Macro-Informed Systematic FX Strategy Project

Deborah Akintoye

Inception Date: 21-November-2025

Project Overview

Motivation

Foreign exchange (FX) markets are the largest and most liquid financial markets globally, with daily turnover exceeding $6 trillion. Despite their liquidity, FX markets are influenced by a wide variety of economic, political, and financial factors, including interest rate differentials, macroeconomic releases, risk sentiment, and commodity price movements. A systematic FX strategy that incorporates macroeconomic information allows a quant researcher to model both predictable market patterns (e.g., carry effects driven by interest rate differentials) and conditional responses to macro shocks (e.g., risk-on/risk-off events reflected in volatility indices). By combining multiple signals in a disciplined, reproducible framework, a researcher can produce strategies that are robust and interpretable, while allowing for quantitative assessment of risk and performance.

Objective of the Project

The objective of this project is to simulate a realistic quant research pipeline in an FX trading desk. The project integrates multiple disciplines:

1. **Data engineering:** Acquire, clean, and merge FX, rates, and macroeconomic datasets into a consistent panel suitable for quantitative modelling.
2. **Feature engineering:** Compute returns, carry, momentum, and macro-informed signals. Apply scaling and normalisation techniques to produce combined signal scores.
3. **Strategy design:** Construct a systematic FX trading strategy based on weighted combination of signals and implement position management using object-oriented design.
4. **Backtesting and performance analysis:** Simulate trading over historical data, measure performance metrics, and analyse attribution at the signal and currency level.
5. **Risk analytics:** Evaluate portfolio risk using historical VaR, expected shortfall, drawdowns, and Monte Carlo simulations. Apply stress tests to assess robustness under extreme scenarios.
6. **Documentation and reporting:** Produce reproducible notebooks, visualisations, and a final report that summarises methodology, results, and insights.

Quantitative Finance Concepts

This project draws upon core quantitative finance principles, including:

- **Returns and volatility:** Measurement of FX price changes using simple or log returns, and estimation of rolling volatility as a risk measure.
- **Covariance and correlation:** Computation of interdependencies between currency pairs and macro variables to inform portfolio construction and Monte Carlo simulations.
- **Carry and forward pricing:** Application of covered interest parity (CIP) to calculate forward prices and derive carry signals that exploit interest rate differentials.

- **Momentum:** Use of rolling price changes over defined lookback periods to capture trend-following behaviour.
- **Risk-adjusted performance metrics:** Calculation of Sharpe and Sortino ratios to evaluate risk-return trade-offs.

Role of Macroeconomic Signals

Macroeconomic indicators provide critical contextual information that complements market-based signals:

- **Yield curve slope:** Captures market expectations of future interest rates and economic growth.
- **Equity indices:** Provide a proxy for global growth and risk appetite, which can affect currency valuations.
- **Commodity prices:** Certain FX pairs are sensitive to commodity shocks (e.g., AUDUSD and oil).
- **Risk sentiment measures:** Volatility indices such as VIX or MOVE provide insight into market stress conditions.

By incorporating these macro factors, the strategy can adjust exposures dynamically in response to global financial conditions, reducing vulnerability to regime shifts and market shocks.

Python as the Implementation Language

Python is chosen due to its combination of readability, flexibility, and extensive quantitative finance ecosystem. Key advantages include:

- Rich numerical libraries (NumPy, SciPy) for vectorised computations.
- Pandas for time series data manipulation and alignment.
- Matplotlib and Seaborn for visualisation of PnL, signals, and risk metrics.
- Object-oriented design capabilities for structuring positions, portfolios, and backtesting engines.
- Reproducibility and ease of integrating with Git for version control.

Project Methodology

The project follows a modular, multi-stage methodology:

1. Data Acquisition and Validation: Collect FX spot rates, interest rates, and macroeconomic variables. Apply validation checks for missing data, duplicates, and alignment inconsistencies. Outputs a clean dataset for downstream modelling.

