VaR)", fontsize=12,␣
 ↪fontweight="bold")
plt.xlabel("VaR (Log Return) - More Negative = Higher Risk")
plt.ylabel("GICS Sector")
plt.axvline(x=0, color="black", linewidth=0.8)
plt.grid(True, axis="x", alpha=0.3)
plt.tight_layout()
plt.show()

# figure 10: sector Expected Shortfall (ES captures average loss in worst 5% of␣
 ↪days)
plt.figure(figsize=(10, 6))
colors = ["#d62728" if v < es_sector.median() else "#ff7f0e" for v in es_sector.
 ↪sort_values()]
es_sector.sort_values().plot(kind="barh", color=colors, edgecolor="black",␣
 ↪linewidth=0.5)
plt.title("Sector 1 Day 95% Expected Shortfall (CVaR)", fontsize=12,␣
 ↪fontweight="bold")
plt.xlabel("ES (Log Return) - Average Loss in Worst 5% of Days")
plt.ylabel("GICS Sector")
```

```python
plt.axvline(x=0, color="black", linewidth=0.8)
plt.grid(True, axis="x", alpha=0.3)
plt.tight_layout()
plt.show()

# VaR backtest: count days where return < VaR (breach ratio should be ~5%)
breach_ratios = (sector_returns.lt(var_sector)).sum() / sector_returns.notna().
 ↪sum()
print("\nVaR breach ratios (should be ~5% for a perfect 95% VaR):")
print(breach_ratios.sort_values())
```

```
Loaded sector info from GitHub (cols: symbol, gics sector); 484 rows mapped to
'Unknown'.
Sector returns shape: (121, 12)
Sectors: ['Communication Services', 'Consumer Discretionary', 'Consumer
Staples', 'Energy', 'Financials', 'Health Care', 'Industrials', 'Information
Technology', 'Materials', 'Real Estate', 'Unknown', 'Utilities']
Sector VaR (95%):
Unknown                   -0.065108
Information Technology    -0.032883
Energy                    -0.029714
Consumer Discretionary    -0.024959
Financials                -0.024437
Real Estate               -0.022470
Communication Services    -0.022044
Health Care               -0.020824
Industrials               -0.020579
Utilities                 -0.019883
Materials                 -0.019027
Consumer Staples          -0.019010
dtype: float64

Sector ES (95%):
Unknown                   -0.094387
Energy                    -0.057422
Information Technology    -0.049587
Financials                -0.039918
Materials                 -0.037403
Consumer Discretionary    -0.035933
Communication Services    -0.033955
Industrials               -0.033373
Real Estate               -0.030468
Health Care               -0.029960
Utilities                 -0.027019
Consumer Staples          -0.024212
dtype: float64
```





```
VaR breach ratios (should be ~5% for a perfect 95% VaR):
sector
Consumer Discretionary    0.049587
```

