checkpointc

November 8, 2025

```
[ ]: # PART 1 - Imports, config, and data download with yfinance

import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import os

# Reproducibility
np.random.seed(42)

# Output directory
OUTDIR = "aman_hist_outputs"
os.makedirs(OUTDIR, exist_ok=True)

# Universe and backtest settings
TICKERS = ["SPY", "EFA", "IEF", "VNQ", "GLD"]   # US equity, intl equity, US␣
 ↪bonds, REITs, gold
START = "2000-01-01"
END = None                                      # latest available
TRADING_DAYS_PER_YEAR = 252

# Signal and portfolio settings
REBALANCE_DAYS = 21                             # ~ monthly
MOM_LOOKBACK_DAYS = 252 - 21                    # 12-1 momentum
VOL_LOOKBACK_DAYS = 60                          # recent volatility window
CAP_PER_ASSET = 0.40                            # 40% per-asset cap
TC_PER_TURNOVER = 0.0005                        # 5 bps per unit turnover on␣
 ↪rebalance

# Management fee (use same rate family as last checkpoint: 1% )
MGMT_FEE_ANNUAL = 0.01                          # 1.00% per year
MGMT_FEE_DAILY = MGMT_FEE_ANNUAL / TRADING_DAYS_PER_YEAR

# Download adjusted prices
raw = yf.download(TICKERS, start=START, end=END, auto_adjust=True,␣
 ↪progress=False)
```

```
if isinstance(raw.columns, pd.MultiIndex) and "Adj Close" in raw.columns.
 ↪get_level_values(0):
    px = raw["Adj Close"].copy()
else:
    # fallback: try Close
    px = raw["Close"].copy() if "Close" in raw.columns else raw.copy()

# Clean and align
px = px.dropna(how="all").ffill().dropna()
px = px[TICKERS].dropna()
px = px.asfreq("B").ffill()                      # business days
ret = px.pct_change().dropna()

px.to_csv(os.path.join(OUTDIR, "prices.csv"))
ret.to_csv(os.path.join(OUTDIR, "returns.csv"))
```

[2]:
```
# PART 2 - AMAN signal construction and monthly weight schedule
# 12-1 momentum + inverse-volatility scaling + 40% cap, with defensive fallback

def compute_weights_from_window(window_prices: pd.DataFrame) -> pd.Series:
    # Momentum over lookback window (end / start - 1)
    momentum = window_prices.iloc[-1] / window_prices.iloc[0] - 1.0

    # Recent volatility using last VOL_LOOKBACK_DAYS of simple returns
    r = window_prices.pct_change().dropna()
    recent = r.tail(VOL_LOOKBACK_DAYS)
    vol = recent.std(ddof=1).replace(0, np.nan)

    inv_vol = 1.0 / vol
    pos_momo = (momentum > 0).astype(float)

    raw = pos_momo * inv_vol
    raw = raw.replace([np.inf, -np.inf], 0.0).fillna(0.0)

    # If no positive momentum assets, fallback to defensives
    if raw.sum() <= 0:
        w = pd.Series(0.0, index=window_prices.columns)
        if "IEF" in w.index: w["IEF"] = 0.6
        if "GLD" in w.index: w["GLD"] = 0.4
        if w.sum() == 0: w[:] = 1.0 / len(w)
        return w

    # Normalize
    w = raw / raw.sum()

    # Cap per asset and renormalize iteratively
    for _ in range(3):
        over = w > CAP_PER_ASSET
        if not over.any(): break
        w[over] = CAP_PER_ASSET
        if w.sum() > 0:
            w = w / w.sum()

    w = w / w.sum()
    return w

# Build monthly rebalancing weights using 12-1 momentum
weights = []
dates = []
last_reb_idx = None

for i, dt in enumerate(px.index):
    # need history for 12-1 momentum -> window ends 21 trading days before dt
    if i < MOM_LOOKBACK_DAYS + 21:
        continue
    if last_reb_idx is None or (i - last_reb_idx) >= REBALANCE_DAYS:
        end_idx = i - 21
        start_idx = end_idx - MOM_LOOKBACK_DAYS
        window = px.iloc[start_idx:end_idx + 1]
        w = compute_weights_from_window(window)
        weights.append(w)
        dates.append(dt)
        last_reb_idx = i

weights_df = pd.DataFrame(weights, index=pd.DatetimeIndex(dates))
weights_df.to_csv(os.path.join(OUTDIR, "aman_weights.csv"))
```

