**FinMetrics — Market Performance & Risk Analytics**

*A portfolio-ready, end-to-end Data Analyst project (Python • SQL • Tableau)*

*Author: Darshan Patil*

**1) Executive Summary**

FinMetrics is an end-to-end analytics pipeline that turns raw market data into decisions you can see and defend. Over ~10 years of daily OHLCV prices for 15 large-cap U.S. tickers, the project standardizes calendars, engineers risk/return features (daily returns, moving averages, rolling volatility, RSI), and trains interpretable baseline classifiers to forecast next-day direction. Outputs flow into a polished Tableau dashboard designed for business users and hiring managers: it highlights risk–return trade-offs, cross-asset correlations, and how model probabilities evolve alongside actual prices.

The build intentionally mirrors what analysts do on the job: Python (Pandas/SQLAlchemy/Scikit-learn) for data/ML, SQLite for durable storage and ad-hoc SQL, and Tableau for communication. The repo is modular and reproducible (scripts for collect → clean → model → export), with saved artifacts and BI-ready CSVs. In short: not a Kaggle demo, a small, production-style system.

**What you get at a glance**

* **Data**: ~2,579 trading days per ticker; ~38.7k fully featured rows post-engineering.
* **Models**: pooled logistic baseline + per-ticker variants; threshold tuning to improve F1 on select names.
* **Insights**: clear clustering on risk vs return, strong sector co-movement in the correlation matrix, and intuitive probability overlays vs actual closes.
* **Deliverables**: public dashboard, packaged .twbx, model artifacts (.joblib), and exportable CSVs for BI.

**Live dashboard** (Tableau Public):

https://public.tableau.com/app/profile/darshan.patil4472/viz/FinMetrics\_Dashboard\_v1\_Stable/FinMetricsDashboard?publish=yes

**Why it matters**: FinMetrics demonstrates the full loop:- data engineering → modeling → stakeholder-ready visuals—exactly the competency stack modern Data Analyst roles expect.

**2) Project Idea & Scope**

The FinMetrics project began with a simple but industry-relevant goal to create a robust, data-driven system that quantifies, compares, and visualizes the behavior of large-cap U.S. stocks while applying a machine learning lens to forecast short-term movement trends. The underlying philosophy was not to chase perfect prediction accuracy, but to build a transparent, scalable, and explainable pipeline that reflects how professional data analysts and financial engineers operate in real-world organizations.

**Core Idea: -**

At its core, FinMetrics attempts to capture how major stocks perform and interact in terms of both return and risk. By analyzing daily OHLCV data (Open, High, Low, Close, Volume) over a decade, it seeks to understand how different tickers behave individually and collectively and how one can translate these patterns into actionable insights using interpretable machine learning.

The final layer of the system introduces a binary classifier (predicting whether a stock’s next-day close will be “up” or “down”), bridging traditional financial analysis with modern AI methodologies. The result is a self-contained system that integrates quantitative finance, data science, and visualization into one coherent workflow.

**Key Analytical Questions Addressed: -**

1. **Performance Comparison:**

Which stocks have the best average daily returns when adjusted for volatility (risk)?

→ This identifies risk–reward trade-offs similar to a Sharpe-ratio analysis.

1. **Market Relationships:**

How strongly do tickers move together?

→ The **correlation matrix** reveals sector-based dependencies and diversification potential.

1. **Model Behavior:**

What does an interpretable baseline “probability-of-up” model look like over time?

→ FinMetrics visualizes ML-generated probabilities alongside actual price movements, allowing users to see where the model aligns or diverges.

1. **Model Optimization:**

How does threshold tuning or per-ticker modeling affect forecast precision and recall?

→ It compares a global pooled model vs individual models per ticker a valuable trade-off between generalization and specialization.

**Target Audience: -**

The system was designed to serve:

* Financial Analysts & Portfolio Managers who want a clean overview of stock performance, volatility, and correlations.
* Data Analysts or Data Scientists (Entry–Mid level) who aim to demonstrate data engineering + ML + visualization in a single reproducible pipeline.
* Hiring Managers / Recruiters looking for candidates capable of building production-grade analytical workflows, not just static notebooks.

