Barnstable and Long-Run Risk

HBS Case

The Risk of Stocks in the Long-Run: The Barnstable College Endowment

---

2. Estimating Underperformance

Data

Use the returns on the S&P 500 (r^m) and 1-month T-bills, (r^f) provided in `barnstable_analysis_data.xlsx`.

- Data goes through `END_YR=2024`.

Barnstable's estimates of mean and volatility are based on the subsample of 1965 to 1999.

- We consider this subsample, as well as 2000-{END_YR}, as well as the full sample of 1926-{END_YR}.

Notation

- r = level return rates
- R = cumulative return factor
- $\mathbf{r}$ = log return rates

$$R \equiv 1 + r$$

$$\mathbf{r} \equiv \ln(1 + r) = \ln(R)$$

In [1]:
```
import polars as pl
import polars.selectors as cs
import math
from datetime import datetime
import calendar

FREQ = 12
END_YR = 2024

xlsx_file = "../data/barnstable_analysis_data.xlsx"
```

```python
raw = pl.read_excel(xlsx_file, sheet_name="data")
raw.tail()
```

Out[1]: shape: (5, 3)

date	SPX	TB1M
date	f64	f64
2024-08-30	0.024283	0.00438
2024-09-30	0.022821	0.003826
2024-10-31	-0.00869	0.003752
2024-11-29	0.06042	0.003475
2024-12-31	-0.023445	0.003337

In [2]:
```python
# add excess return
df = raw.with_columns(pl.col("SPX").sub(pl.col("TB1M")).alias("excess"))
```

In [3]:
```python
def parse_date(date_str: str, is_end: bool = False) -> datetime:
    """Parse date string with flexible format, defaulting to first day."""
    if date_str is None:
        return None

    # Try different formats
    for fmt in ["%Y-%m-%d", "%Y-%m", "%Y"]:
        try:
            dt = datetime.strptime(date_str, fmt)
            if fmt == "%Y-%m" and is_end:
                last_day = calendar.monthrange(dt.year, dt.month)[1]
                return dt.replace(day=last_day)
            elif fmt == "%Y" and is_end:
                return dt.replace(month=12, day=31)
            return dt
        except ValueError:
            continue

    raise ValueError(f"Date string '{date_str}' doesn't match any supported
```

In [4]:
```python
def get_period(
        data: pl.DataFrame,
        start_str: str = None,
        end_str: str = None
    ) -> pl.DataFrame:
    """
    Args:
        - data: pl.DataFrame
        - start_str: Date string in format %Y-%m-%d, %Y-%m, or %Y
        - end_str: Date string in format %Y-%m-%d, %Y-%m, or %Y
    """
    start_day = parse_date(start_str)
    end_day = parse_date(end_str, is_end=True)

    if start_day is None and end_day is None:
```

```python
        tb = data
    elif start_day and end_day:
        tb = data.filter(pl.col("date").is_between(start_day, end_day))
    elif start_day:
        tb = data.filter(pl.col("date").ge(start_day))
    elif end_day:
        tb = data.filter(pl.col("date").le(end_day))

    return tb

def log_series(series: pl.Series) -> pl.Series:
    return (series + 1).log(base=math.exp(1))

def log_dataframe(df: pl.DataFrame) -> pl.DataFrame:
    return df.with_columns([
        log_series(pl.col(col)).alias(col)
        for col in df.columns if df.schema[col] == pl.Float64
    ])
```

In [5]:
```python
def get_stat(
        data: pl.DataFrame,
        start_str: str = None,
        end_str: str = None
    ) -> pl.DataFrame:
    result = (
        get_period(data, start_str, end_str)
        .drop("date")
        .describe()
        .filter(
            pl.col("statistic").is_in(["mean", "std"])
        )
        .with_columns(
            pl.when(pl.col("statistic") == "mean")
            .then(cs.float() * FREQ)
            .when(pl.col("statistic") == "std")
            .then(cs.float() * math.sqrt(FREQ))  # Annualize std differently
            .otherwise(cs.float())
            .name.keep()
        )
    )

    return result
```

1. Summary Statistics

Report the following (annualized) statistics.

| | 1965-1999 | | | 2000-{END_YR} | | | 1926-{END_YR} | | | |---|---|---|---|---|---|---|---|---|---|---| | | | mean | vol | mean | vol | | mean | vol | | **levels** | r^m |0.129354 |0.149405 |0.087542 |0.152815 | |0.115529 |0.18665 | | | $\tilde{r}^m$ |0.06866 |0.150227 |0.070091 |0.153093 | |0.083308 |0.187329 | | | r^f |0.061503 |0.007179 |0.017451 |0.005553 | |0.031928 |0.008507 | | **logs** | $\mathbf{r}^m$ |0.1176 |0.149568 |0.075553 |0.153763 | |0.097821

|0.185938 | | | $\tilde{\mathbf{r}}^m$ |0.057161 |0.151207 |0.058143 |0.154227 | |0.065673 |0.186914 | | | $\mathbf{r}_f$ |0.06132 |0.007132 |0.017423 |0.005541 | |0.03185 |0.008473 |

- Comment on how the full-sample return stats compare to the sub-sample stats.
- Comment on how the level stats compare to the log stats.

