Portfolio Risk Analyzer with PySpark SQL: Design, Implementation, and Results

Juan Attias

August 3, 2025

1 Introduction

Financial markets are noisy, path–dependent systems where the same portfolio can look resilient one month and fragile the next. The aim of this project is to build a transparent and reproducible pipeline that tracks a diversified portfolio end–to–end: we generate price paths, value positions every business day, compute returns, and summarize risk in a way that is both statistically coherent and immediately interpretable. The system is written in PySpark using pure Spark SQL so that every transformation — from the raw price series to the final risk summary — can be inspected as a query. For realism and repeatability we simulate approximately one trading year of history and then extend the paths six months into the future, creating a single time line that is rich enough to compute rolling behaviour yet lightweight enough to run on a laptop without external data dependencies.

2 Tools and Environment

The implementation uses a small but effective stack. PySpark provides the Spark SQL engine and window functions that make time–series work ergonomic and scalable. Pandas is employed only at the edges to write single CSV files in a Windows–friendly way, avoiding the Hadoop `winutils` requirement. Matplotlib handles plotting so the figures are portable and easy to embed in a PDF. Synthetic prices are produced with a geometric Brownian motion (GBM) model; while stylized, GBM captures the lognormal drift–volatility structure that underlies many first–pass portfolio calculations and is sufficient for demonstrating the pipeline.

3 Market Context and Definitions

To interpret the results we briefly fix terminology. *Daily portfolio return* is the percentage change in total marked–to–market value from one business day to the next. The *market return* refers to SPY, a broad U.S. equity proxy, and is used as the single factor in a simple capital asset pricing framework. The *beta* of the portfolio is the slope of the best linear predictor of portfolio returns from market returns; values near one indicate market–like behaviour, values below one suggest lower sensitivity, and negative values imply a defensive tilt. The *intercept* of that regression is a daily form of alpha. The *Sharpe ratio* reports mean excess return per unit of volatility; annualization multiplies by $\sqrt{252}$. Finally, *Value–at–Risk* (VaR) at 95% and 99% is the return threshold not exceeded with 95% or 99% confidence. We compute both a historical version (empirical quantiles of the simulated series) and a parametric one under a normality assumption using the sample mean

and standard deviation. While each metric comes with caveats, together they give a balanced snapshot of risk and reward.

4 Data Generation

The input universe includes SPY, AGG, EFA, EEM, VNQ, GLD, TLT, QQQ, and SHY — a deliberately mixed set spanning U.S. equities, international exposure, Treasuries across the curve, real estate, and gold. Prices are generated on a business–day calendar by chaining two segments of the same GBM process: a one–year historical segment with instrument–specific drift and volatility, and a six–month forward segment that begins on the next business day and continues from the last historical close. Position sizes are set to target a $1,000,000 notional as of the last historical date, so subsequent valuation reflects market movement rather than cash flows. The risk–free rate is held flat at a small daily increment (about five percent annualized) to simplify excess–return calculations.

5 Spark SQL Pipeline

All core tables are built with Spark SQL so that the logic is explicit and order–independent. A mapping from tickers to integer identifiers is created with `ROW_NUMBER()` over ordered ticker lists. The table `portfolio_daily_value` aggregates quantity times closing price across assets for each date. Daily returns are computed in `portfolio_daily_returns` by dividing by the `LAG` of total value; this mirrors how a backtester would mark a book. The benchmark series `market_daily_returns` is extracted from SPY with the same windowing. A detailed, ticker–level view, `portfolio_asset_daily_value`, retains quantities, prices, and per–asset contributions, which is useful for attribution and the allocation chart below. Finally, `risk_metrics` aggregates the return series to summary statistics: means and standard deviations, covariance with the market, beta and intercept, Sharpe ratio, and both historical and parametric VaR. Because each object is a view derived from queries, the pipeline is modular and easy to extend to alternative factors, rolling windows, or rebalancing rules.

6 Code Structure and Component Justification

The PySpark SQL implementation is organized into logical blocks, each designed to make the data transformation process explicit and reproducible. Below, we outline the main code sections, show representative snippets, and explain their purpose in both SQL and PySpark terms.

