Portfolio Risk Analyzer with PySpark (SQL) Past Year + Next 6 Months

Juan Attias

August 3, 2025

Objective

- Simulate a diversified portfolio and compute valuation & risk using **PySpark SQL**.
- Generate **past ∼1 trading year** and **forward 6 months** (GBM paths).
- Produce tables, CSVs, and visualizations for reporting.

Tools Used

- **PySpark** (Spark SQL for transformations)
- **Pandas** (Windows-friendly CSV export)
- **Matplotlib** (charts)
- **GBM** for synthetic prices; risk-free rate flat $\approx 5\%/\mathrm{yr}$

Methodology

1. Generate business-day prices for tickers (SPY, AGG, EFA, EEM, VNQ, GLD, TLT, QQQ, SHY).
2. Size positions to a $1M portfolio with target weights.
3. Spark SQL:
   - `portfolio_daily_value`, `portfolio_daily_returns`
   - `portfolio_asset_daily_value`, `market_daily_returns`
   - `risk_metrics` (Beta, intercept, Sharpe, VaR95/99)

Code: Data Ingestion & Views (Lines 1–7)

Python (PySpark)

```
1 # (1) Create DataFrame with simulated prices
2 prices_df = spark.createDataFrame(prices_data, schema=["date","ticker","close
      "])
3
4 # (4) Ensure correct dtypes and register as a SQL view
5 from pyspark.sql.functions import col, to_date
6 prices_df = prices_df.select(to_date(col("date")).alias("date"),
7                              col("ticker").cast("string"),
8                              col("close").cast("double"))
9 prices_df.createOrReplaceTempView("prices")
```

Why it matters

- **L1–2**: Centralizes market data in a distributed DataFrame for scale.
- **L5–7**: Enforces schema (date/double) to prevent downstream numeric/time errors.
- **L8**: Registers `prices` so every step can be written as SQL (auditable).

Code: Ticker Mapping with `ROW_NUMBER()` (Lines 1–9)

SQL (Spark SQL)

```sql
1 -- (1) Surrogate keys for tickers
2 CREATE OR REPLACE TEMP VIEW ticker_map AS
3 SELECT
4   ticker,
5   ROW_NUMBER() OVER (ORDER BY ticker) AS ticker_id
6 FROM (
7   SELECT DISTINCT ticker FROM prices
8 ) t;
```

Why it matters

- **L3–7**: `ROW_NUMBER()` yields stable numeric IDs; joins/group-bys get faster.
- **L6**: `DISTINCT` isolates unique tickers to avoid duplicate IDs.

Code: Position Sizing to $1M (Lines 1–10)

SQL (Spark SQL)

```
1  -- (1) Final hist date for sizing
2  CREATE OR REPLACE TEMP VIEW last_hist AS
3  SELECT MAX(date) AS last_date FROM prices;
4
5  -- (5) Size positions at last historical close to $1M notional
6  CREATE OR REPLACE TEMP VIEW portfolio_positions AS
7  WITH last_px AS (
8    SELECT p.ticker, p.close
9    FROM prices p JOIN last_hist h ON p.date = h.last_date
10 ),
11 weights AS (
12   SELECT 'SPY' ticker, 0.25 w UNION ALL SELECT 'QQQ', 0.15 UNION ALL
13   SELECT 'AGG', 0.15 UNION ALL SELECT 'TLT', 0.10 UNION ALL
14   SELECT 'EFA', 0.10 UNION ALL SELECT 'EEM', 0.10 UNION ALL
15   SELECT 'VNQ', 0.05 UNION ALL SELECT 'GLD', 0.05 UNION ALL
16   SELECT 'SHY', 0.05
17 )
18 SELECT
19   l.ticker,
20   CAST( (w.w * 1000000.0) / l.close AS DOUBLE ) AS quantity
21 FROM last_px l JOIN weights w USING (ticker);
```

Why it matters

- **L1–3**: Establishes sizing date once (consistent across tables).
- **L6–8**: Captures last close per ticker; sizing uses observable prices.
- **L9–15**: Explicit weights are auditable and easy to tweak.
- **L19–21**: Positions are derived, not hand-picked; reproducible and exact.