In [6]:
```
log_df = log_dataframe(df)
log_df.tail()
```

Out[6]: shape: (5, 4)

date	SPX	TB1M	excess
date	f64	f64	f64
2024-08-30	0.023993	0.004371	0.019707
2024-09-30	0.022564	0.003819	0.018817
2024-10-31	-0.008728	0.003745	-0.01252
2024-11-29	0.058665	0.003469	0.055383
2024-12-31	-0.023724	0.003332	-0.027147

In [7]:
```
# levels
get_stat(df, "1965", "1999")
```

Out[7]: shape: (2, 4)

statistic	SPX	TB1M	excess
str	f64	f64	f64
"mean"	0.129354	0.061503	0.06866
"std"	0.149405	0.007179	0.150227

In [8]:
```
get_stat(df, "2000")
```

Out[8]: shape: (2, 4)

statistic	SPX	TB1M	excess
str	f64	f64	f64
"mean"	0.087542	0.017451	0.070091
"std"	0.152815	0.005553	0.153093

In [9]:
```
get_stat(df, "1926")
```

Out[9]: shape: (2, 4)

statistic	SPX	TB1M	excess
str	f64	f64	f64
"mean"	0.115529	0.031928	0.083308
"std"	0.18665	0.008507	0.187329

In [10]:
```
# log
get_stat(log_df, "1965", "1999")
```

Out[10]: shape: (2, 4)

statistic	SPX	TB1M	excess
str	f64	f64	f64
"mean"	0.1176	0.06132	0.057161
"std"	0.149568	0.007132	0.151207

In [11]:
```
get_stat(log_df, "2000")
```

Out[11]: shape: (2, 4)

statistic	SPX	TB1M	excess
str	f64	f64	f64
"mean"	0.075553	0.017423	0.058143
"std"	0.153763	0.005541	0.154227

In [12]:
```
get_stat(log_df, "1926")
```

Out[12]: shape: (2, 4)

statistic	SPX	TB1M	excess
str	f64	f64	f64
"mean"	0.097821	0.03185	0.065673
"std"	0.185938	0.008473	0.186914

2. Probability of Underperformance

Recall the following:

- If $x \sim \mathcal{N}\left(\mu_x, \sigma_x^2\right)$, then

$$\Pr\left[x < \ell\right] = \Phi_{\mathcal{N}}\left(L\right)$$

where $L = \frac{\ell - \mu_x}{\sigma_x}$ and $\Phi_{\mathcal{N}}$ denotes the standard normal cdf.

- Remember that cumulative log returns are simply the sum of the single-period log returns:

$$\mathbf{r}^m_{t,t+h} \equiv \sum_{i=1}^{h} \mathbf{r}^m_{t+i}$$

- It will be convenient to use and denote sample averages. We use the following notation for an h-period average ending at time $t+h$:

$$\bar{\mathbf{r}}^m_{t,t+h} = \frac{1}{h}\sum_{i=1}^{h} \mathbf{r}^m_{t+i}$$

Calculate the probability that the cumulative market return will fall short of the cumulative risk-free return:

$$\Pr\left[R^m_{t,t+h} < R^f_{t,t+h}\right]$$

To analyze this analytically, convert the probability statement above to a probability statement about mean log returns.

$$\Pr\left[R^m_{t,t+h} < R^f_{t,t+h}\right] = \Pr\left[\exp(\mathbf{r}^m_{t,t+h}) < \exp(\mathbf{r}^f_{t,t+h})\right] = \Pr\left[\mathbf{r}^m_{t,t+h} < \mathbf{r}^f_{t,t+h}\right] = \Pr\left[\bar{\mathbf{r}}^m_{t}\right.$$

Log returns are approximately normally distributed (by CLT for sums) Level returns (products) are log-normally distributed, which doesn't have the nice properties needed for the $\Phi_{\mathcal{N}}$ formula

2.1

Calculate the probability using the subsample 1965-1999.

