Polymarket → TradFi Engine Functions · How-To Guide

Standalone local research pipeline · Python 3.11+ · April 2026

What This Document Covers

This guide explains how to run the engine for each supported function – the original LASSO regression pipeline plus the new correlation and regulatory-filter utilities – and how the new engine dispatcher and config toggles fit together.

It assumes you are in the project root (C:\AI\pm-regression-engine on Windows) and have already followed the setup steps from the main explainer (virtualenv, requirements, .env, config.yaml).

Functions Overview

The engine now exposes three high-level functions behind a single unified CLI entrypoint:

- LASSO
  - Full 6-phase regression pipeline (Phases 1–6 in main.py).
  - Wrapped in run_lasso_pipeline(), then dispatched via engine_dispatcher.py.
- Correlation
  - Lightweight Pearson/Spearman correlation analysis between Polymarket markets and mapped TradFi tickers.
  - Uses existing mappings and price histories; exports a styled Excel report.
- Regulatory Filter
  - Canadian/Ontario regulatory plausibility screen over all Polymarket markets.
  - Assigns risk levels and plausibility scores, and exports a styled Excel report.

All three functions share the same CLI surface:

```
python -m src.cli run --function <lasso|correlation|regulatory-filter> [options]
```

The active default can be set in config.yaml under functions.active_mode.

Pre-requisites (Once Per Machine)

You only need to do these once on a new machine; after that you can jump straight to the function-specific sections.

1. Create and activate a virtual environment

```
cd C:\AI\pm-regression-engine
python -m venv .venv
.venv\Scripts\activate      # Windows
```

```
# source .venv/bin/activate   # macOS / Linux
```

2. Install dependencies

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

3. Configure your OpenAI key (required for LLM mapping in Phase 2 and for any future LLM usage)

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

4. Review config.yaml (optional but recommended)
    - Confirm polymarket, tradfi, llm, and verification settings.
    - New: functions section with:
        - active_mode: default function for python -m src.cli run.
        - correlation_config: knobs for correlation engine behaviour.
        - regulatory_filter_config: knobs for the regulatory filter.
5. Initialize or reset the database

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

After this, you can use any of the functions described below.

LASSO Function (Full Regression Pipeline)

The LASSO function is the original 6-phase Polymarket → TradFi regression engine, now wrapped in run_lasso_pipeline() and exposed through the new run --function lasso command.
This function is not guaranteed and requires changes to work as designed.

What It Does

- Runs Phase 1–6 exactly as in the original engine:
    - Phase 1: Discover and filter active Polymarket markets.
    - Phase 2: Deduplicate and map markets to TradFi tickers via LLM.
    - Phase 3: Ingest 5-min yfinance OHLCV and Polymarket token prices.
    - Phase 4: Align and transform features (logits, lags, dummies, stationarity checks).
    - Phase 5: LASSO CV → post-LASSO OLS (HAC) → Granger → walk-forward OOS.
    - Phase 6: Persist runs and export JSON + styled Excel library snapshot.

- Classify each ticker/market grouping into Null, Candidate, Verified, or Degraded based on config.yaml thresholds.

How To Run (LASSO)

Run the full pipeline for all mapped tickers:

```
# All tickers (full pipeline)
python -m src.cli run --function lasso
```

Run the pipeline for a single ticker:

```
# One ticker (e.g., SPY)
python -m src.cli run --function lasso --ticker SPY
```

Run a quick smoke test:

```
# Smoke test (small sample, fast runtime)
python -m src.cli run --function lasso --smoke-test
```

Behind the scenes, engine_dispatcher.py will:

- Inspect the --function flag (or functions.active_mode if omitted).
- Route to run_lasso_pipeline() in main.py.
- run_lasso_pipeline() will orchestrate Phases 1–6 exactly as in the original design.

LASSO Outputs

- SQLite
  - db/pm_sentiment_engine.db → pm_sentiment_library table updated with latest run.
- JSON
  - exports/json/verified_relationships.json (and other JSON snapshots, depending on config).
- Excel
  - exports/excel/library_snapshot.xlsx
  - Includes conditional formatting for Null, Candidate, Verified, and Degraded.
- Logs
  - logs/engine.log (rotating, DEBUG-level).

When To Use LASSO

- You want a full statistical test of whether Polymarket sentiment leads to TradFi prices.

- You care about regression coefficients, p-values, Granger causality, and OOS validation.
- You are maintaining or extending the core pm_sentiment_library.

Correlation Function (Pearson/Spearman)

The correlation function is a lighter-weight analysis that re-uses existing mappings and data to compute simple correlation statistics and export them as a formatted Excel sheet.

What It Does

- Pulls mapped Polymarket markets and their associated TradFi tickers from the SQLite database via db.get_all_markets() and related helpers.
- For each PM/TradFi pair, computes:
  - Pearson correlation between PM price series and TradFi returns (or prices, depending on config).
  - Spearman rank correlation for robustness.
- Computes p-values for each correlation statistic.
- Creates significance flags based on configurable p-value thresholds.
- Exports a styled Excel file with:
  - Ticker
  - Polymarket market ID and title
  - Pearson and Spearman correlation values
  - P-values
  - Boolean / textual flags for statistical significance
  - Green highlighting for significant correlations

How To Run (Correlation)

Run with default output path:

```
python -m src.cli run --function correlation
```

Use a custom output filename:

```
python -m src.cli run --function correlation --output my_correlations.xlsx
```

Under the hood, engine_dispatcher.py resolves --function correlation and calls into correlation_engine.py.