2. Feature Engineering: Compute key features:

- **Returns:** Simple returns $R_t = \frac{S_t - S_{t-1}}{S_{t-1}}$ or log returns $r_t = \ln(S_t/S_{t-1})$.
- **Volatility:** Rolling standard deviation over a defined window N.
- **Covariance:** Historical or exponentially weighted covariance matrices for risk assessment and Monte Carlo simulation.
- **Carry:** Forward pricing using CIP and derived carry signals.
- **Momentum:** Difference of current price and price n days prior.
- **Macro signals:** Rate differentials, equity growth rates, and volatility z-scores.

3. Signal Normalisation: Apply z-score or ranking to ensure comparability of features across currencies and macro variables. Combine signals into a single weighted score:

$$\text{score}_t = w_1 \cdot \text{carry}_t + w_2 \cdot \text{macro}_t + w_3 \cdot \text{momentum}_t$$

4. Strategy Construction: Define long/short rules based on signal thresholds. Implement `Position` and `Portfolio` classes with methods for PnL tracking, NAV computation, transaction cost application, and position sizing.

5. Backtesting: Simulate strategy execution over historical data. Compute daily returns, cumulative NAV, drawdowns, and performance metrics (Sharpe, Sortino). Ensure separation of signal computation and backtesting logic.

6. Risk Analytics: Compute historical VaR and ES:

$$\text{VaR}_\alpha = -q_\alpha(R), \quad \text{ES}_\alpha = -E[R|R \leq q_\alpha(R)]$$

Perform Monte Carlo simulations of correlated returns using covariance matrices. Apply stress scenarios to evaluate resilience to macro shocks.

7. Performance Attribution: Decompose returns and PnL contributions by signal type and by currency pair. Visualise results using plots to communicate insights clearly.

8. Documentation and Reporting: Produce notebooks documenting data exploration, feature engineering, backtesting, risk analysis, and final results. Prepare a written report summarising methodology, results, risk analysis, and next steps.

Project Folder Structure and File Descriptors

The following folder structure and file descriptions should be implemented. For each Python file, we provide an explanation of its role and key functionality.

Top-level Files

- **README.md**: Overview of the project, instructions to run modules and notebooks.
- **requirements.txt**: List of all Python dependencies (numpy, pandas, matplotlib, scipy, etc.).
- **notation.txt**: Document explaining all symbols and formulas used in the project (e.g., S_t = spot rate, r_t = interest rate).

Data Directory

- **data/raw/placeholder.txt**: Placeholder for raw FX, interest rate, and macroeconomic datasets.
- **data/processed/placeholder.txt**: Placeholder for cleaned and aligned panel dataset.
- **data/synthetic/placeholder.txt**: Placeholder for optional synthetic datasets used for testing.

Source Code Directory: src/

Data Modules

- **load_data.py**: Loads raw CSVs, validates them, ensures consistent date indices, and outputs processed CSVs.
- **preprocess.py**: Cleans data (removes duplicates, handles missing values), aligns multiple datasets, merges FX, interest rates, and macro data.
- **synthetic_fx.py**: Optional module to generate synthetic FX rates, interest rates, and macro variables using stochastic simulations.

Features Modules

- **returns.py**: Compute simple returns, log returns, rolling volatility, and covariance matrices. **Formulas:**

$$R_t = \frac{S_t - S_{t-1}}{S_{t-1}}, \quad r_t = \ln \frac{S_t}{S_{t-1}}$$

$$\sigma_t = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(r_{t-i} - \bar{r}_t)^2}, \quad \mathrm{Cov}(r_i, r_j) = \frac{1}{N-1}\sum(r_i - \bar{r}_i)(r_j - \bar{r}_j)$$

- **carry.py**: Computes forward prices using Covered Interest Parity (CIP) and derives carry signal:

$$F_{t,T} = S_t \frac{1 + r_t^{dom}}{1 + r_t^{for}}, \quad \mathrm{carry}_t = \frac{F_{t,T} - S_t}{S_t}$$

- **momentum.py**: Computes rolling momentum over 3M/6M windows:

$$\mathrm{momentum}_t = S_t - S_{t-n}$$

- **macro_signals.py**: Constructs macroeconomic signals (e.g., rate differentials, VIX, yield curve slope, commodity proxies) and normalises them.
- **scaling.py**: Z-score, rank, or min-max normalisation utilities.
- **feature_set.py**: Combines carry, momentum, and macro features into a final weighted signal for strategy input.