In [13]:
```
from scipy.stats import norm
```

In [ ]:
```
def prob_underperform(data, h, os=False, os_data=None):
    """
    Arg:
        - data: log return dataframe
        - h: period
        - os: whether it's out sample prediction
        - os_data: out sample data
    """
    # in-sample
    excess_mean = data["excess"].mean() * FREQ
    excess_std = data["excess"].std() * math.sqrt(FREQ)

    # Calculate standardized value: L = (0 - mu_e) / sigma_e
    if not os:
```

```
        L = -excess_mean / excess_std
    else:
        os_mean = os_data["excess"].mean() * FREQ
        L = (os_mean - excess_mean) / excess_std
    prob = norm.cdf(math.sqrt(h) * L)

    return prob
```

In [54]:
```
sub_1965_1999 = get_period(log_df, "1965", "1999")
prob_underperform(sub_1965_1999, 35)
```

Out[54]: np.float64(0.01266043954250752)

2.2

Report the precise probability for $h = 15$ and $h = 30$ years.

In [55]:
```
prob_underperform(sub_1965_1999, 15)
```

Out[55]: np.float64(0.07158133198503584)

In [56]:
```
prob_underperform(sub_1965_1999, 30)
```

Out[56]: np.float64(0.019199471302532578)

2.3

Plot the probability as a function of the investment horizon, h, for $0 < h \leq 30$ years.

Hint: The probability can be expressed as:

$$p(h) = \Phi_{\mathcal{N}}\left(-\sqrt{h}\,\mathrm{SR}\right)$$

where SR denotes the sample Sharpe ratio of **log** market returns.

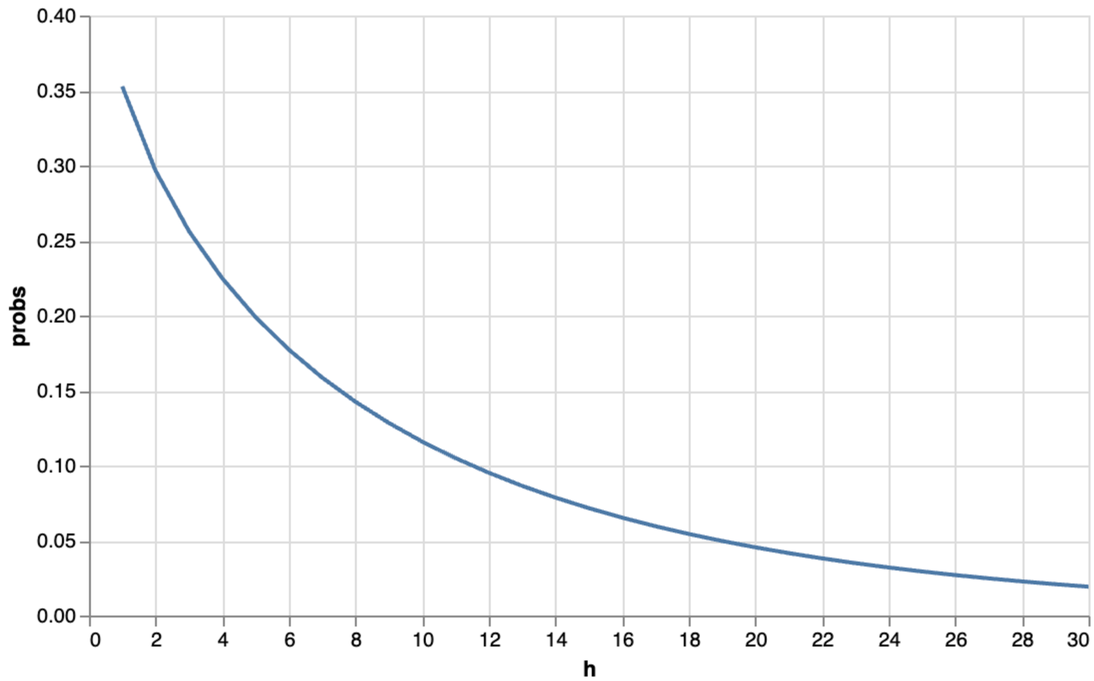
In [59]:
```
def h_years_probs(data, h, os=False, os_data=None):
    probs = []
    for i in range(1, h+1):
        probs.append(prob_underperform(data, i, os=os, os_data=os_data))

    result = pl.DataFrame({
        "h": pl.arange(1, h+1, eager=True),
        "probs": probs
    })
    return result
```

In [60]:
```
def plot_line(df: pl.DataFrame, x, y):
    return df.plot.line(
        x=x, y=y
    ).properties(width=500)
```

In [63]:
```
probs_1965_1999 = h_years_probs(sub_1965_1999, 30)
plot_line(probs_1965_1999, x="h", y="probs")
```

Out[63]:

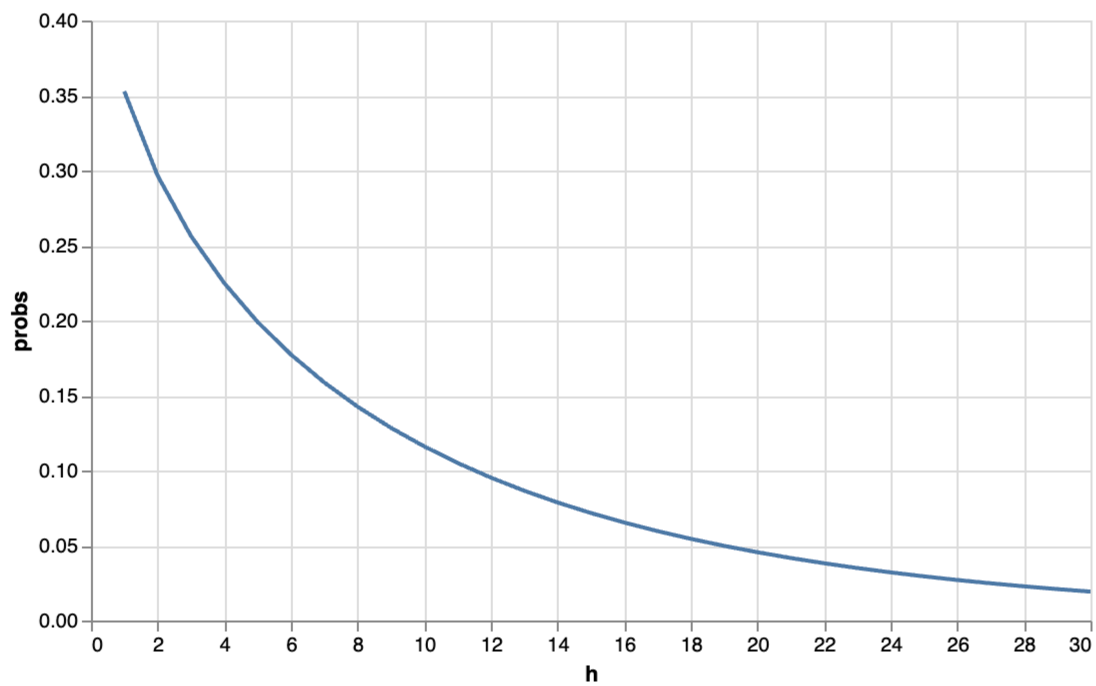


3. Full Sample Analysis

Use the sample 1965-{END_YR} to reconsider the 30-year probability. As of the end of {END_YR}, calculate the probability of the stock return underperforming the risk-free rate over the next 30 years. That is, $R^m_{t,t+h}$ underperforming $R^f_{t,t+h}$ for $0 < h \leq 30$.

In [64]:
```
full_1965_end = get_period(log_df, "1965")
probs_1965_end = h_years_probs(full_1965_end, 30)
plot_line(probs_1965_end, x="h", y="probs")
```

Out[64]:



4. In-Sample Estimate of Out-of-Sample Likelihood

Let's consider how things turned out relative to Barnstable's 1999 expectations.

What was the probability (based on the 1999 estimate of μ) that the `h` -year market return, $R^m_{t,t+h}$, would be smaller than that realized in `2000-{END_YR}` ?

Hint: You can calculate this as:

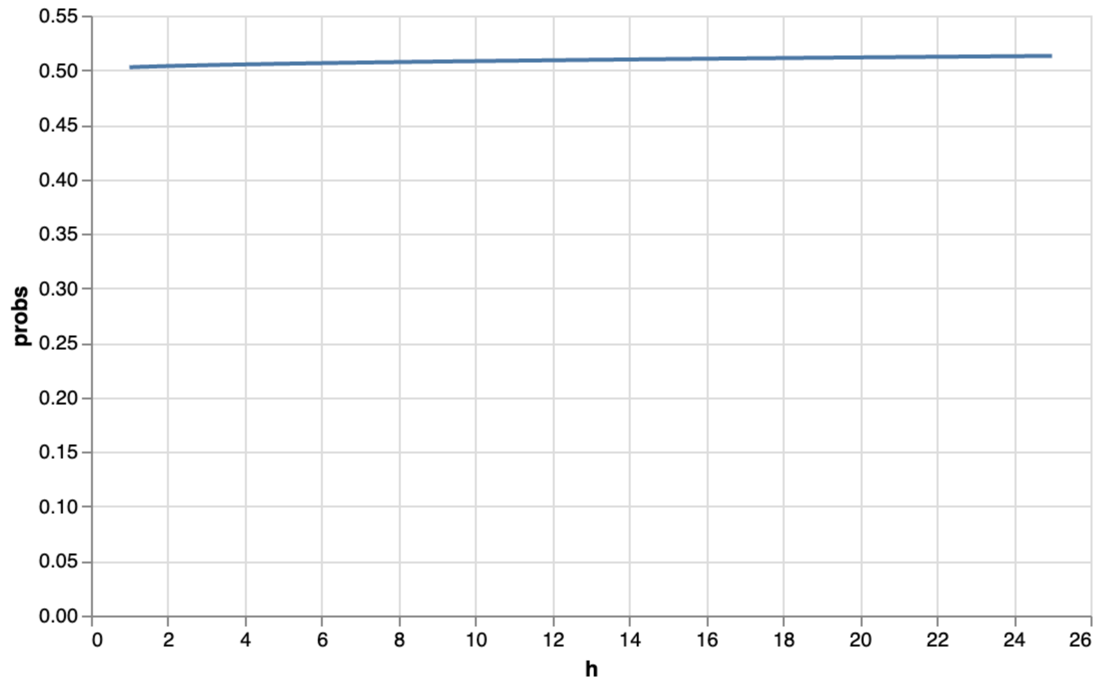
$$p = \Phi_{\mathcal{N}}\left(\sqrt{h}\,\frac{\bar{\mathbf{r}}_{\text{out-of-sample}} - \bar{\mathbf{r}}_{\text{in-sample}}}{\sigma_{\text{in-sample}}}\right)$$