**Scope of Work: -**

FinMetrics is not just an academic project, it’s structured to mirror real industry data workflows:

* Pulling live data (from Yahoo Finance via API)
* Normalizing and cleaning it using Python and SQL
* Applying robust feature engineering (returns, MA, volatility, RSI)
* Running predictive models for next-day movement
* Exporting summarized datasets for Tableau visualization
* Publishing an interactive dashboard where end-users can explore results by ticker, model type, or time window

**3) What I have Built (End-to-End)**

FinMetrics wasn’t just built to crunch numbers it was designed to feel like a real analytics product from start to finish. Every step, from data collection to visualization, was carefully planned to mimic how data pipelines function in professional environments. The entire workflow was broken down into six clear stages (as per my *FinMetrics Roadmap document*), ensuring clean modularity, version control, and reusability.

**Stage 1: Data Ingestion**

Everything started with raw data collection. Using the yfinance API, I pulled 10 years of daily OHLCV (Open, High, Low, Close, Volume) data for 15 large-cap U.S. tickers names like AAPL, TSLA, NVDA, and MSFT.

* The data was saved both as CSV files and into a SQLite database (finmetrics.db) for persistent querying.
* I designed this ingestion pipeline so it could be easily refreshed in the future with minimal effort (e.g., to add new tickers or extend the time window).

**Stage 2: Data Cleaning & Feature Engineering**

Raw stock data is rarely analysis-ready, so I built a detailed cleaning process:

* **Calendar Alignment:** Created a business-day index and reindexed all tickers to remove gaps.
* **Missing Value Handling:** Used forward-fill for non-trading days and ensured all tickers had consistent timelines.
* **Feature Engineering:** Added analytical indicators widely used in quantitative finance:
  + Daily Returns (%)
  + Moving Averages (MA7, MA30)
  + Rolling Volatility (21-day)
  + RSI14 (Relative Strength Index)
  + Binary Target: “1” if next day’s close > today’s close, else “0”

This stage transformed messy time series into a clean, model-ready dataset, stored back into SQLite as prices\_metrics.

**Stage 3: SQL + Exploratory Data Analysis (EDA)**

Once the cleaned dataset was ready, I used SQL queries directly from Python to extract summary statistics and analytical slices:

* Record counts and coverage dates per ticker
* Average daily return and volatility calculations
* Correlation matrices of closing prices

All these summaries were exported as Tableau-ready CSVs, giving the BI layer structured, lightweight data sources optimized for fast visual rendering.

**Stage 4: Machine Learning Baselines**

This is where the fun began building interpretable predictive baselines rather than complex black boxes.

**Models implemented:**

1. **Pooled Logistic Regression (Global Model):**

A single logistic regression trained across all tickers combined. This serves as a market-level classifier that predicts next-day direction (“up” or “down”) using engineered features.

1. **Threshold Tuning (Per Ticker):**

Since probabilities are often poorly calibrated, I performed threshold optimization per ticker to maximize the F1 score on validation data.

* + Example: META performed best at threshold 0.40, TSLA at 0.45, NVDA at 0.55.

1. **Per-Ticker Logistic Models:**

I also trained individual logistic models for each stock, recognizing that each ticker has its own market dynamics and volatility patterns.

Each model was saved as a separate .joblib file under models/artifacts/per\_ticker/.

This ML layer showed that even basic models, when engineered properly, can extract valuable structure from market data.

**Stage 5: Business Intelligence (Tableau Layer)**

After the modeling was complete, I shifted to data storytelling — converting results into an interactive, visual experience using Tableau.

The dashboard includes:

* **Risk vs Return (Scatter Plot):** Average daily return vs volatility, colored by ticker.
* **Coverage Chart:** Number of records per ticker with hover tooltips showing coverage period.
* **Correlation Heatmap:** Closing-price correlations across tickers, centered around 0.
* **Forecast vs Actual:** Dual-axis view of predicted probabilities (area) vs closing prices (line), filterable by ticker and model source (tuned/per-ticker).

These visuals transformed technical model outputs into insights non-technical stakeholders can instantly grasp.