```
Materials                 0.049587
Real Estate               0.049587
Communication Services    0.057851
Consumer Staples          0.057851
Energy                    0.057851
Financials                0.057851
Health Care               0.057851
Industrials               0.057851
Information Technology    0.057851
Unknown                   0.057851
Utilities                 0.057851
dtype: float64
```

1.7 7. Portfolio Strategies and Performance

We construct and compare three portfolio strategies:

Strategy	Description
Equal Weight (EW)	$w_i = 1/N$ for all assets
Inverse Volatility (IV)	$w_i \propto 1/\sigma_i$, tilts toward lower volatility stocks
Minimum Variance (MV)	$w = \Sigma^{-1}\mathbf{1}/(\mathbf{1}'\Sigma^{-1}\mathbf{1})$, uses Ledoit Wolf shrinkage covariance

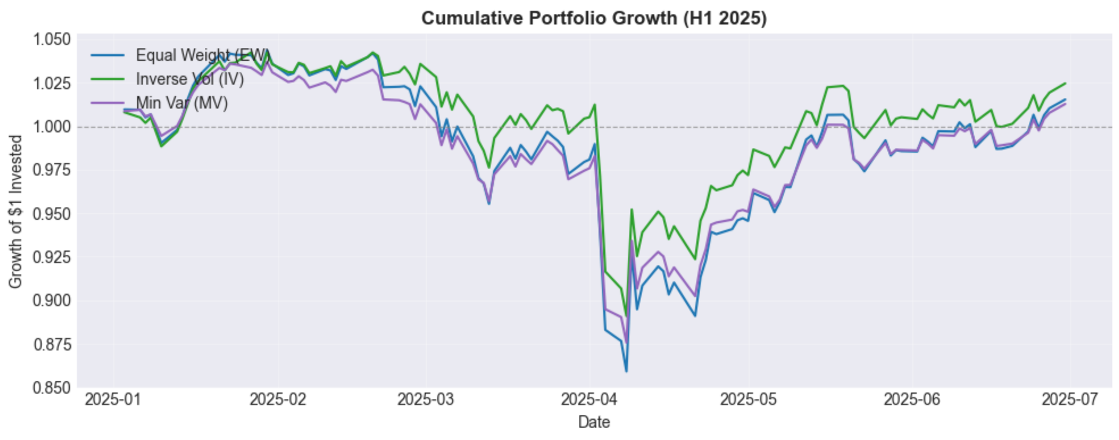
We evaluate annualized return, volatility, Sharpe ratio, and maximum drawdown. Finally, we split performance by volatility regime to see whether regime aware analysis provides additional insight.

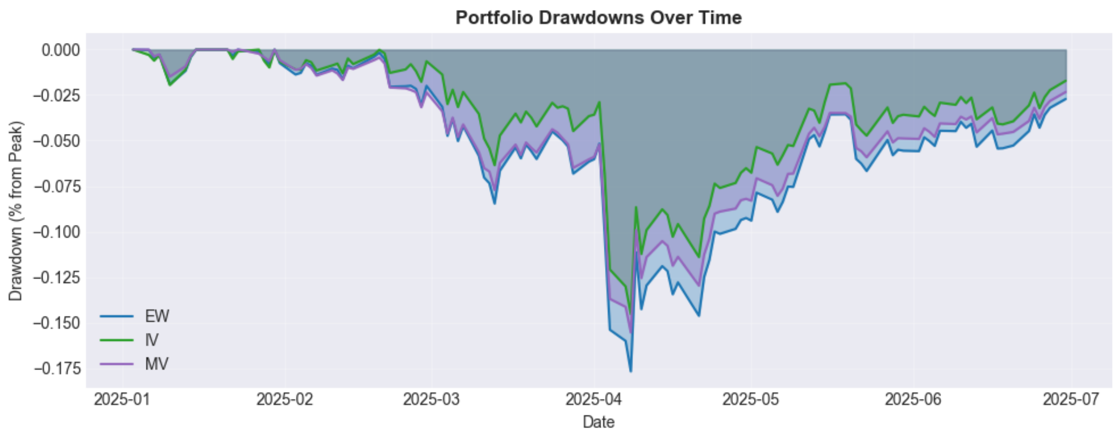
[49]:
```
# wide returns matrix; drop days with missing values for consistent asset␣
↪universe
returns_wide = (
    df_long.pivot(index="date", columns="ticker", values="log_return")
    .sort_index()
)
returns_wide = returns_wide.dropna(axis=0, how="any")

print("Returns matrix shape:", returns_wide.shape)

tickers_clean = returns_wide.columns
n_assets = len(tickers_clean)

# portfolio 1: equal weight (EW)
w_eq = np.repeat(1.0 / n_assets, n_assets)

# portfolio 2: inverse volatility (IV)
vols = returns_wide.std()
w_inv = 1.0 / vols
w_inv = w_inv / w_inv.sum()  # Normalize to sum to 1
```

```python
# portfolio 3: minimum variance (MV) with Ledoit Wolf shrinkage covariance
lw = LedoitWolf().fit(returns_wide.values)
Sigma = lw.covariance_

