

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	META	NVDA	TSLA
---	---	---	---
f64	f64	f64	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]: def calc_cvar(series: pl.Series) -> pl.Float64:
      return series.filter(series <= series.quantile(.05)).mean()

[5]: 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]: 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]: 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]: (
      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, $z_{.05} = -1.65$.

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: $r_{\tau,t} \sim N(\mu_{\tau,t}, \sigma_{\tau,t}^2)$ - 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	META	NVDA	TSLA	portfolio
---	---	---	---	---
f64	f64	f64	f64	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, uncon_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