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]:
```python
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]:
```python
# 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]:
```python
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]:
```python
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(0.012660439542507 52)

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.

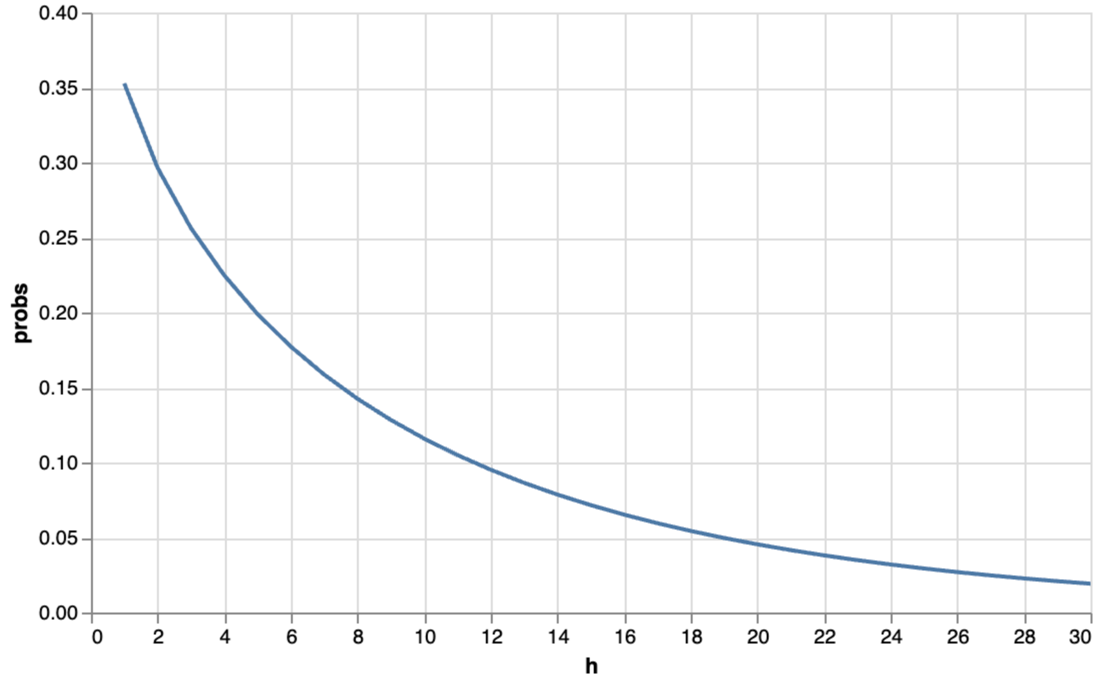
In [59]:
```
def h_years_probs(data, h, os=False, os_data=None):
    probs = []