**Stage 6: Artifacts, Documentation & Deliverables**

Finally, I made sure the project looked and behaved like something a real analytics team would maintain:

* All trained models saved (.joblib) for reproducibility.
* Predictions & thresholds stored in /data/processed/ and /models/forecasts/.
* Tableau exports (risk summaries, correlations, forecasts) for public BI integration.
* Packaged workbook (.twbx) with embedded data for offline viewing.
* Screenshots & Tableau Public link included in documentation for recruiter and professor evaluation.

**4) Programming Languages / Tech Stack**

FinMetrics was intentionally built using a modern, production-style analytics stack, balancing performance, readability, and deployability. Every component from data ingestion to visualization was selected to reflect what’s used in real-world data science teams.

**Python 3.13 — Core Engine (Data, Features, ML, Exports)**

Python served as the foundation of the project. Every major phase data collection, cleaning, feature engineering, machine learning, and exporting outputs was implemented in Python.

**Key Libraries Used:**

* **pandas / numpy:** For data wrangling, transformation, and matrix computations.
* **ta (Technical Analysis):** To compute financial indicators like RSI, moving averages, and volatility windows.
* **scikit-learn:** The backbone for building, training, and evaluating machine learning models (logistic regression baselines, per-ticker classifiers, F1 optimization).
* **statsmodels:** For basic econometric checks and regression diagnostics (used in experimentation).
* **SQLAlchemy:** For database connectivity between Python scripts and the SQLite backend, ensuring clean read/write transactions.
* **yfinance:** For direct API-based stock data ingestion (10 years of daily OHLCV data).
* **matplotlib / seaborn:** For exploratory visualizations and chart exports prior to Tableau integration.
* **xgboost (installed for future versions):** Prepared for potential model upgrades—ready to replace linear baselines with boosted-tree classifiers in later versions.

**SQL (SQLite) — Persistent Storage & Ad-hoc Analysis**

A lightweight **SQLite database (finmetrics.db)** was used to store and manage all intermediate and processed data.

* **Tables Included:**
  + raw\_prices — 10-year OHLCV data for all tickers
  + prices\_metrics — cleaned, feature-engineered dataset with technical indicators
  + predictions, predictions\_tuned, prices\_forecasts — ML outputs for Tableau

SQLite offered the perfect balance between simplicity and analytical power: no server setup, yet full SQL capabilities for querying, filtering, and joining.

**Tableau Public — Interactive Visualization & Storytelling**

All final deliverables were synthesized into a Tableau dashboard that tells the complete story from stock risk/return comparisons to machine learning forecast performance.

Key dashboards included:

* **Risk vs Return Scatter Plot:** Plots volatility against average daily return by ticker.
* **Coverage Chart:** Visualizes dataset length and coverage for each stock.
* **Correlation Heatmap:** Shows pairwise closing-price correlations (−1 to +1).
* **Forecast vs Actual:** Dual-axis view of price movement vs predicted probability, filterable by ticker and model source (tuned/per-ticker).

**Summary of the Stack**

|  |  |  |
| --- | --- | --- |
| Layer | Technology | Purpose |
| Data Collection | yfinance, pandas | Fetch and preprocess OHLCV data |
| Storage | SQLite (via SQLAlchemy) | Central, queryable data store |
| Processing | pandas, numpy, ta | Cleaning and feature engineering |
| Modeling | scikit-learn, statsmodels | Logistic regression baselines |
| Tuning & Evaluation | pandas, numpy | Threshold optimization per ticker |
| Visualization | matplotlib, seaborn, Tableau Public | Static and interactive visualization |
| Exports & Reporting | pandas.to\_csv(), Tableau | BI integration, public dashboard |

**5) What I Achieved (Concrete Outputs)**

**Data pipeline**

* Pulled ~10 years of daily OHLCV for 15 tickers (AAPL, MSFT, NVDA, AMZN, GOOGL, META, TSLA, JPM, V, UNH, XOM, PG, KO, PEP, HD).
* Raw long table: 565,650 rows × 7 cols (15 tickers × ~37.7k dates).
* Business-day aligned tidy table: ~39,135 rows (≈ 15 × 2,609 days).
* Feature-complete dataset after warm-up drops: 38,685 rows × 13 cols.