where "in-sample" denotes 1965-1999 and "out-of-sample" denotes 2000-{END_YR}.

In [62]:

```
sub_2000_end = get_period(log_df, "2000")
probs_2000_end = h_years_probs(sub_1965_1999, 25, os=True, os_data=sub_2000_
plot_line(probs_2000_end, x="h", y="probs")
```

Out[62]:



In [74]:

```
probs_2000_end
```

Out[74]: shape: (25, 2)

h	probs
i64	f64
1	0.502591
2	0.503664
3	0.504487
4	0.505181
5	0.505793
...	...
21	0.51187
22	0.512149
23	0.512422
24	0.51269
25	0.512951

hw3_exercise

October 15, 2025

1 Exercise - VaR

1.1 Data

This problem uses `weekly` return data from `data/spx_returns_weekly.xlsx`.

Choose any 4 stocks to evaluate below.

For example, * `AAPL` * `META` * `NVDA` * `TSLA`

[1]:
```
import polars as pl
import polars.selectors as cs
import numpy as np
from scipy import stats
from openpyxl import load_workbook

xlsx_file = '../data/spx_returns_weekly.xlsx'
TICKERS = ["AAPL", "META", "NVDA", "TSLA"]
FREQ = 52
```

[2]:
```
wb = load_workbook(xlsx_file, read_only=True)
print(wb.sheetnames)
wb.close()
```

```
['s&p500 names', 'benchmark names', 's&p500 rets', 'benchmark rets']
```

[3]:
```
rets = pl.read_excel(xlsx_file, sheet_name='s&p500 rets')[TICKERS]
rets.head(3)
```

[3]: shape: (3, 4)

AAPL --- f64	META --- f64	NVDA --- f64	TSLA --- f64
0.024514	-0.009055	-0.009315	-0.057685
-0.053745	-0.032931	0.000836	-0.06576
0.06595	0.035255	0.037578	0.042575

2 Diversification

2.1 Unconditional Vol, VaR, cVaR

Using the full sample, calculate for each series the (unconditional) * volatility * empirical VaR (.05) * empirical CVaR (.05)

Recall that by **empirical** we refer to the direct quantile estimation. (For example, using `.quantile()` in pandas.

[4]:
```python
def calc_cvar(series: pl.Series) -> pl.Float64:
    return series.filter(series <= series.quantile(.05)).mean()
```

[5]:
```python
def get_risk_metrics(rets: pl.DataFrame|pl.Series) -> pl.DataFrame:
    if isinstance(rets, pl.Series):
        rets = rets.to_frame()

    vol = rets.std() * np.sqrt(FREQ)
    var = rets.quantile(.05)
    cvar = rets.select(pl.all().map_batches(calc_cvar, returns_scalar=True))
    result = pl.concat(
        [vol, var, cvar], how='vertical'
    ).insert_column(
        0, pl.Series("metric", ["vol", "var", "cvar"])
    )
    return result
```

[6]:
```python
ans1_1 = get_risk_metrics(rets)
print(ans1_1)
```

```
shape: (3, 5)

 metric  AAPL       META       NVDA       TSLA
 ---     ---        ---        ---        ---
 str     f64        f64        f64        f64

 vol     0.276629   0.351336   0.463283   0.586431
 var     -0.056501  -0.07004   -0.086876  -0.117432
 cvar    -0.083125  -0.103196  -0.116455  -0.147814
```

Note: The volatility output in the textbook is incorrect. It's the result of mistakenly using `FREQ=252` to annualize the volatility.

2.2 Equally-weighted portfolio

Form an equally-weighted portfolio of the investments.

Calculate the statistics of `1.1` for this portfolio, and compare the results to the individual return statistics. What do you find? What is driving this result?