Correlation Outputs

- Excel (primary)
    - Default: exports/excel/correlation_results.xlsx
    - Custom: path passed via --output
- Logs
    - Summary of pair counts, number of significant correlations, and any data exclusions.

When To Use Correlation

- You want a quick sanity check on linear and rank correlations without running the full LASSO pipeline.
- You want an at-a-glance Excel view for exploratory analysis or slideware.
- You are iterating on filters/mappings and want to see whether relationships look non-random before committing to heavy modeling.

Regulatory-Filter Function (Ontario/Canada Plausibility)

The regulatory-filter function evaluates all Polymarket markets through a Canadian/Ontario regulatory lens and assigns plausibility and risk scores.

What It Does

- Fetches all markets (not just mapped ones) from the SQLite database using db.get_all_markets().
- Classifies each market into asset / product categories, such as:
    - Securities (equity-like, bond-like, ETF-like).
    - Derivatives (options, futures-style payoff, leveraged products).
    - Crypto / commodities / FX.
    - Other / exotic / non-financial.
- Detects regulatory triggers based on market structure and wording, for example:
    - Margin or leverage.
    - Options-like asymmetric payoffs.
    - Complex structured or basket products.
- Scores regulatory plausibility in the Ontario/Canada context, typically 0–100%, using a weighted rubric, e.g.:
    - Category match: 40%.
    - Ontario relevance (local issuers, Canadian macro, CAD-based products): 30%.
    - Volume credibility (is this economically meaningful size vs. noise): 20%.
    - Regulatory risk (triggers, complexity, marketing signals): 10%.

- Tags markets with risk levels (CRITICAL / HIGH / MEDIUM / LOW) and recommendation flags to aid downstream decision-making.

How To Run (Regulatory Filter)

Run with default output path:

```
python -m src.cli run --function regulatory-filter
```

Use a custom output filename:

```
python -m src.cli run --function regulatory-filter --output regulatory_assessment.xlsx
```

The dispatcher resolves regulatory-filter and calls into regulatory_filter.py, which performs the analysis and writes the Excel report.

Regulatory Filter Outputs

- Excel (primary)
  - Default: exports/excel/regulatory_assessment.xlsx
  - Custom: path passed via --output
- Each row typically includes:
  - Market ID and title.
  - Ticker (if mapped) and any detected asset categories.
  - Plausibility score (0–100%).
  - Risk level (CRITICAL / HIGH / MEDIUM / LOW).
  - Detected regulatory triggers (margin, leverage, options, etc.).
  - Recommendation flags (e.g., DO_NOT_LIST, REVIEW_REQUIRED, PLAUSIBLE).
- Logs
  - Count of markets by risk tier, any parsing/classification warnings.

When To Use Regulatory Filter

- You want a first-pass view of which markets are plausibly listable to Ontario/Canadian users.
- You need a structured, documented audit trail for why certain markets are higher risk.
- You are prototyping product/UX flows for compliant prediction-market exposure.

Engine Dispatcher and Config Toggling

The engine_dispatcher.py module centralizes routing logic so that all new analytical functions can share the same run entrypoint.

CLI vs Config Precedence

There are two ways to choose which function runs:

1. Via CLI flag (highest priority)

```
# Correlation via CLI
python -m src.cli run --function correlation
```

2. Via config.yaml

```
functions:
  active_mode: "correlation"  # Changes default for `run` when --function is omitted
```

Precedence rules:

- If --function is provided on the CLI, it always wins.
- If --function is omitted, functions.active_mode is used.
- If neither is set, the engine falls back to lasso (for backward compatibility), or whatever default you define in config_loader.py.

Legacy Commands (Still Supported)

All original CLI commands remain available and unchanged, so any existing scripts or docs continue to work:

```
python -m src.cli run-phase --phase 1
python -m src.cli run-phase --phase 2
python -m src.cli run-pipeline --ticker SPY
python -m src.cli run-pipeline-all --smoke-test
python -m src.cli export-library
python -m src.cli inspect --ticker SPY
python -m src.cli summary
```

Under the hood, these call into the same main.py and db.py functions as before. The new dispatcher only affects the unified run --function entrypoint.

Where To Find Outputs (All Functions)

Regardless of function, outputs follow the same general layout:

- Database
  - db/pm_sentiment_engine.db

- Logs
  - logs/engine.log
- Excel exports
  - LASSO library: exports/excel/library_snapshot.xlsx
  - Correlation: exports/excel/correlation_results.xlsx (or custom path)
  - Regulatory filter: exports/excel/regulatory_assessment.xlsx (or custom path)
- JSON exports (LASSO only)
  - exports/json/verified_relationships.json and related files

All Excel exports are formatted with color-coding and auto-widened columns for readability.

Safety, Debugging, and Tests

- Ctrl+C can safely interrupt any function; SQLite WAL mode protects DB integrity.
- LLM calls and expensive computations are cached or re-usable where possible, so repeated runs are incremental when underlying data is unchanged.
- Sanity checks and diagnostic logs help catch NaNs, empty datasets, and misconfigured tickers/functions before modeling.

To run the test suite (recommended when you change engine_dispatcher.py, correlation_engine.py, or regulatory_filter.py):

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

This will re-run coverage across DB operations, filters, feature alignment, LASSO modeling, status classification, CLI parsing, and any new tests you add for correlation and regulatory-filter behavior.