**Modeling**

* Pooled Logistic Regression trained on engineered features + lags.  
  Test window: 2023-11-09 → 2025-10-24 (n=7,680 rows)
  + Accuracy 0.847, Precision 0.035, Recall 0.122, F1 0.055, ROC-AUC 0.506  
    (Imbalanced classes → good teaching point about baselines.)
* Threshold tuning (per ticker) produced better F1 on several names; examples from held-out evaluation:
  + META F1 0.137 @ threshold 0.40
  + TSLA F1 0.113 @ 0.45
  + NVDA F1 0.111 @ 0.55
* Per-ticker models (logistic) showed heterogeneous behavior; sample test metrics reported per ticker (e.g., PG F1 0.125, TSLA F1 0.061).

**BI / Deliverables**

* Exported Tableau-ready CSVs:
  + risk\_summary.csv (avg daily return %, avg vol %)
  + ticker\_summary.csv (coverage by rows/start/end)
  + correlation\_matrix.csv (and pivoted to long inside Tableau)
  + predictions\_tuned.csv, predictions\_per\_ticker.csv
  + predictions\_combined.csv (15,368 rows; tuned + per\_ticker sources)
* Public dashboard published (link above)
* Packaged workbook .twbx for submission

**6) The Machine Learning Engine**

The FinMetrics ML Engine was purpose-built to be transparent, interpretable, and recruiter-facing a compact example of how to construct, tune, and explain predictive models in financial analytics. The design intentionally avoids overcomplicated architectures (like LSTMs or deep nets) in favor of clarity, reproducibility, and explainability, which are essential for data roles in finance, analytics, and risk engineering.

**Label Definition — Target Construction**

The prediction target is simple yet powerful:  
target\_up\_next\_day = 1 if Close(t+1) > Close(t), else 0  
This ensures:

* No data leakage, since only past and present features are used to predict the next day’s movement.
* A clearly binary, event-based framework ideal for logistic regression.
* Real-world interpretability: predicting “will the price close higher tomorrow?”

**Feature Engineering**

Each observation (stock × date) is enriched with technical and statistical indicators widely used in trading analytics:

|  |  |  |
| --- | --- | --- |
| Category | Features | Description |
| Returns | Daily return %, MA7, MA30 | Short and medium moving averages capturing short-term trend strength |
| Volatility | 21-day rolling volatility | Captures market risk through standard deviation of returns |
| Momentum | RSI14 (Relative Strength Index) | Measures overbought/oversold momentum using a 14-day window |
| Lagged Signals | Lag1, Lag2, Lag3 for each above | Introduces short-term memory — helping models detect recent direction shifts |

All these were computed using pandas + ta library, ensuring consistent definitions and window alignment. The inclusion of lag features made the dataset richer and more temporal-aware, giving the model a “lookback memory” similar to how human traders assess recent performance.

**Modeling Framework**

**1️ Pooled Logistic Regression (Global Baseline)**

* Trained on all tickers combined.
* Learns general market-wide dynamics.
* Ideal for capturing broad relationships between volatility, momentum, and next-day outcomes.
* Outputs: prob\_up (predicted probability), pred\_up (binary decision).

*Why Logistic Regression?*  
It’s interpretable, fast, and easy to visualize. Coefficients directly indicate which features drive bullish vs. bearish probabilities.

**2️ Per-Ticker Logistic Models (Specialized Forecasting)**

Recognizing that AAPL ≠ TSLA ≠ XOM in behavior, FinMetrics trains one mini-model per stock — 15 in total.  
Each learns ticker-specific dynamics, such as:

* TSLA’s volatility sensitivity
* META’s momentum-heavy swings
* PG’s stability and mean reversion

Each model was saved under:

models/artifacts/per\_ticker/

and generated its own predictions file under:

models/forecasts/

This design allowed for cleaner experimentation and ensemble potential in future versions.

**3️ Threshold Sweep Optimization (Tuned Decision Boundaries)**

Rather than blindly assuming the classic 0.5 cut-off, FinMetrics runs a threshold sweep from 0.10 → 0.90 for each ticker.  
For each threshold:

* Compute Precision, Recall, F1, ROC-AUC
* Pick the F1-optimal threshold per ticker

Example results:

|  |  |  |  |
| --- | --- | --- | --- |
| Ticker | Threshold | F1 | ROC-AUC |
| META | 0.40 | 0.137 | 0.577 |
| TSLA | 0.45 | 0.113 | 0.575 |
| NVDA | 0.55 | 0.111 | 0.614 |
| PG | 0.55 | 0.125 | 0.576 |

**7) Data Sources & Ingestion**

FinMetrics uses a clean, automated data ingestion workflow built around Yahoo Finance’s public API and organized Python ETL scripts to ensure traceability and reproducibility at every step.