[7]:
```python
eq_w_port = rets.mean_horizontal().alias("eq_w_rets")
ans1_2 = get_risk_metrics(eq_w_port)
print(ans1_2)
```

```
shape: (3, 2)

 metric   eq_w_rets
 ---      ---
 str      f64

 vol      0.315543
 var      -0.061982
 cvar     -0.084992
```

Compare with 1.1, we find that the absolute value of three risk metrics have decreased after forming the portfolio. AAPL is lowering the volatility, while other three securities are boosting it.

2.3 Drop most volatile asset

Re-calculate `1.2`, but this time drop your most volatile asset, and replace the portion it was getting with 0. (You could imagine we're replacing the most volatile asset with a negligibly small risk-free rate.)

In comparing the answer here to 1.2, how much risk is your most volatile asset adding to the portfolio? Is this in line with the amount of risk we measured in the stand-alone risk-assessment of `1.1`?

[8]:
```python
(
    get_risk_metrics(rets).unpivot(index="metric", variable_name="ticker",␣
 ↪value_name="value")
    .sort(pl.col("value").abs(), descending=True)
    .group_by("metric")
    .first()
)
```

[8]:
```
shape: (3, 3)

 metric   ticker   value
 ---      ---      ---
 str      str      f64

 var      TSLA     -0.117432
 cvar     TSLA     -0.147814
 vol      TSLA     0.586431
```

`TSLA` is the most volatile asset, drop it and replace with 0.

[9]:
```
drop_volatile = rets.drop("TSLA")
ans1_3 = get_risk_metrics(
    drop_volatile
    .mean_horizontal().alias("eq_w_rets")
)
print(ans1_3)
```

```
shape: (3, 2)

 metric   eq_w_rets
 ---      ---
 str      f64

 vol      0.291153
 var      -0.056641
 cvar     -0.080708
```

[10]:
```
# compare with 1.2
print(ans1_2)
```

```
shape: (3, 2)

 metric   eq_w_rets
 ---      ---
 str      f64

 vol      0.315543
 var      -0.061982
 cvar     -0.084992
```

[11]:
```
print(rets.corr())
```

```
shape: (4, 4)

 AAPL       META       NVDA       TSLA
 ---        ---        ---        ---
 f64        f64        f64        f64

 1.0        0.429349   0.492375   0.453314
 0.429349   1.0        0.426005   0.274745
 0.492375   0.426005   1.0        0.415838
 0.453314   0.274745   0.415838   1.0
```

3 Dynamic Measures

3.1 Conditional vol, VaR, cVaR

Let's measure the **conditional** statistics of the equally-weighted portfolio of `1.2`, as of the end of the sample.

Volatility For each security, calculate the **rolling** volatility series, σ_t, with a window of $m = 26$.

The value at σ_t in the notes denotes the estimate using data through time $t-1$, and thus (potentially) predicting the volatility at σ_t.

Mean Suppose we can approximate that the daily mean return is zero.

VaR Calculate the **normal VaR** and **normal CVaR** for $q = .05$ and $\tau = 1$ as of the end of the sample.Use the approximation, $\mathtt{z}_{.05} = -1.65$.

Notation Note In this setup, we are using a forecasted volatility, σ_t to estimate the VaR return we would have estimated at the end of $t-1$ in prediction of time t.

Conclude and Compare Report * volatility (annualized). * normal VaR (.05) * normal CVaR (.05)

How do these compare to the answers in `1.2`?

Setup: - Returns: $r_{\tau,t} \sim N(\mu_{\tau,t}, \sigma^2_{\tau,t})$ - Standardized: $z = \frac{r_{\tau,t}-\mu_{\tau,t}}{\sigma_{\tau,t}} \sim N(0,1)$ - Quantile: $P(z \leq z_q) = q$

Key Result for Truncated Normal:

$$E[z \mid z < z_q] = -\frac{\phi_z(z_q)}{\Phi_z(z_q)}$$

Proof:

$$E[z \mid z < z_q] = \frac{E[z\mathbf{1}_{\{z<z_q\}}]}{P(z < z_q)} = \frac{\int_{-\infty}^{z_q} z \cdot \phi_z(z)\, dz}{\Phi_z(z_q)}$$

Since $\phi_z(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$ and $\frac{d}{dz}\phi_z(z) = -z \cdot \phi_z(z)$:

$$\int_{-\infty}^{z_q} z \cdot \phi_z(z)\, dz = -[\phi_z(z)]_{-\infty}^{z_q} = -\phi_z(z_q)$$

Therefore: $E[z \mid z < z_q] = -\frac{\phi_z(z_q)}{\Phi_z(z_q)} = -\frac{\phi_z(z_q)}{q}$