[3]:
```
# PART 3 - Backtest engine: AMAN gross and net vs Equal-Weight and 60/40
# Net curve deducts management fee daily and turnover costs on rebalance days

# Benchmark constant weights
ew_w = pd.Series(1.0 / len(TICKERS), index=TICKERS)
w_6040 = pd.Series(0.0, index=TICKERS)
w_6040[["SPY", "EFA", "VNQ"]] = 0.60 / 3.0
w_6040["IEF"] = 0.40

def backtest_dynamic(weights_df: pd.DataFrame,
                     ret: pd.DataFrame,
                     fee_daily: float = 0.0,
                     tc_per_turnover: float = 0.0,
                     name: str = "portfolio") -> pd.DataFrame:
    # Align weights to return dates and forward fill to next rebalance
    w = weights_df.reindex(ret.index, method="ffill")
    # Early dates before first weight: use equal weight
```

```python
    w = w.combine_first(pd.DataFrame(np.tile(ew_w.values, (len(ret), 1)),
                                     index=ret.index, columns=ret.columns))
    w = w.fillna(method="ffill")

    # Detect rebalances as any row change vs previous day
    is_reb = w.ne(w.shift(1)).any(axis=1)

    # Daily gross return
    port_ret_gross = (w * ret).sum(axis=1)

    # Management fee applied daily
    port_ret_net = port_ret_gross - fee_daily

    # Turnover and transaction costs on rebalance days
    turnover = (w - w.shift(1)).abs().sum(axis=1).fillna(0.0)
    tc = tc_per_turnover * turnover
    port_ret_net_tc = port_ret_net - tc.where(is_reb, 0.0)

    # Value curves
    v_gross = (1.0 + port_ret_gross).cumprod()
    v_net = (1.0 + port_ret_net_tc).cumprod()

    df = pd.DataFrame({
        f"{name}_gross": v_gross,
        f"{name}_net": v_net
    })
    return df

def backtest_static(w_const: pd.Series, ret: pd.DataFrame, name: str) -> pd.
 ↪Series:
    port_ret = (w_const * ret).sum(axis=1)
    v = (1.0 + port_ret).cumprod()
    return v.rename(name)

aman_df = backtest_dynamic(weights_df=weights_df,
                           ret=ret,
                           fee_daily=MGMT_FEE_DAILY,
                           tc_per_turnover=TC_PER_TURNOVER,
                           name="AMAN")

ew_curve = backtest_static(ew_w, ret, "EW_gross")
s6040_curve = backtest_static(w_6040, ret, "60_40_gross")

aman_df.to_csv(os.path.join(OUTDIR, "aman_values.csv"))
pd.concat([ew_curve, s6040_curve], axis=1).to_csv(os.path.join(OUTDIR,␣
 ↪"bench_values.csv"))
```

```
/var/folders/qk/zn5nr6ws0gz0yxbp7gph1gm40000gn/T/ipykernel_97925/847828275.py:20
```

```
: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in
a future version. Use obj.ffill() or obj.bfill() instead.
  w = w.fillna(method="ffill")
```

[5]:
```
# PART 4 - Performance metrics and gross vs net comparison (fixed index labels)

def ann_return_from_curve(series: pd.Series) -> float:
    dr = series.pct_change().dropna()
    return (1.0 + dr.mean())**TRADING_DAYS_PER_YEAR - 1.0

def ann_vol_from_curve(series: pd.Series) -> float:
    dr = series.pct_change().dropna()
    return dr.std(ddof=1) * np.sqrt(TRADING_DAYS_PER_YEAR)

def sharpe_from_curve(series: pd.Series, rf_annual: float = 0.0) -> float:
    dr = series.pct_change().dropna()
    ex = dr - rf_annual / TRADING_DAYS_PER_YEAR
    vol = ex.std(ddof=1)
    return np.sqrt(TRADING_DAYS_PER_YEAR) * (ex.mean() / vol if vol > 0 else np.
 ↪nan)

def max_drawdown(series: pd.Series) -> float:
    roll_max = series.cummax()
    dd = series / roll_max - 1.0
    return float(dd.min())

def summarize_curve(series: pd.Series) -> pd.Series:
    return pd.Series({
        "ann_return": ann_return_from_curve(series),
        "ann_vol": ann_vol_from_curve(series),
        "sharpe": sharpe_from_curve(series),
        "max_drawdown": max_drawdown(series),
        "terminal_value": float(series.iloc[-1])
    })

# Build summary with explicit, stable index labels
summary_dict = {
    "AMAN_gross": summarize_curve(aman_df["AMAN_gross"]),
    "AMAN_net":   summarize_curve(aman_df["AMAN_net"]),
    "EW_gross":   summarize_curve(ew_curve),
    "60_40_gross": summarize_curve(s6040_curve),
}
summary = pd.DataFrame(summary_dict).T

# Net vs gross relative impact for AMAN
summary.loc["AMAN_net", "net_vs_gross_relative"] = (
    summary.loc["AMAN_net", "terminal_value"] / summary.loc["AMAN_gross",␣
  ↪"terminal_value"] - 1.0
)

summary.to_csv(os.path.join(OUTDIR, "summary.csv"))
summary
```