Models Modules

- **signal_model.py**: Class to combine signals into a weighted score.

$$\text{score}_t = w_1 \cdot \text{carry}_t + w_2 \cdot \text{macro}_t + w_3 \cdot \text{momentum}_t$$

- **execution.py**: Handles execution price, slippage, and transaction cost modelling.
- **portfolio.py**: Contains `Position` and `Portfolio` classes; handles PnL, NAV, position sizing, and transaction costs.

Backtest Modules

- **base_engine.py**: Parent class for backtesting engines; defines abstract methods for simulation loop.
- **daily_engine.py**: Implements daily iteration over signals and executes trades.
- **performance.py**: Calculates Sharpe, Sortino, drawdowns, cumulative returns.
- **attribution.py**: Computes signal-level and currency-level return attribution.

Risk Modules

- **var_es.py**: Computes rolling historical Value-at-Risk (VaR) and Expected Shortfall (ES).
- **stress_tests.py**: Applies macro shocks, FX shocks, and rate shocks to evaluate strategy resilience.
- **mc_simulator.py**: Generates Monte Carlo correlated return scenarios using historical covariance.
- **covariance.py**: Implements robust covariance estimation including EWMA.

Utils Modules

- **logger.py**: Standardises logging across modules.
- **timer.py**: Decorator for measuring function execution time.
- **plotting.py**: Utilities for plotting NAV, PnL, signal trajectories.
- **validators.py**: Type and data validation checks with custom exceptions.
- **config.py**: Defines global constants, file paths, and configuration parameters.

Experiments Modules

- **run_full_strategy.py**: Orchestrates the full pipeline: data loading, feature computation, backtesting, performance output.
- **run_sensitivity_tests.py**: Varies signal weights and time horizons to test strategy robustness.
- **run_stress_analysis.py**: Runs stress tests and Monte Carlo simulations to evaluate risk.

Notebooks

- 01_data_exploration.ipynb: Explore FX, macro, and rate data; visualise distributions, missing data.
- 02_feature_engineering.ipynb: Build and test carry, momentum, macro signals; normalise and combine features.
- 03_backtesting_engine.ipynb: Apply backtest engine to generate PnL and returns.
- 04_risk_analysis.ipynb: Compute VaR, ES, drawdowns, stress test results.
- 05_final_results.ipynb: Summarise overall performance, visualisations, and attribution tables.

Reports

- **reports/figures/**: Store charts, plots, and visual output.
- **final_report.pdf**: Write-up of strategy motivation, methodology, backtest results, risk analysis, and conclusions.

Tests

- **test_returns.py**: Unit tests for returns, volatility, and covariance calculations.
- **test_carry.py**: Validate forward pricing and carry signals.
- **test_portfolio.py**: Test PnL calculations, NAV tracking, and transaction cost application.
- **test_backtest.py**: Test backtest engine loop and position execution.
- **test_risk.py**: Test VaR, ES, and stress scenario computations.

Part I: Data Engineering

1.1 Data Acquisition

You should acquire or synthetically generate:

- Daily FX spot rates for six major currency pairs (EURUSD, GBPUSD, USDJPY, AUDUSD, USDCAD, USDCHF)
- 3-month short-term interest rates
- Macro variables: yield curve slope, equity indices, commodity proxies, VIX or similar risk sentiment

1.2 Cleaning and Alignment

- Remove duplicates and handle missing data via forward-fill or interpolation.
- Align all datasets on a shared daily index.
- Validate with custom error handling.

Part II: Feature Construction

2.1 Returns

Formulas:

$$R_t = \frac{S_t - S_{t-1}}{S_{t-1}}, \quad r_t = \ln \frac{S_t}{S_{t-1}}$$

$$\sigma_t = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (r_{t-i} - \bar{r}_t)^2}$$

2.2 Carry

$$F_{t,T} = S_t \frac{1 + r_t^{dom}}{1 + r_t^{for}}, \quad \text{carry}_t = \frac{F_{t,T} - S_t}{S_t}$$

2.3 Momentum

$$\text{momentum}_t = S_t - S_{t-n}, \quad n = 63(\text{3M}), 126(\text{6M})$$

2.4 Macro Signals

Examples:

- Rate differential: $r^{dom} - r^{for}$
- Equity growth: $\Delta I_t / I_{t-1}$
- Risk sentiment: VIX z-score

Part III: Strategy Design and Portfolio Construction

3.1 Signal Combination

Weighted combination:

$$\text{score}_t = w_1 \cdot \text{carry}_t + w_2 \cdot \text{macro}_t + w_3 \cdot \text{momentum}_t$$

3.2 Position and Portfolio

- `Position` class: open(), close(), update_pnl()
- `Portfolio` class: size positions, mark-to-market, transaction costs, NAV tracking

Part IV: Backtesting

- Daily loop over signals, open/close positions.
- Compute NAV, PnL, cumulative returns.
- Performance metrics: Sharpe $\frac{\bar{r}}{\sigma_r}$, Sortino $\frac{\bar{r}}{\sigma_{down}}$, max drawdown

Part V: Risk & Stress Testing

- Rolling historical VaR and ES:

$$\text{VaR}_\alpha = -q_\alpha(R), \quad \text{ES}_\alpha = -E[R | R \leq q_\alpha(R)]$$

- Monte Carlo simulations of correlated returns:

$$R_{MC} \sim \mathcal{N}(\mu, \Sigma)$$

- Macro and FX shocks to evaluate strategy resilience.

Part VI: Performance Attribution

- Decompose PnL by signal: carry vs momentum vs macro.
- Currency-level attribution: which pairs contribute most to returns and risk.

Part VII: Deliverables and Evaluation

- **Code:** Modular Python package with docstrings, inheritance, unit tests.
- **Notebooks:** Data exploration, feature engineering, backtesting, risk analysis, final results.
- **Report:** Written summary with motivation, methodology, results, risk analysis, and conclusions.
- **Evaluation:** Python elegance, quantitative accuracy, clarity of rationale, visualisation quality.

Conclusion and Next Steps

Summary of Achievements

This project demonstrates a complete end-to-end pipeline for a macro-informed systematic FX strategy. Key achievements include:

- Construction of a clean, modular Python codebase with classes, inheritance, and well-documented functions suitable for research and production environments.
- Acquisition, cleaning, and alignment of multi-source datasets including FX rates, interest rates, and macroeconomic indicators.
- Computation of a variety of signals (returns, volatility, carry, momentum, macro) and combination into a unified trading signal.
- Implementation of a backtesting framework capable of simulating daily trades, tracking PnL, NAV, and risk-adjusted performance metrics.
- Risk analysis via historical VaR/ES, Monte Carlo simulations, and stress tests to evaluate robustness under extreme market conditions.
- Performance attribution at both signal-level and currency-level to provide actionable insights.

Insights Gained

Through this project, one gains:

- Practical understanding of how macroeconomic factors interact with FX markets.
- Experience in object-oriented programming for quantitative finance applications.
- Appreciation for the importance of risk management, including scenario analysis and drawdown evaluation.
- Awareness of the challenges of data quality, alignment, and validation in real-world trading research.
- Familiarity with constructing and interpreting quantitative performance metrics, such as Sharpe, Sortino, and max drawdown.

Limitations and Considerations

While the project aims to be as realistic as possible, several limitations are acknowledged:

- Historical data may not capture all future market regimes or shocks.
- Transaction costs and slippage are modelled, but real execution conditions can differ.
- Macro signals are simplified proxies; in practice, richer datasets and alternative specifications may improve performance.
- Monte Carlo simulations assume normally distributed returns, which may understate tail risks in turbulent markets.

Next Steps and Extensions

Potential extensions for further research include:

- Incorporating more sophisticated machine learning models to enhance signal prediction.
- Adding regime-switching logic to adapt signal weights dynamically in different market conditions.
- Expanding the universe of currency pairs, including emerging market FX.
- Integrating options or other derivatives to hedge or enhance FX exposures.
- Conducting out-of-sample validation using live or pseudo-live trading simulations.

Final Remarks

This flagship project serves as a comprehensive showcase of quantitative finance and Python programming skills. It demonstrates not only technical competence but also the ability to structure, document, and analyse a full research workflow akin to that of a professional FX trading desk. The project provides a foundation for further research and practical application in systematic trading, risk management, and portfolio construction.