CVaR Formula:

$$\begin{aligned} \text{CVaR}_{q,\tau} &= E[r_{\tau,t} \mid r_{\tau,t} \leq \text{VaR}_{q,\tau}] \\ &= \mu_{\tau,t} + \sigma_{\tau,t} \cdot E[z \mid z < z_q] \\ &= \mu_{\tau,t} - \frac{\phi_z(z_q)}{q}\sigma_{\tau,t} \end{aligned}$$

Compare with VaR: - VaR: $\mu_{\tau,t} + z_q \sigma_{\tau,t}$ - CVaR: $\mu_{\tau,t} - \frac{\phi_z(z_q)}{q}\sigma_{\tau,t}$

Both are linear in μ and σ (sufficient statistics for normal distribution)

Implementation: - z_q: `stats.norm.ppf(q)` - $\phi_z(z_q)$: `stats.norm.pdf(z_q)`

Volatility (annualized)

[12]:
```
# annualized rolling vol
def annual_rolling_vol(rets: pl.DataFrame|pl.Series) -> pl.DataFrame:
    if isinstance(rets, pl.Series):
        rets = rets.to_frame()

    res = rets.with_columns(
        pl.all().rolling_std(window_size=26)
        .mul(np.sqrt(FREQ)).name.suffix("_vol")
    ).drop_nulls()

    return res.select(cs.ends_with("_vol")).rename(lambda col: col[:-4])
```

[13]:
```
roll_vol = (
    pl.concat([
        annual_rolling_vol(rets)[-1, :],
        annual_rolling_vol(eq_w_port)[-1, :]
        .select(pl.all().alias("portfolio"))
    ], how="horizontal")
)
print(roll_vol)
```

```
shape: (1, 5)

 AAPL      META     NVDA      TSLA      portfolio
 ---       ---      ---       ---       ---
 f64       f64      f64       f64       f64

 0.371224  0.41831  0.598249  0.598597  0.38975
```

normal VaR (0.05) and normal cVaR (0.05)

[14]:
```
def calc_var_cvar(
    rets: pl.DataFrame|pl.Series,
    conditional: bool=False,
    type: str="var",
    q: pl.Float64=.05
) -> pl.DataFrame:
    """
    Args:
        - rets: rows are periodic returns, cols are tickers
        - conditional: assume normal dist or not
```

```
        - type: "var" or "cvar"
        - q: quantile
    Return:
        - a 1*(rets.width) shape DataFrame
    """
    z = stats.norm.ppf(q)
    vol_annual = annual_rolling_vol(rets)[-1, :]
    vol_daily = vol_annual / np.sqrt(252)
    if type == "var":
        if not conditional:
            var = rets.quantile(q)
            return var
        else:
            # assume daily mean is 0
            return 0 + vol_daily * z
    elif type == "cvar":
        if not conditional:
            cvar = rets.select(pl.all().map_batches(calc_cvar,␣
↪returns_scalar=True))
            return cvar
        else:
            return 0 + vol_daily * (-1) * stats.norm.pdf(z) / q
```

[15]:
```
# concat individual stock and portfolio return
total_rets = pl.concat(
    [rets, eq_w_port.to_frame()], how="horizontal"
).rename({"eq_w_rets": "portfolio"})
total_rets.head(3)
```

[15]: shape: (3, 5)

AAPL --- f64	META --- f64	NVDA --- f64	TSLA --- f64	portfolio --- f64
0.024514	-0.009055	-0.009315	-0.057685	-0.012885
-0.053745	-0.032931	0.000836	-0.06576	-0.0379
0.06595	0.035255	0.037578	0.042575	0.04534

[16]:
```
uncon_var = calc_var_cvar(total_rets, conditional=False, type="var")
con_var = calc_var_cvar(total_rets, conditional=True, type="var")
ucon_cvar = calc_var_cvar(total_rets, conditional=False, type="cvar")
con_cvar = calc_var_cvar(total_rets, conditional=True, type="cvar")
```

[17]:
```
summary = {
    "conditional": [False, True, False, True],
```

```
    "type": ["var", "var", "cvar", "cvar"]
}

# Add the value columns
tickers = total_rets.schema.names()
results = [uncon_var, con_var, ucon_cvar, con_cvar]
for ticker in tickers:
    summary[f"{ticker}"] = [result[ticker][0] for result in results]

ans2_1 = pl.DataFrame(summary)
print(ans2_1)
```

shape: (4, 7)

conditional	type	AAPL	META	NVDA	TSLA	portfolio
---	---	---	---	---	---	---
bool	str	f64	f64	f64	f64	f64
false	var	-0.056501	-0.07004	-0.086876	-0.117432	-0.061982
true	var	-0.038465	-0.043344	-0.061988	-0.062024	-0.040384
false	cvar	-0.083125	-0.103196	-0.116455	-0.147814	-0.084992
true	cvar	-0.048236	-0.054355	-0.077736	-0.077781	-0.050644

3.2 Hit test

Backtest the VaR using the **hit test**. Namely, check how many times the realized return at t was smaller than the VaR return calculated using σ_t, (where again remember the notation in the notes uses σ_t as a vol based on data through $t-1$.)