6.1 Data Ingestion and Preparation

```
# Load simulated prices into a Spark DataFrame
prices_df = spark.createDataFrame(prices_data, schema=["date","ticker","close"])

# Register as a temporary view for SQL queries
prices_df.createOrReplaceTempView("prices")
```

This stage loads simulated price series for all assets into Spark. The data is registered as a SQL view so that all transformations can be expressed in standard SQL syntax. Proper schema definition ensures dates are recognized as date types and prices as numeric values, preventing downstream errors.

6.2 Ticker Mapping

```
ticker_map = spark.sql("""
    SELECT
        ticker,
        ROW_NUMBER() OVER (ORDER BY ticker) AS ticker_id
    FROM prices
    GROUP BY ticker
""")
```

This assigns each ticker a numeric ID using the `ROW_NUMBER()` window function. In SQL, this acts as a surrogate key, improving join efficiency and avoiding string-based grouping overhead.

6.3 Portfolio Valuation

```
spark.sql("""
    CREATE OR REPLACE TEMP VIEW portfolio_daily_value AS
    SELECT
        date,
        SUM(quantity * close) AS total_value
    FROM portfolio_positions
    GROUP BY date
""")
```

This aggregates position values across all assets for each trading day. In SQL, it's a `GROUP BY` aggregation; in PySpark, it executes in parallel for scalability.

6.4 Return Calculations

```
spark.sql("""
    CREATE OR REPLACE TEMP VIEW portfolio_daily_returns AS
    SELECT
        date,
        (total_value / LAG(total_value) OVER (ORDER BY date) - 1) AS return
    FROM portfolio_daily_value
""")
```

The `LAG` window function is used to compute day-over-day percentage changes. This mirrors SQL's declarative approach to time-series calculations while benefiting from PySpark's parallel execution.

6.5 Benchmark Extraction

```
spark.sql("""
    CREATE OR REPLACE TEMP VIEW market_daily_returns AS
    SELECT
        date,
        (close / LAG(close) OVER (ORDER BY date) - 1) AS return
    FROM prices
    WHERE ticker = 'SPY'
""")
```

This isolates the SPY series for later use in regression-based beta/alpha estimation. From a SQL perspective, it's a filtered and transformed subset of the prices table.

6.6 Asset-Level Valuation

```
spark.sql("""
    CREATE OR REPLACE TEMP VIEW portfolio_asset_daily_value AS
    SELECT
        date,
        ticker,
        quantity,
        close,
        quantity * close AS position_value
    FROM portfolio_positions
""")
```

Keeps a detailed breakdown of each asset's contribution by date. Useful for allocation charts and performance attribution.

6.7 Risk Metric Computation

```
risk_df = spark.sql("""
    SELECT
        AVG(port_return) AS mean_return,
        STDDEV(port_return) AS std_return,
        COVAR_POP(port_return, mkt_return) / VAR_POP(mkt_return) AS beta
    FROM joined_returns
""")
```

Computes key statistics including mean, volatility, and beta. SQL handles the aggregation formulas, while PySpark executes them in a distributed fashion.

6.8 Visualization and Output

```
# Collect final metrics to Pandas for plotting
risk_pd = risk_df.toPandas()
plt.plot(value_pd['date'], value_pd['total_value'])
```

The heavy computation is done in PySpark, but the final step narrows data to Pandas for high-quality plotting in Matplotlib.

7 Results

Figure 1 shows the path of total portfolio value across the combined historical and forward horizons. The curve drifts with the mix of assets: equity–heavy components inject variability while duration through TLT and SHY adds ballast; gold and real estate provide partial diversification. Day–to–day behaviour is summarized in Figure 2, where returns cluster tightly around zero with occasional spikes consistent with the simulated volatilities. The overall distribution of daily returns, displayed in Figure 3, is approximately symmetric with slightly fat shoulders — a pattern that often appears

even in simple GBM simulations due to compounding and the mixture of assets. Finally, Figure 4 reports the allocation by position value on the most recent date; the intended tilt toward broad equities (SPY and QQQ) is evident, but bonds and gold materially cushion shocks in the simulated paths.