Code: Portfolio Daily Value (Lines 1–9)

SQL (Spark SQL)

```sql
1 -- (1) Daily total value across assets
2 CREATE OR REPLACE TEMP VIEW portfolio_daily_value AS
3 SELECT
4   p.date,
5   SUM(pos.quantity * p.close) AS total_value
6 FROM prices p
7 JOIN portfolio_positions pos USING (ticker)
8 GROUP BY p.date
9 ORDER BY p.date;
```

Why it matters

- **L4–6**: The book is valued like a backtester: quantity $\times$ close.
- **L7–9**: `GROUP BY` and ordering ensure a canonical, time-ordered value path.

Code: Daily Returns via `LAG()` (Lines 1–9)

SQL (Spark SQL)

```sql
1 -- (1) Portfolio daily returns
2 CREATE OR REPLACE TEMP VIEW portfolio_daily_returns AS
3 SELECT
4   date,
5   (total_value / LAG(total_value) OVER (ORDER BY date) - 1) AS port_ret
6 FROM portfolio_daily_value
7 ORDER BY date;
```

Why it matters

- **L4–6**: Window function `LAG()` gives exact prior total for percentage change.
- **L6–7**: Declarative, auditable time-series transform; no imperative loops.

Code: Market Returns Extraction (Lines 1–10)

SQL (Spark SQL)

```sql
1 -- (1) SPY market factor
2 CREATE OR REPLACE TEMP VIEW market_daily_returns AS
3 SELECT
4   date,
5   (close / LAG(close) OVER (ORDER BY date) - 1) AS mkt_ret
6 FROM prices
7 WHERE ticker = 'SPY'
8 ORDER BY date;
```

Why it matters

- **L4–6**: Mirrors portfolio return logic; factor series is computed identically.
- **L7**: Isolation of SPY as a clean single-factor benchmark for beta/alpha.

Code: Asset-Level Value for Attribution (Lines 1–10)

SQL (Spark SQL)

```sql
1 -- (1) Per-asset daily position value
2 CREATE OR REPLACE TEMP VIEW portfolio_asset_daily_value AS
3 SELECT
4   p.date,
5   p.ticker,
6   pos.quantity,
7   p.close,
8   pos.quantity * p.close AS position_value
9 FROM prices p
10 JOIN portfolio_positions pos USING (ticker);
```

Why it matters

- **L4–8**: Retains granular contributions for allocation charts/attribution.
- **L9–10**: Logical view (lazy) until materialized; efficient in Spark.

Code: Join & Risk Metrics (Lines 1–16)

SQL (Spark SQL)

```sql
1  -- (1) Join returns
2  CREATE OR REPLACE TEMP VIEW joined_returns AS
3  SELECT p.date, p.port_ret, m.mkt_ret
4  FROM portfolio_daily_returns p
5  JOIN market_daily_returns  m USING (date);
6
7  -- (6) Summary statistics + beta/alpha
8  CREATE OR REPLACE TEMP VIEW risk_metrics AS
9  SELECT
10   COUNT(*) AS days,
11   AVG(port_ret) AS mean_ret,
12   STDDEV(port_ret) AS std_ret,
13   COVAR_POP(port_ret, mkt_ret) / VAR_POP(mkt_ret) AS beta,
14   AVG(port_ret) - (COVAR_POP(port_ret, mkt_ret)/VAR_POP(mkt_ret))*AVG(mkt_ret
        ) AS intercept
15 FROM joined_returns;
```

Why it matters

- **L1–5**: Left as an inner join to align trading days exactly.
- **L9–15**: Pure-SQL beta/intercept formulae (no UDFs) — portable and transparent.