    for i in range(1, h+1):
        probs.append(prob_underperform(data, i, os=os, os_data=os_data))

    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]:

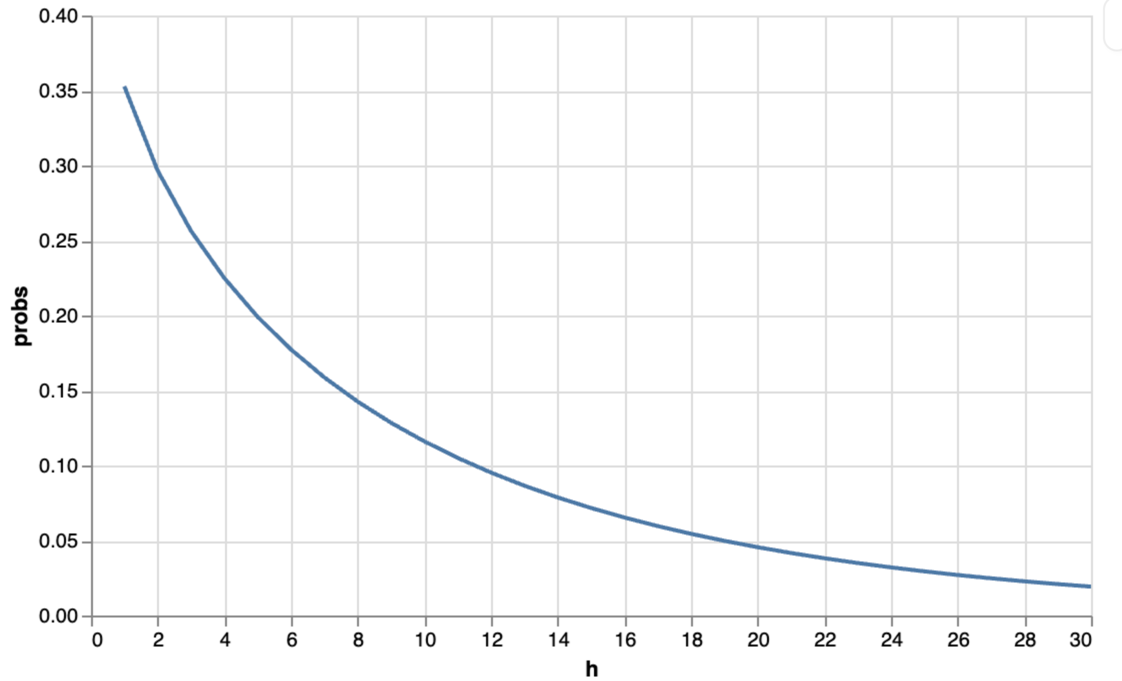


3. Full Sample Analysis

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:

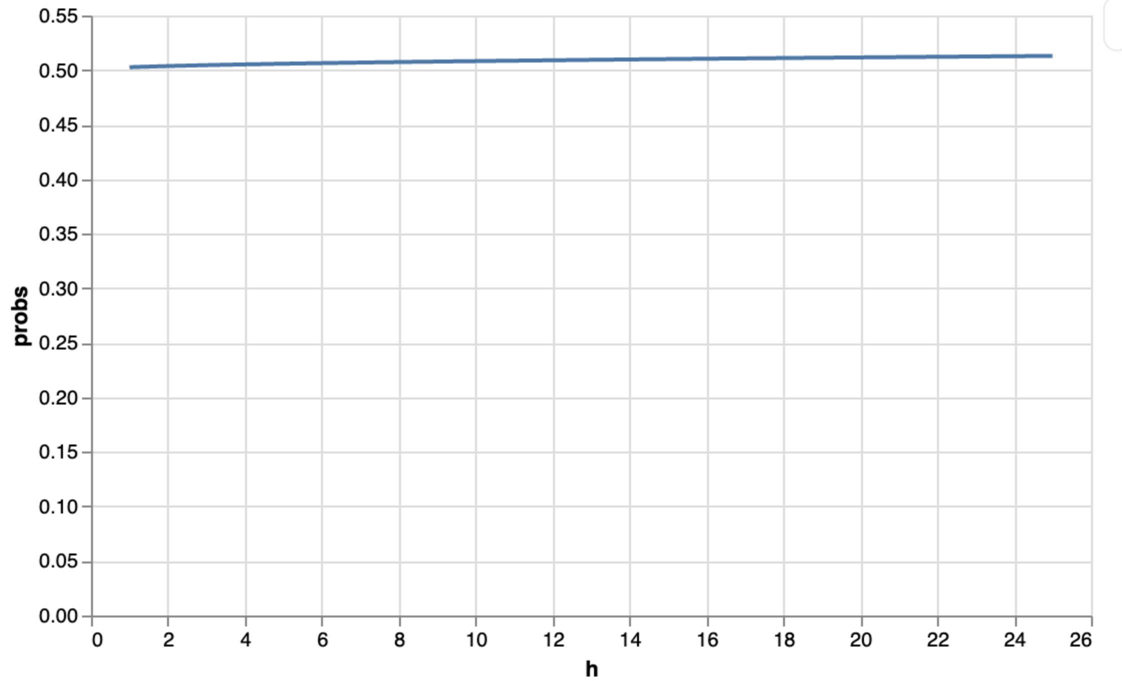
$$p = \Phi_{\mathcal{N}}\left(\sqrt{h}\,\frac{\bar{\mathbf{r}}_{\text{out-of-sample}} - \bar{\mathbf{r}}_{\text{in-sample}}}{\sigma_{\text{in-sample}}}\right)$$

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