# verify covariance matrix is positive definite
eigenvals = np.linalg.eigvals(Sigma)
if np.any(eigenvals <= 0):
    print(f"Warning: Covariance matrix not positive definite. Min eigenvalue:␣
  ↪{eigenvals.min():.2e}")
    Sigma = Sigma + 1e-6 * np.eye(n_assets)

# minimum variance weights: w = Σ ¹·1 / (1'·Σ ¹·1)
ones = np.ones(n_assets)
inv_Sigma = np.linalg.inv(Sigma)
w_mv = inv_Sigma.dot(ones)
w_mv = np.clip(w_mv, 0, None)  # long only
w_mv = w_mv / w_mv.sum()

# verify weights sum to 1
print("Equal weight weights sum:", w_eq.sum())
print("Inverse vol weights sum:", w_inv.sum())
print("Min var weights sum:", w_mv.sum())
assert np.allclose([w_eq.sum(), w_inv.sum(), w_mv.sum()], 1.0), "Weights must␣
  ↪sum to 1"

# compute portfolio returns: r_portfolio = Σ w_i · r_i
port_eq = pd.Series(
    returns_wide.values.dot(w_eq),
    index=returns_wide.index,
    name="EW"
)
port_inv = pd.Series(
    returns_wide.values.dot(w_inv.values),
    index=returns_wide.index,
    name="IV"
)
port_mv = pd.Series(
    returns_wide.values.dot(w_mv),
    index=returns_wide.index,
    name="MV"
)


def perf_stats(r):
    r = r.dropna()
    ann_ret = r.mean() * 252
    ann_vol = r.std() * np.sqrt(252)
    sharpe = ann_ret / ann_vol if ann_vol > 0 else np.nan
    cum = (1 + r).cumprod()
    peak = cum.cummax()
    dd = cum / peak - 1
    max_dd = dd.min()
    return {
        "ann_return": ann_ret,
        "ann_vol": ann_vol,
        "sharpe": sharpe,
        "max_drawdown": max_dd
    }

# performance summary
for series in [port_eq, port_inv, port_mv]:
    print("\n=== Portfolio:", series.name, "===")
    stats = perf_stats(series)
    for k, v in stats.items():
        print(f"{k:>15}: {v:.4f}")

# cumulative growth
cum_eq = (1 + port_eq).cumprod()
cum_inv = (1 + port_inv).cumprod()
cum_mv = (1 + port_mv).cumprod()

# figure 11: cumulative portfolio growth
plt.figure(figsize=(10, 4))
plt.plot(cum_eq.index, cum_eq, label="Equal Weight (EW)", linewidth=1.5,␣
 ↪color="#1f77b4")
plt.plot(cum_inv.index, cum_inv, label="Inverse Vol (IV)", linewidth=1.5,␣
 ↪color="#2ca02c")
plt.plot(cum_mv.index, cum_mv, label="Min Var (MV)", linewidth=1.5,␣
 ↪color="#9467bd")
plt.axhline(y=1.0, color="gray", linestyle="--", linewidth=0.8, alpha=0.7)
plt.title("Cumulative Portfolio Growth (H1 2025)", fontsize=12,␣
 ↪fontweight="bold")
plt.xlabel("Date")
plt.ylabel("Growth of $1 Invested")
plt.legend(loc="upper left")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()


def drawdown_series(r):
    r = r.dropna()
    cum = (1 + r).cumprod()
    peak = cum.cummax()
    dd = cum / peak - 1
    return dd


dd_eq = drawdown_series(port_eq)
dd_inv = drawdown_series(port_inv)
dd_mv = drawdown_series(port_mv)