Code: VaR (Historical & Parametric) (Lines 1–17)

SQL (Spark SQL)

```sql
1 -- (1) VaR using empirical quantiles + Normal approx
2 CREATE OR REPLACE TEMP VIEW risk_metrics_var AS
3 WITH stats AS (
4   SELECT
5     AVG(port_ret) mu,
6     STDDEV(port_ret) sigma
7   FROM joined_returns
8 ),
9 hist AS (
10   SELECT
11     PERCENTILE_CONT(port_ret, 0.05) AS var95_hist,
12     PERCENTILE_CONT(port_ret, 0.01) AS var99_hist
13   FROM joined_returns
14 )
15 SELECT
16   h.var95_hist, h.var99_hist,
17   (s.mu + s.sigma * (-1.64485)) AS var95_param,
18   (s.mu + s.sigma * (-2.32635)) AS var99_param
19 FROM stats s CROSS JOIN hist h;
```

Why it matters

- **L4–8**: Central limit stats in SQL for parametric VaR.
- **L9–12**: `PERCENTILE_CONT` for non-parametric (historical) VaR.
- **L14–16**: Z-scores baked in; easy to stress with alternative cutoffs.

Code: Export & Plot (Lines 1–9)

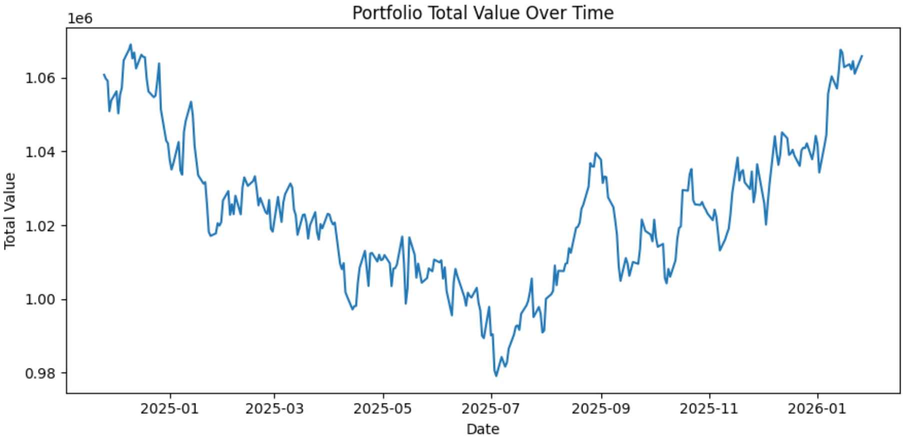
Python (PySpark & Pandas/Matplotlib)

```
1 # (1) Export single CSVs (Windows-friendly)
2 risk_pd = spark.sql("SELECT * FROM risk_metrics").toPandas()
3 risk_var_pd = spark.sql("SELECT * FROM risk_metrics_var").toPandas()
4 risk_pd.to_csv("outputs/risk_metrics.csv", index=False)
5 risk_var_pd.to_csv("outputs/risk_metrics_var.csv", index=False)
6
7 # (7) Plot (after narrow collect)
8 val_pd = spark.sql("SELECT * FROM portfolio_daily_value").toPandas()
9 plt.plot(val_pd["date"], val_pd["total_value"]); plt.title("Portfolio Value
      Over Time")
```

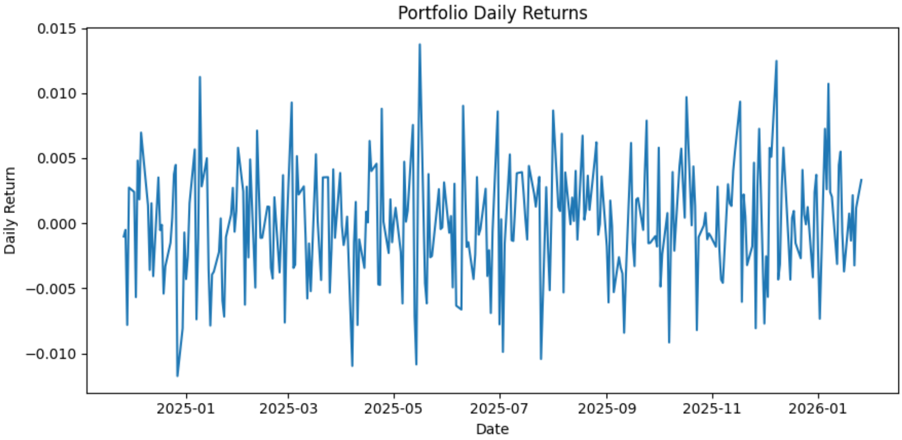
Why it matters

- **L1–5**: Narrow to Pandas only at the edges (I/O, plots) — Spark does the heavy lifting.
- **L7–9**: Presentation-ready figures while retaining SQL auditability upstream.