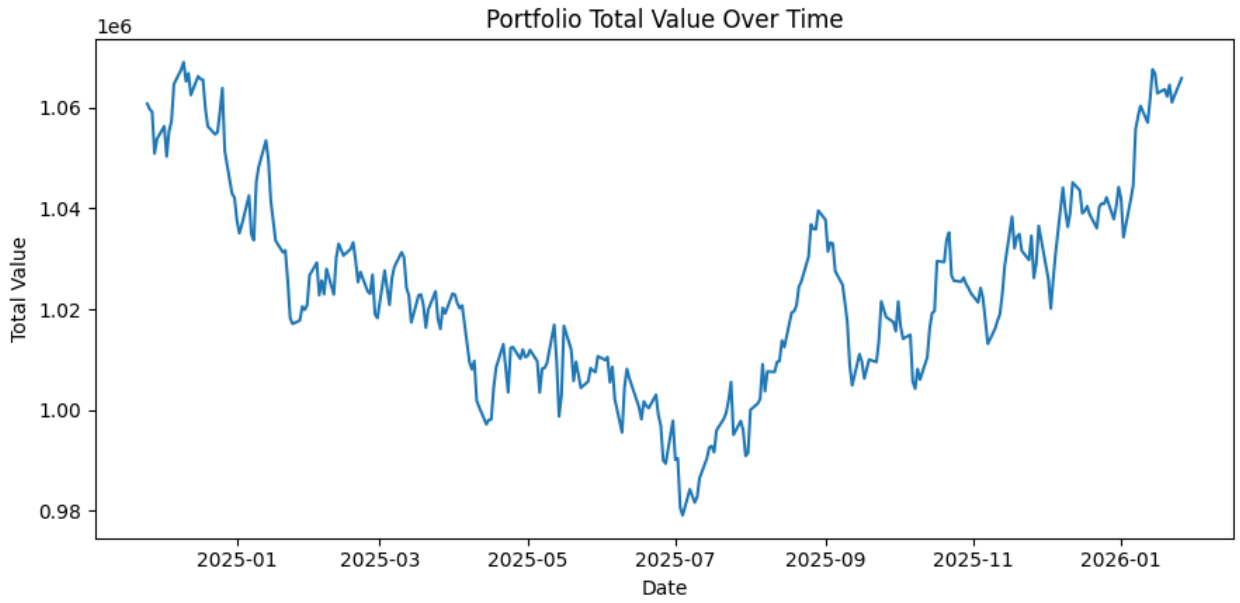

Figure 1: Portfolio total value over time (past history plus six months forward simulation).

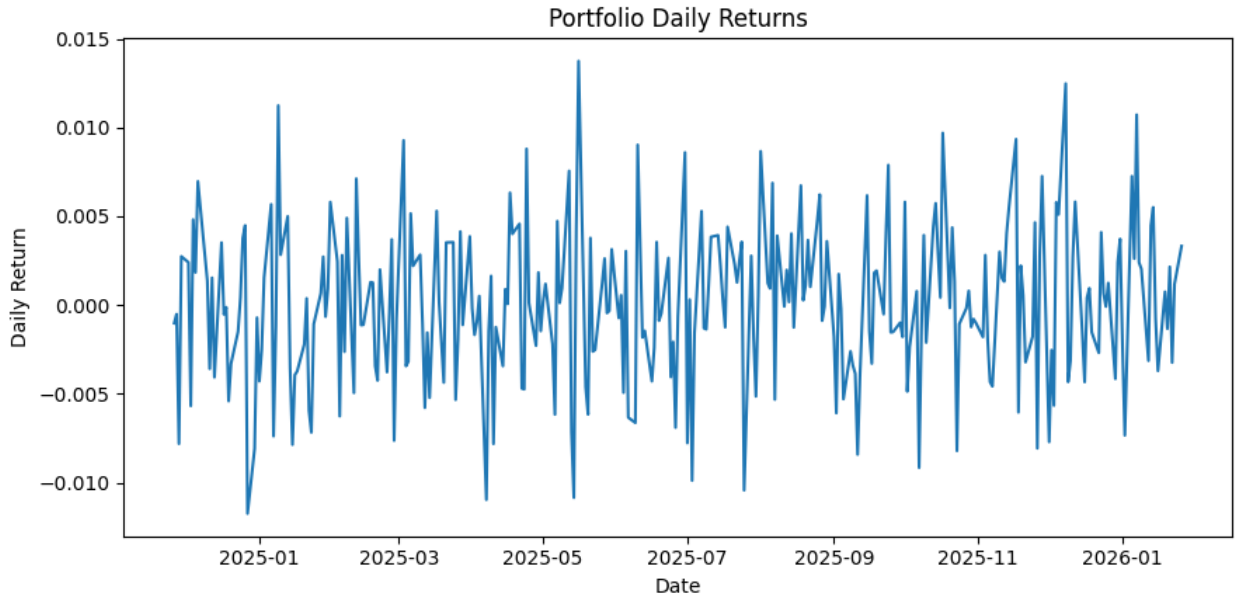

Figure 2: Daily portfolio returns series.