**Primary Data Source**

* **Yahoo Finance (yfinance API):**  
  Collected 10 years of daily OHLCV (Open, High, Low, Close, Volume) data for major U.S. large-cap stocks such as AAPL, TSLA, NVDA, META, MSFT, and others.  
  This ensured broad market coverage with consistent data granularity.

**Storage Architecture**

Data is stored in two synchronized formats for flexibility:

* **Raw CSV:** data/raw/all\_stocks\_raw.csv human-readable backup for inspection.
* **SQLite Database:** finmetrics.db — serves as the main analytical backend with tables:
  + raw\_prices → Original fetched prices
  + prices\_metrics → Cleaned + feature-engineered version

**ETL (Extract–Transform–Load) Scripts**

Each processing step is fully modularized into independent Python scripts:

|  |  |
| --- | --- |
| Script | Purpose |
| scripts/collect\_data.py | Fetches and merges all stock data, saves to CSV + SQLite |
| scripts/clean\_features.py | Transforms wide-format data to long-form, aligns trading calendars, and adds engineered features |
| scripts/train\_baseline.py | Trains pooled logistic regression model |
| scripts/train\_per\_ticker.py | Trains individual per-ticker models |
| scripts/tune\_threshold.py | Runs threshold optimization (0.1–0.9) to maximize F1 per ticker |

**8) Quantitative Highlights (numbers to quote)**

* 15 tickers, ~10 years, ~2,579 trading days per ticker after alignment
* 38,685 fully-featured rows for modeling
* 7,680 test rows (most recent ~2 years)
* Baseline pooled classifier: AUC ≈ 0.506 (as expected for a humble baseline)
* F1 improved for several tickers after threshold tuning (e.g., META F1 0.137 at thr 0.40)
* Combined predictions sheet for Tableau: 15,368 rows across sources

**9) Trends, Patterns & Results (from the dashboard)**

* **Risk–Return**: Mega-cap tech names cluster toward higher average returns with moderate-to-higher volatility; defensives (PG/KO/PEP) show lower returns and lower vol.
* **Coverage**: Equal record counts confirm a clean, continuous time series per ticker (good pipeline hygiene).
* **Correlation Heatmap**: Strong positive correlations within large-cap growth; useful for diversification discussion financials/energy often less correlated with pure tech.
* **Forecast vs Actual**: Probability overlays track momentum regimes reasonably; threshold tuning helps convert probability into decent directional flags for selected names.

**10) Tools Used**

* **Python**: Pandas, NumPy, TA, Scikit-learn, Statsmodels, SQLAlchemy, Matplotlib/Seaborn
* **Database**: SQLite (via SQLAlchemy)
* **Data source**: Yahoo Finance (yfinance)
* **Visualization**: Tableau Public (+ packaged .twbx)
* **Dev**: macOS (Apple Silicon), VS Code, venv, pip

**11) Repo Structure & Reproducibility**

**Run locally (condensed):**

python -m venv .venv && source .venv/bin/activate

pip install -r requirements.txt

python -m scripts.collect\_data

python -m scripts.clean\_features

python -m scripts.train\_baseline

python -m scripts.tune\_threshold

python -m scripts.train\_per\_ticker

# Tableau: connect to tableau/exports/\*.csv and build dashboard

**12) Appendix — Key Files for Reviewers**

* **Packaged workbook**: FinMetrics\_Final\_Dashboard.twbx
* **Predictions (for Business Intelligence)**:
  + data/processed/predictions\_tuned.csv
  + models/forecasts/predictions\_per\_ticker.csv
  + data/processed/predictions\_combined.csv
* **Thresholds**: data/processed/thresholds\_by\_ticker.csv
* **Model artifact**: models/artifacts/logreg\_baseline.joblib

**13) Conclusion**

This project shows you can design, build, and communicate a full analytics solution with clean engineering and business-grade visuals. It’s intentionally honest about limits, and thoughtfully staged so a team can extend it in the real world.