Report the percentage of "hits" using both the * expanding volatility * rolling volatility

[18]:
```
from dataclasses import dataclass

@dataclass
class BacktestResults:
    """Container for backtest results"""
    hit_rate: float
```

```
    num_hits: int
    num_observations: int
    expected_hit_rate: float
    hit_series: pl.Series
    var_series: pl.Series

    def __repr__(self) -> str:
        return (
            f"BacktestResults(\n"
            f"  Hit Rate: {self.hit_rate:.2%}\n"
            f"  Expected: {self.expected_hit_rate:.2%}\n"
            f"  Hits: {self.num_hits} / {self.num_observations}\n"
            f")"
        )
```

[19]:
```
class Backtest:

    def __init__(
            self,
            data: pl.DataFrame,
            ticker: str = "AAPL",
            loss_prob: float = 0.05
    ) -> None:

        self.data = data
        self.ticker = ticker
        self.q = loss_prob

        if ticker not in self.data.columns:
            raise ValueError(f"Ticker '{ticker}' not found in data")

    def calc_expanding_vol(self, min_periods: int=26) -> pl.DataFrame:
        result = self.data.with_columns(
            pl.col(self.ticker).shift(1)
            .cumulative_eval(
                pl.element().std()
            )
            .alias("vol_expanding")
        )
        return result[min_periods:, :]

    def calc_rolling_vol(
            self,
            window_size: int = 26,
            min_periods: int = 26
    ) -> pl.DataFrame:
        result = self.data.with_columns(
```

```
            pl.col(self.ticker).shift(1)
            .rolling_std(window_size=window_size, min_samples=min_periods)
            .alias("vol_rolling")
        ).drop_nulls()
        return result

    def _calc_var(
            self,
            test_df: pl.DataFrame,
            vol_col: str = "vol_expanding",
            mean_return: float = 0.
    ) -> pl.Series:
        """
        Calculate conditional (normal) VaR value series
        Args:
            - test_df: DataFrame with volatility column
            - vol_col: column name for vol
            - mean_return: typical to ignore
        Return:
            - VaR value over time
        """
        z = stats.norm.ppf(self.q)
        var = test_df.select(
            (pl.lit(mean_return) + z * pl.col(vol_col))
            .alias("var")
        )["var"]
        return var

    def hit_test(
            self,
            method: str = "expanding",
            params: dict = {}
    ) -> BacktestResults:

        window_size, min_periods = params["window_size"], params["min_periods"]

        if method == "expanding":
            test_df = self.calc_expanding_vol(min_periods)
        elif method == "rolling":
            test_df = self.calc_rolling_vol(window_size, min_periods)
        else:
            raise ValueError("Can only enter 'expanding' or 'rolling' for␣
↪method")

        vol_col = f"vol_{method}"
        var_series = self._calc_var(test_df, vol_col=vol_col, mean_return=0)
        test_df = test_df.with_columns(
```

```
            var_series.alias("var"),
        ).with_columns(
            (pl.col(self.ticker) < pl.col("var"))
            .cast(pl.Int32)
            .alias("hit")
        )

        n_hit = test_df.select("hit").sum().to_numpy()[0][0]
        n_obs = test_df.select(pl.len()).cast(pl.Int32).to_numpy()[0][0]
        hit_rate = n_hit / n_obs
        expd_hit_rate = self.q  # prob of loss exceeding q
        hit_series = test_df.select("hit")
        var_series = test_df.select(vol_col)

        return BacktestResults(
            hit_rate, n_hit, n_obs, expd_hit_rate, hit_series, var_series
        )

    def run(self, params):
        results = {}
        results["expanding"] = self.hit_test(method="expanding", params=params)
        results["rolling"] = self.hit_test(method="rolling", params=params)
        return results
```

[21]:
```
expanding = []
rolling = []
for ticker in tickers:
    bt = Backtest(total_rets, ticker, .05)
    params = {"window_size": 26, "min_periods": 26}
    results = bt.run(params)
    expanding.append(results["expanding"].hit_rate)
    rolling.append(results["rolling"].hit_rate)

print(
    pl.DataFrame({
        'tickers': tickers,
        'expanding_hit_rate': expanding,
        'rolling_hit_rate': rolling
    })
)
```

```
shape: (5, 3)

 tickers     expanding_hit_rate   rolling_hit_rate
 ---         ---                  ---
 str         f64                  f64

 AAPL        0.044574             0.052326
```

META	0.04845	0.044574
NVDA	0.02907	0.044574
TSLA	0.065891	0.052326
portfolio	0.052326	0.042636