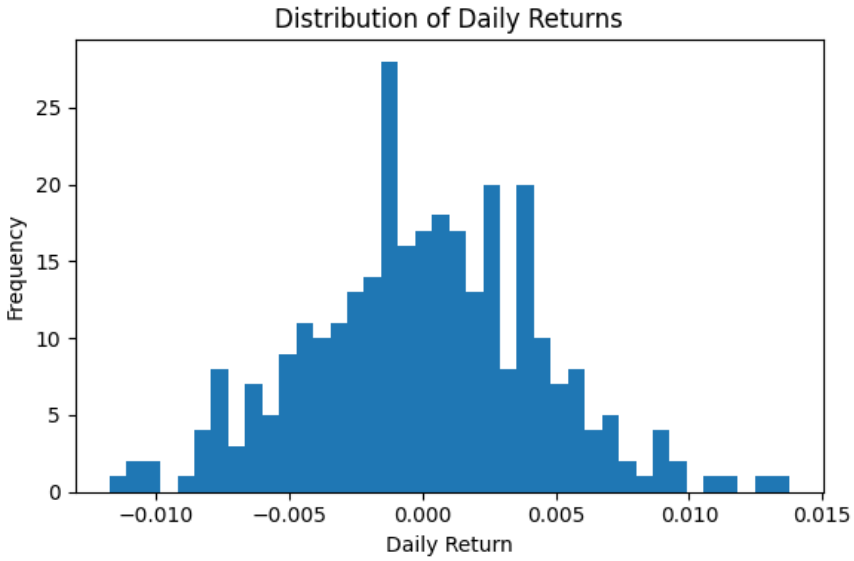

Figure 3: Distribution of daily returns.

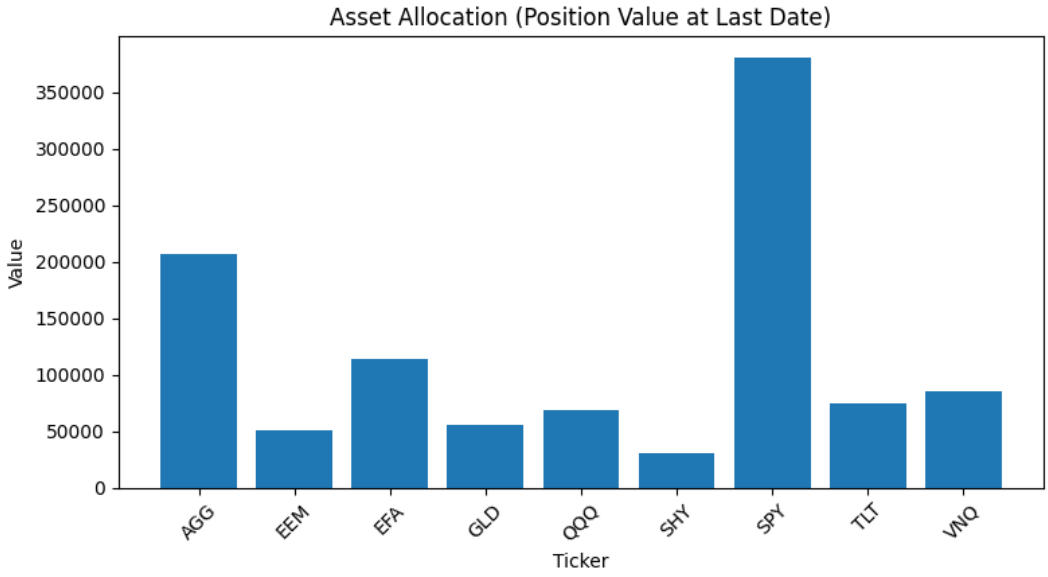

Figure 4: Asset allocation by position value on the last date.