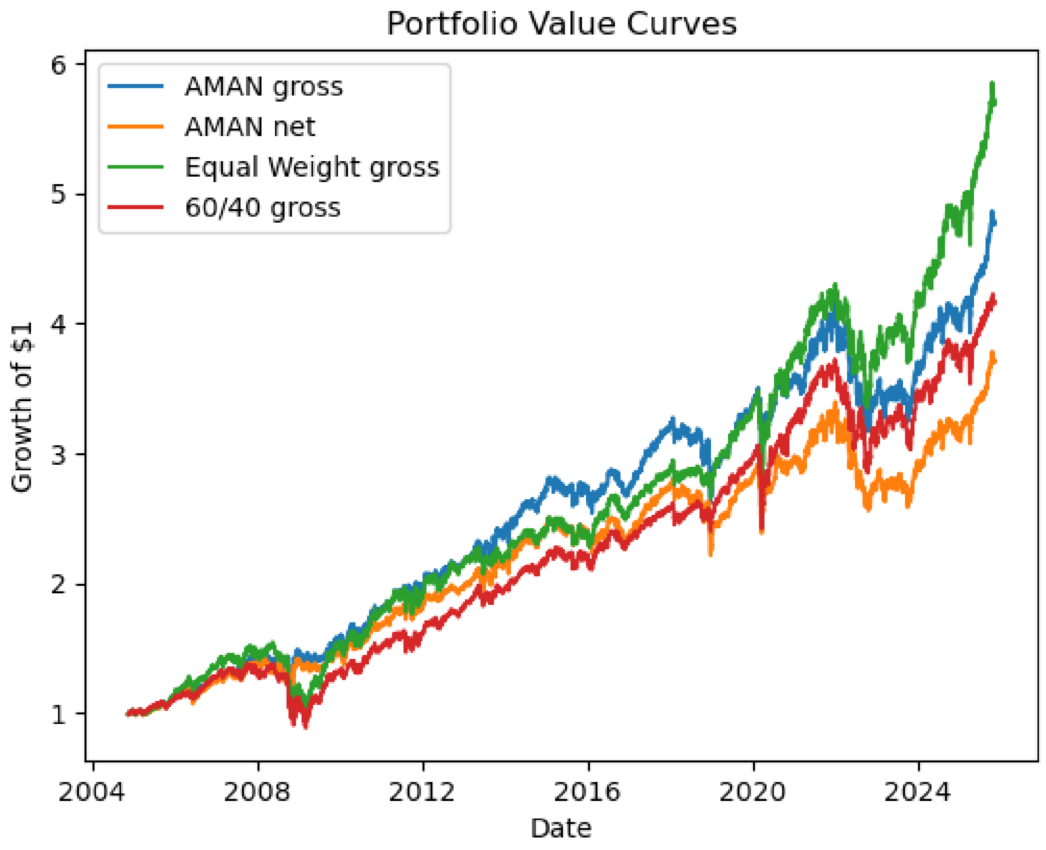
[5]:

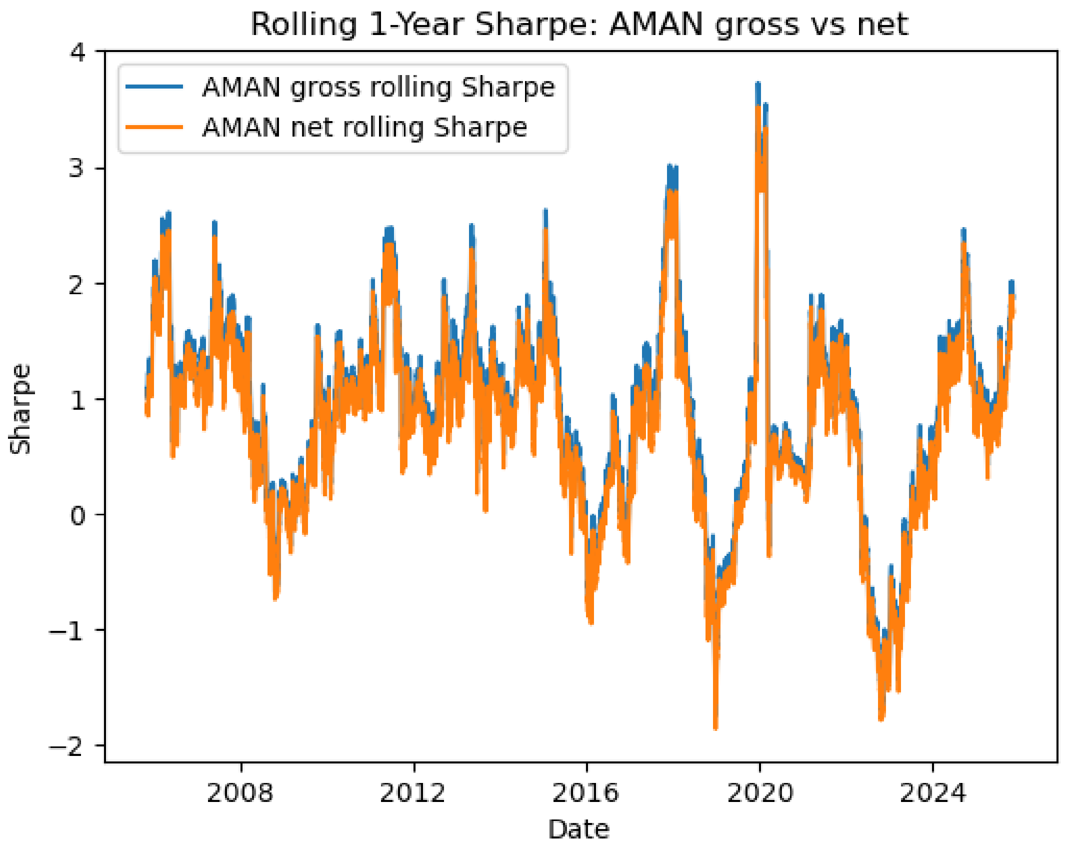
```
             ann_return   ann_vol    sharpe  max_drawdown  terminal_value  \
AMAN_gross     0.080563  0.102773  0.754027     -0.238388        4.769604
AMAN_net       0.068083  0.102779  0.640928     -0.245995        3.706335
EW_gross       0.093068  0.130503  0.682009     -0.373457        5.708147
60_40_gross    0.076068  0.120502  0.608489     -0.356758        4.162018

             net_vs_gross_relative
AMAN_gross                     NaN
AMAN_net                 -0.222926
EW_gross                       NaN
60_40_gross                    NaN
```

[6]:

```
# PART 5 - Plots: portfolio curves and rolling 1-year Sharpe (gross vs net)
# One chart per figure and default colors only

# Curves
plt.figure()
plt.plot(aman_df.index, aman_df["AMAN_gross"], label="AMAN gross")
plt.plot(aman_df.index, aman_df["AMAN_net"], label="AMAN net")
plt.plot(ew_curve.index, ew_curve, label="Equal Weight gross")
plt.plot(s6040_curve.index, s6040_curve, label="60/40 gross")
plt.title("Portfolio Value Curves")
plt.xlabel("Date")
plt.ylabel("Growth of $1")
plt.legend()
plt.savefig(os.path.join(OUTDIR, "curves.png"), dpi=150, bbox_inches="tight")
plt.show()

# Rolling Sharpe helper
def rolling_sharpe_from_returns(dr: pd.Series, window: int =␣
  ↪TRADING_DAYS_PER_YEAR) -> pd.Series:
    return dr.rolling(window).apply(
        lambda x: (np.sqrt(TRADING_DAYS_PER_YEAR) * (x.mean() / x.std(ddof=1)))␣
  ↪if x.std(ddof=1) > 0 else np.nan,
        raw=False
    )

# Recompute daily returns for AMAN gross vs net
aman_gross_dr = aman_df["AMAN_gross"].pct_change().dropna()
aman_net_dr = aman_df["AMAN_net"].pct_change().dropna()
```

```
plt.figure()
plt.plot(rolling_sharpe_from_returns(aman_gross_dr), label="AMAN gross rolling␣
  ↪Sharpe")
plt.plot(rolling_sharpe_from_returns(aman_net_dr), label="AMAN net rolling␣
  ↪Sharpe")
plt.title("Rolling 1-Year Sharpe: AMAN gross vs net")
plt.xlabel("Date")
plt.ylabel("Sharpe")
plt.legend()
plt.savefig(os.path.join(OUTDIR, "rolling_sharpe.png"), dpi=150,␣
  ↪bbox_inches="tight")
plt.show()
```





```
[ ]: !jupyter nbconvert --to pdf checkpointc.ipynb
```

```
[ ]:
```

```
[ ]:
```