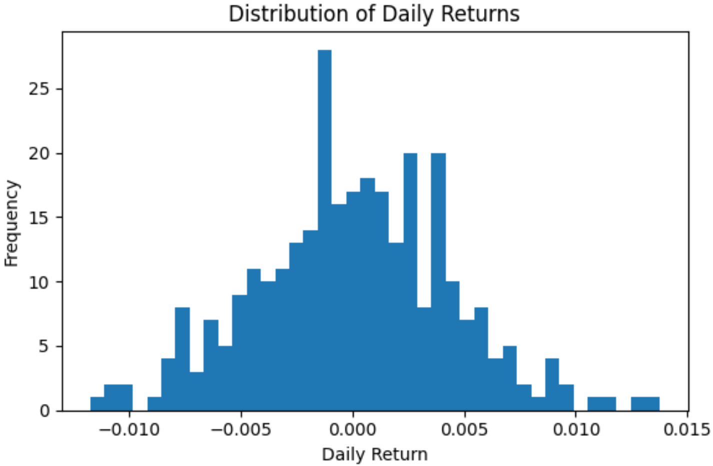
Portfolio Value Over Time



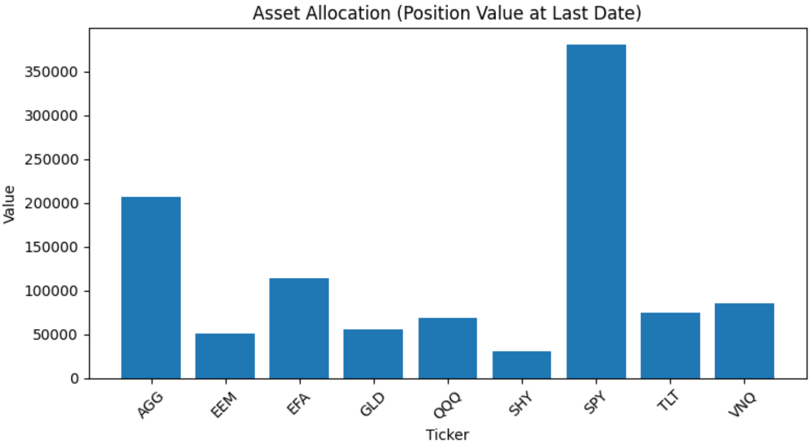
Portfolio Daily Returns



Distribution of Daily Returns



Asset Allocation (Position Value at Last Date)



Risk Metrics

Portfolio	As of	Days	Beta	Intercept	Sharpe	VaR_{95} (hist)
Well Rounded Portfolio	2026-01-26	305	0.363598	0.000045	-0.633985	0.007631

Portfolio	As of	Days	VaR_{99} (hist)	VaR_{95} (param)	VaR_{99} (param)	Mean / Std
Well Rounded Portfolio	2026-01-26	305	0.010432	-0.007226	-0.010209	0.000025 / 0.004378

Table: Risk metrics calculated from simulated daily returns (past $\sim$1Y + next 6 months). VaR values are daily magnitudes; parametric assumes normality.

Conclusions

- Portfolio beta ≈ 0.36 vs. SPY indicates lower market sensitivity.
- Sharpe (annualized) is negative in this run, reflecting return vs. volatility under the simulated path.
- VaR indicates typical daily loss bounds at 95% and 99% confidence.

Future Work

- Use real market data (Alpha Vantage, Yahoo Finance, Quandl) for backtests.
- Extend risk: ES/CVaR, drawdown, regime switching, multi-factor betas.
- Multiple portfolios, rebalancing logic, and scenario analysis.