7.1 Risk Metrics Table

To keep the report readable on standard paper sizes, the risk summary is scaled to the full text width. The values correspond to the combined horizon used elsewhere in the analysis.

Portfolio	As of	Days	Beta	Intercept	Sharpe	VaR_{95} (hist)	VaR_{99} (hist)	VaR_{95} (param)	VaR_{99} (param)	Mean	Std
Well Rounded Portfolio	2026-01-26	305	0.363598	0.000045	-0.633985	0.007631	0.010432	-0.007226	-0.010209	0.000025	0.004378

Table 1: Risk metrics calculated from simulated daily returns (past ∼1Y plus six months forward). VaR values are daily magnitudes; the parametric version assumes normality.

8 Interpretation

The portfolio exhibits a beta near 0.36 with respect to SPY. In practical terms, if the broad market were to fall one percent on a given day, the portfolio would, on average, move about a third as much in the same direction, all else equal. That damped sensitivity is consistent with the presence of high–quality fixed income and defensive exposures such as gold. The intercept is close to zero on a daily basis, which is typical for diversified strategies over short horizons; alpha estimates tend to be noisy at the daily cadence and stabilize over longer windows. The annualized Sharpe ratio for this specific simulation is negative, underscoring a central theme of risk management: a portfolio can be well diversified and still experience periods where the balance of mean and volatility is unfavourable. VaR at the 95% and 99% levels places intuitive bounds on one–day losses. The historical VaR reads directly from the empirical distribution and therefore adapts to any skewness or mild tail weight produced by the mixture of assets. The parametric VaR, by contrast, encodes the same information through a Gaussian lens; it is faster to compute and easy to stress under hypothetical volatilities, but it can understate risk in the presence of genuine fat tails. Reconciling the two perspectives is part of responsible reporting: when they agree, confidence rises; when they differ, the discrepancy itself is a valuable signal.

Beyond the single numbers, the figures tell a coherent story. The value chart trends upward despite interim drawdowns, which is precisely what one hopes to see in a balanced allocation. The returns series hovers around zero with bursts that are more pronounced during equity–dominated weeks, while the histogram remains compact, emphasizing that most days are uneventful. The allocation snapshot shows where the risk budget truly lives: equities supply return, bonds and cash–like instruments supply stability, and real assets offer diversification that is imperfect but meaningful. Together these elements make the case for the pipeline as a daily tracking tool: it is transparent, composable, and anchored in quantities practitioners actually watch.

9 Limitations and Improvements

Although the simulation is internally consistent, it is intentionally simple. GBM ignores volatility clustering, jumps, and structural breaks that are common in real markets; those features can be introduced in extensions via stochastic volatility, regime switching, or jump–diffusion processes. Using real historical prices would allow backtesting against known events and would make the scale of risk immediately relatable. The current framework treats the portfolio as buy–and–hold; in production one would add rebalancing rules, transaction costs, and tax lots, all of which can be expressed cleanly in Spark SQL. Finally, single–factor beta is a starting point rather than an ending point; multi–factor models and downside–sensitive measures such as Expected Shortfall (CVaR) and drawdown statistics would broaden the diagnostic reach.

10 Conclusion

This project demonstrates that a compact PySpark SQL workflow can do the heavy lifting of portfolio tracking without sacrificing clarity. By generating a realistic time line, valuing positions correctly, and presenting risk in a compact table augmented by intuitive graphics, the pipeline offers a template that is both educational and operational. It is easy to swap synthetic paths for real data, to expand the set of risk measures, or to layer in portfolio policies. In short, the approach is a practical way to keep score in markets that refuse to stand still, and a solid foundation on which richer analytics can be built.

Works Cited

Apache Software Foundation. "Spark SQL, DataFrames and Datasets Guide." *Apache Spark Documentation*. https://spark.apache.org/docs/latest/sql-programming-guide.html. Accessed 2 Aug. 2025.

Yahoo Finance. "ETF Screener." *Yahoo Finance*. https://finance.yahoo.com/screener/etf. Accessed 2 Aug. 2025.

The PostgreSQL Global Development Group. "SQL Command Reference." *PostgreSQL Documentation*. https://www.postgresql.org/docs/current/sql-commands.html. Accessed 2 Aug. 2025.