Polymarket → TradFi Regression Engine

Standalone local research pipeline · Python 3.10+ · March 2026

What This Engine Does

This pipeline discovers active prediction markets on Polymarket, maps them to tradable financial assets (SPY, BTC-USD, etc.), and statistically tests whether prediction-market sentiment Granger-causes short-term price movements in traditional markets. It produces a scored research library with tiered confidence levels.

Pipeline Overview

Phase	What Happens	Key Output
1 — Discovery	Fetch active markets from Polymarket Gamma API; apply tag, keyword, and volume filters	Filtered market list in SQLite
2 — Mapping	Sentence-embedding dedup + LLM maps markets to Yahoo Finance tickers	Ticker mappings (cached)
3 — Ingestion	Pull 5-min OHLCV bars (yfinance) + CLOB token price history	Raw time-series data
4 — Features	Align grids via merge_asof, logit-transform probabilities, add lags & dummies, ADF/KPSS	Feature matrix per ticker
5 — Modeling	LASSO CV (TimeSeriesSplit) → post-LASSO OLS (HAC errors) → Granger → walk-forward OOS	Betas, p-values, IC, Sharpe
6 — Export	Persist runs to SQLite, export JSON and styled Excel with status tiers	Research library + exports

Status Tiers

Status	Criteria
Null	No non-zero LASSO coefficients or all p-values above threshold
Candidate	At least one beta with $p < 0.10$
Verified	Candidate + Granger $p < 0.05$ + OOS IC > 0.05 + OOS Sharpe > 0.5 + $n \geq 500$
Degraded	Previously Verified but current run no longer meets thresholds

Setup Instructions

Prerequisites

Python 3.10 or later, pip, and an OpenAI API key (for LLM ticker mapping in Phase 2).

Step 1 — Create a virtual environment

```
cd pm-regression-engine
python -m venv .venv
source .venv/bin/activate   # macOS / Linux
.venv\Scripts\activate     # Windows
```

Step 2 — Install dependencies

```
pip install -r requirements.txt
```

Step 3 — Configure secrets

```
cp .env.example .env
# Edit .env and add: OPENAI_API_KEY="sk-proj-..."
```

Step 4 — Review config.yaml (optional)

All defaults are sensible. Key settings to tweak: tradfi.lookback_days (data window, max 59), llm.dry_run (preview mappings without calling the LLM), and general.debug_save_intermediates (save CSVs at each phase).

Running the Engine

Initialize the database

```
python -m src.cli init-db
```

Run step by step (recommended first time)

```
python -m src.cli run-phase --phase 1   # Discover & filter markets
python -m src.cli run-phase --phase 2   # LLM ticker mapping
python -m src.cli run-pipeline-all     # Phases 3–6 for all tickers
```

Or run a quick smoke test

```
python -m src.cli run-pipeline-all --smoke-test
```

Smoke-test mode limits to 50 markets, 3-day lookback, and relaxed thresholds so all six phases complete in under a minute.

Single-ticker mode

```
python -m src.cli run-pipeline --ticker SPY
```

CLI Reference

Command	Description
init-db	Create or update the SQLite schema
run-phase --phase N	Run a single phase (1–6)
run-pipeline --ticker X	Full pipeline for one ticker (phases 3–6)
run-pipeline-all	Full pipeline for all mapped tickers
run-pipeline-all --smoke-test	Quick end-to-end validation
inspect --ticker X	Diagnostic report for a ticker
summary	High-level database overview

export-library	Export JSON + Excel files

Where to Find Results

What	Location
Logs	logs/engine.log (rotating, DEBUG-level)
Database	db/pm_sentiment_engine.db (any SQLite browser)
LLM cache	data/debug/llm_cache/*.json
Debug CSVs	data/raw/ and data/processed/ (when enabled)
JSON export	exports/json/verified_relationships.json
Excel export	exports/excel/library_snapshot.xlsx

Statistical Methods

LASSO with TimeSeriesSplit CV — prevents look-ahead bias; selects a sparse feature set from lagged PM logits, autoregressive return lags, and calendar dummies.

Post-LASSO OLS with HAC errors — refits OLS on LASSO-selected features using Newey-West robust standard errors for autocorrelated residuals.

Granger causality — tests whether PM logit changes Granger-cause TradFi returns at multiple lag orders.

Walk-forward OOS — trains on the first 67% of the sample, tests on the remaining 33%; reports information coefficient (IC) and a simple long/short Sharpe ratio.

Key Dependencies

requests, pandas, numpy, yfinance, scikit-learn, statsmodels, python-dotenv, pydantic, sentence-transformers, openpyxl, PyYAML, loguru, openai, click. All statistics run in Python — no Stata or R required.

Safety and Debugging

- Ctrl+C is safe — SQLite WAL mode ensures DB consistency on interruption
- LLM calls are cached per market ID — re-runs never re-query mapped markets
- Each phase logs counts (markets fetched, filtered, modeled) to console and file
- Sanity checks enforce minimum observations, no NaN targets, and no all-zero features
- The project is fully standalone — no external servers, websites, or cloud dependencies

Running Tests

```
python -m pytest tests/ -v
```

27 tests covering DB operations, filters, feature alignment, LASSO modeling, status classification, and CLI parsing.