# figure 12: portfolio drawdowns
plt.figure(figsize=(10, 4))
plt.fill_between(dd_eq.index, dd_eq, 0, alpha=0.3, color="#1f77b4")
plt.fill_between(dd_inv.index, dd_inv, 0, alpha=0.3, color="#2ca02c")
plt.fill_between(dd_mv.index, dd_mv, 0, alpha=0.3, color="#9467bd")
plt.plot(dd_eq.index, dd_eq, label="EW", linewidth=1.5, color="#1f77b4")
plt.plot(dd_inv.index, dd_inv, label="IV", linewidth=1.5, color="#2ca02c")
plt.plot(dd_mv.index, dd_mv, label="MV", linewidth=1.5, color="#9467bd")
plt.title("Portfolio Drawdowns Over Time", fontsize=12, fontweight="bold")
plt.xlabel("Date")
plt.ylabel("Drawdown (% from Peak)")
plt.legend(loc="lower left")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# performance by volatility regime
regime_aligned = regime_series.reindex(port_eq.index).ffill().bfill()

for lbl, name in [(0, "Low vol regime"), (1, "High vol regime")]:
    mask = regime_aligned == lbl
    print(f"\n===== {name} =====")
    for series in [port_eq, port_inv, port_mv]:
        stats = perf_stats(series[mask])
        print(
            f"Portfolio {series.name}: "
            f"ann_return={stats['ann_return']:.4f}, "
            f"ann_vol={stats['ann_vol']:.4f}, "
            f"sharpe={stats['sharpe']:.4f}, "
            f"max_dd={stats['max_drawdown']:.4f}"
        )

# figure 13: portfolio growth with regime overlay
plt.figure(figsize=(10, 4))
plt.plot(cum_eq.index, cum_eq, label="EW", linewidth=1.5, color="#1f77b4")
plt.plot(cum_inv.index, cum_inv, label="IV", linewidth=1.5, color="#2ca02c")
plt.plot(cum_mv.index, cum_mv, label="MV", linewidth=1.5, color="#9467bd")

high_mask_port = regime_aligned == 1
```

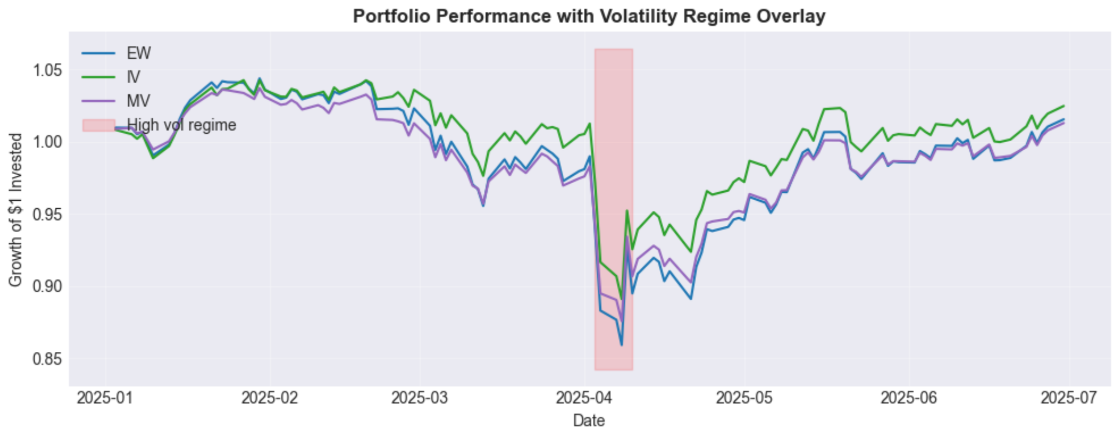
```
plt.fill_between(
    cum_eq.index,
    min(cum_eq.min(), cum_inv.min(), cum_mv.min()) * 0.98,
    max(cum_eq.max(), cum_inv.max(), cum_mv.max()) * 1.02,
    where=high_mask_port,
    color="red",
    alpha=0.15,
    label="High vol regime"
)

plt.title("Portfolio Performance with Volatility Regime Overlay", fontsize=12,␣
  ↪fontweight="bold")
plt.xlabel("Date")
plt.ylabel("Growth of $1 Invested")
plt.legend(loc="upper left")
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

```
Returns matrix shape: (121, 503)
Equal weight weights sum: 1.0
Inverse vol weights sum: 1.0000000000000002
Min var weights sum: 1.0

=== Portfolio: EW ===
     ann_return: 0.0569
        ann_vol: 0.2256
         sharpe: 0.2523
   max_drawdown: -0.1768

=== Portfolio: IV ===
     ann_return: 0.0703
        ann_vol: 0.1995
         sharpe: 0.3523
   max_drawdown: -0.1453

=== Portfolio: MV ===
     ann_return: 0.0437
        ann_vol: 0.1871
         sharpe: 0.2338
   max_drawdown: -0.1555
```





```
===== Low vol regime =====
Portfolio EW: ann_return=0.2653, ann_vol=0.1480, sharpe=1.7926, max_dd=-0.0846
Portfolio IV: ann_return=0.2591, ann_vol=0.1336, sharpe=1.9389, max_dd=-0.0635
Portfolio MV: ann_return=0.2120, ann_vol=0.1222, sharpe=1.7353, max_dd=-0.0772

===== High vol regime =====
Portfolio EW: ann_return=-3.9374, ann_vol=0.8012, sharpe=-4.9142, max_dd=-0.0845
Portfolio IV: ann_return=-3.5490, ann_vol=0.6945, sharpe=-5.1098, max_dd=-0.0827
Portfolio MV: ann_return=-3.1817, ann_vol=0.6692, sharpe=-4.7546, max_dd=-0.0700
```



1.8 8. Discussion and Insights

1.8.1 Key Findings

1. **Non-normality:** The QQ plot shows clear fat tails, and the squared return ACF confirms volatility clustering. This supports using GARCH and Markov switching models.

2. **GARCH captures persistence:** The fitted GARCH(1,1) has $\alpha + \beta \approx 0.89$, indicating persistent volatility. Conditional volatility spikes align with the high volatility regime from the Markov model.

3. **Regime switching is informative:** The two state model cleanly separates calm and turbulent periods. Regime probabilities could support dynamic risk management, for example reducing exposure when $P(\text{high vol}) > 0.5$.

4. **Sector tail risk varies:** VaR and ES estimates differ across sectors. Information Technology and Energy show higher tail risk than Consumer Staples and Utilities.

5. **Portfolio performance:** Inverse volatility weighting delivers the highest Sharpe ratio and shallowest drawdowns. Minimum variance is the most conservative. Regime conditioned analysis shows all strategies struggle during high volatility periods.

1.8.2 Limitations

- **Short sample:** Only 6 months of data (121 trading days) limits statistical power
- **Equal weight proxy:** Our pseudo index ignores market cap weighting
- **No transaction costs:** Real implementation would incur rebalancing costs
- **Sector coverage:** Some tickers could not be mapped to sectors
- **In sample only:** We did not perform out of sample backtesting

1.8.3 References

- Kaggle dataset: S&P 500 Stocks Trade Data for First 6 Months of 2025
- Sector mappings: GitHub `datasets/s-and-p-500-companies`
- GARCH modeling: Bollerslev (1986), `arch` Python package

- Markov switching: Hamilton (1989), `statsmodels` implementation

[50]:
```
import os
os.makedirs("figures", exist_ok=True)

# diagnostics plot
fig, axes = plt.subplots(2, 2, figsize=(10, 8))

ax = axes[0, 0]
ax.hist(r, bins=40, density=True, alpha=0.6)
x = np.linspace(mu - 4 * sigma, mu + 4 * sigma, 200)
ax.plot(x, norm.pdf(x, mu, sigma))
ax.set_title("Return histogram vs Normal")

sm.qqplot(r, line="s", ax=axes[0, 1])
axes[0, 1].set_title("QQ-plot")

plot_acf(r, lags=20, ax=axes[1, 0])
axes[1, 0].set_title("ACF of returns")

plot_acf(r ** 2, lags=20, ax=axes[1, 1])
axes[1, 1].set_title("ACF of squared returns")

plt.tight_layout()
fig.savefig("figures/returns_diagnostics.pdf", bbox_inches="tight")
plt.close(fig)

# garch plot
fig, ax = plt.subplots(figsize=(10, 4))
ax.plot(cond_vol.index, cond_vol.values)
ax.set_title("GARCH(1,1) Conditional Volatility Over Time")
ax.set_xlabel("Date")
ax.set_ylabel("Conditional volatility")
fig.savefig("figures/garch_volatility.pdf", bbox_inches="tight")
plt.close(fig)

# regime overlay plot
fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

ax = axes[0]
ax.plot(index_level.index, index_level.values, label="Index proxy")
high_mask = regime_series == 1
ax.fill_between(
    index_level.index,
    index_level.min(),
    index_level.max(),
    where=high_mask,
    alpha=0.15
)
ax.set_title("Index with high-vol regime shading")
ax.legend()

ax2 = axes[1]
ax2.plot(prob_high.index, prob_high.values)
ax2.set_title("Smoothed probability of high-vol regime")
ax2.set_xlabel("Date")
ax2.set_ylabel("Probability")

plt.tight_layout()
fig.savefig("figures/regime_probabilities.pdf", bbox_inches="tight")
plt.close(fig)

# sector var/es plot
fig, axes = plt.subplots(2, 1, figsize=(8, 8))

var_sector.sort_values().plot(kind="barh", ax=axes[0])
axes[0].set_title("Sector 95% VaR")

es_sector.sort_values().plot(kind="barh", ax=axes[1])
axes[1].set_title("Sector 95% ES")
axes[1].set_xlabel("Log return")

plt.tight_layout()
fig.savefig("figures/sector_var_es.pdf", bbox_inches="tight")
plt.close(fig)

# portfolio performance plot
fig, axes = plt.subplots(2, 1, figsize=(10, 6), sharex=True)

ax = axes[0]
ax.plot(cum_eq.index, cum_eq, label="EW")
ax.plot(cum_inv.index, cum_inv, label="IV")
ax.plot(cum_mv.index, cum_mv, label="MV")
high_mask_port = regime_aligned == 1
ax.fill_between(
    cum_eq.index,
    min(cum_eq.min(), cum_inv.min(), cum_mv.min()) * 0.98,
    max(cum_eq.max(), cum_inv.max(), cum_mv.max()) * 1.02,
    where=high_mask_port,
    alpha=0.15
)
ax.set_title("Portfolio performance with high-vol shading")
ax.legend()
```

```
ax2 = axes[1]
ax2.plot(dd_eq.index, dd_eq, label="EW")
ax2.plot(dd_inv.index, dd_inv, label="IV")
ax2.plot(dd_mv.index, dd_mv, label="MV")
ax2.set_title("Portfolio drawdowns")
ax2.set_xlabel("Date")
ax2.set_ylabel("Drawdown")

plt.tight_layout()
fig.savefig("figures/portfolio_performance.pdf", bbox_inches="tight")
plt.close(fig)
```

[51]:
```
rets_eq = cum_eq.pct_change().fillna(0)
rets_inv = cum_inv.pct_change().fillna(0)
rets_mv = cum_mv.pct_change().fillna(0)

import pandas as pd
import numpy as np

def portfolio_stats(r):
    ann_ret = (1 + r).prod()**(252 / len(r)) - 1
    ann_vol = r.std() * np.sqrt(252)
    sharpe = ann_ret / ann_vol
    cum = (1 + r).cumprod()
    peak = cum.cummax()
    dd = cum / peak - 1
    max_dd = dd.min()
    return ann_ret, ann_vol, sharpe, max_dd

stats = {}
for name, r in [("EW", rets_eq), ("IV", rets_inv), ("MV", rets_mv)]:
    stats[name] = portfolio_stats(r)

metrics = pd.DataFrame(
    stats,
    index=["ann_ret", "ann_vol", "sharpe", "max_dd"]
).T

# Pretty print as percentages with 1 decimal place
display(metrics.apply(
    lambda col: col if col.name == "sharpe" else 100 * col
))
```

```
     ann_ret    ann_vol    sharpe     max_dd
EW  1.187874  22.517831  0.052753 -17.681064
IV  3.454386  19.919113  0.173421 -14.533446
MV  0.860698  18.671438  0.046097 -15.551305
```