Process Documentation

Credit Report Insights Recommendation System

Presented by
DONG Boyuan

Date
23 September 2025

Contents

1 Methodology

This section provides a step-by-step explanation of the methodology used in the project, including data preprocessing, modeling techniques, and ranking strategies. The goal of this project is to recommend **events**, **factors**, and **variables** (EFV) for a given company and section in credit rating reports, supporting analysts to efficiently draft rating report sections.

1.1 Data Preprocessing

The data preprocessing pipeline is designed to transform raw HTML credit rating reports into a well-structured database schema that supports downstream EFV (Event, Factor, Variable) extraction and recommendation. The first two stages focus on building a reliable data foundation, while the latter stages focus on information extraction and persistence.

1.1.1 HTML Ingestion and Report Normalization

The first stage deals with raw HTML files containing credit rating reports. The main objective is to standardize the data format and store the basic metadata in the `report` table.

- **File Discovery and Parsing.** The system scans a target folder and enumerates all HTML files through `data_utils.get_all_html_files(file_path)`. Each file is parsed by a custom Fitch-specific parser, `parse_html.parse_fitch_factiva()`, which outputs a list of structured sections containing fields such as report title, headings, and body text.
- **Company Attribution.** Each parsed report is associated with a company name derived directly from the HTML filename stem (using Python's `Path.stem`). This ensures that even when internal metadata is missing, the report can still be traced back to a company entity.
- **Deduplication and Refresh Policy.** Before inserting new reports, old records belonging to the same rating agency are cleared using `data_utils.delete_reports_by_rating_comp` This guarantees that the database reflects the most recent version of each report set, preventing duplicate data accumulation.
- **Report Table Insertion.** After parsing and cleaning, the data is inserted into the `report` table. Key fields include:
  - `rating_company`: The rating agency publishing the report (e.g., Fitch).
  - `company_name`: The rated company.
  - `title`: Official title of the report.
  - `date` and `year`: Publication date, with year stored separately for fast filtering.
  - `headings`: JSON string of automatically extracted headings for structural analysis.
  - `created_at` / `updated_at`: Timestamps for tracking record creation and updates.

  An index `idx_report_company_year` on `(rating_company, year)` is created to speed up queries that filter by agency and publication year.

1.1.2 Section Materialization

After the reports are normalized at the file level, the next step is to break them into finer-grained components—sections. This step enables downstream EFV extraction to work at the section level rather than on entire reports.

- **Standardized Section Splitting.** Each report is split into standardized sections such as *Derivation Summary*, *Liquidity and Debt Structure*, or *Peer Analysis*. This ensures that different reports from the same agency follow a unified structural naming convention.
- **Section Table Insertion.** Each section is inserted into the `report_sections` table with the following fields:
  - `report_id`: A foreign key linking the section to its parent report.
  - `section_name`: Standardized name for the section.
  - `contents`: The full text of the section.
  - `created_at` / `updated_at`: Metadata timestamps for lifecycle tracking.
- **Referential Integrity.** A foreign key constraint with `ON DELETE CASCADE` ensures that if a report is deleted, all associated sections are automatically removed, preserving database consistency.
- **Indexing for Efficiency.** An index on `(report_id, section_name)` is built to accelerate lookups during EFV extraction and hybrid ranking.

In summary, the `report` table serves as the foundational layer for managing report-level metadata, while the `report_sections` table provides the granularity required for EFV extraction and hybrid recommendation algorithms. This layered design balances flexibility and performance, ensuring that downstream modules operate efficiently.

1.2 Modeling Techniques

I employ a large language model (LLM) extractor to convert section-level text into three structured sets—*events*, *factors*, and *variables* (EFV). The extractor is implemented as a Python class `EventFactorVariableExtractor` (model: `gpt-4.1-2025-04-14`, temperature = 0 to ensure deterministic and reproducible outputs) and is invoked in batch over filtered rows from the report$_s ections table$..

1.2.1 LLM extractor design

The extractor uses a stable instruction template (*Stage-1: text analysis task*) that enforces a strict JSON schema with three top-level lists: `events`, `factors`, and `variables`. Each item contains verbatim evidence from the *current* passage to guarantee auditability and traceability.

- **Events.** Fields: `name` (short title), `contents` (concise description, $\leq$120 chars), `event_type` (enum when applicable), `period` (free text such as "FY2024", "Mar-2025"), and `evidence` (verbatim snippet).

- **Factors.** Fields: `name`, `contents`, optional `period`, and `evidence`.
- **Variables.** Fields: `name`, `contents`, `value` (raw text number), `unit` (e.g., "%", "USD bn"), `period`, and `evidence`.

The prompt explicitly forbids speculation and requires copying numbers, dates, and named entities *verbatim* into the corresponding fields (no normalization). I operate with temperature = 0 for determinism. A retry layer with lightweight validation re-asks the LLM when the JSON contract is violated (e.g., malformed lists or missing mandatory fields).

Sections are selected via `data_utils.select_sections_from_db(...)`. A blacklist (`settings.EXCLUDE_SECTIONS`) removes boilerplate such as contacts or disclaimers (`is_section_includ` The remaining rows are fed into `extract_batch_rows(...)`. Before persisting new outputs for the requested sections, we call `data_utils.delete_efv_by_section_names(...)` to clear stale data, making the pipeline idempotent at the section scope.

Each extracted item is augmented with provenance (`report_id`, `section_id`, `section_name`) and written to the corresponding table with denormalized `section_name` for fast lookups:

`section_name`, `name`, `contents`, `event_type`, `period`, `evidence`}

1. **factor**: {`report_id`, `section_id`, `section_name`, `name`, `contents`, `period`, `evidence`}
2. **variable**: {`report_id`, `section_id`, `section_name`, `name`, `contents`, `value`, `unit`, `period`, `evidence`}

Each table has an index on `(report_id, section_name)` to support downstream recommendation queries. When sentence-level relations are inferred in later stages, they are recorded in `event_relation` with an edge `score`, enabling graph-aware ranking; however, the extractor itself remains purely generative-to-structured and does not enforce cross-entity linking decisions.

To control cost and variance, I simplified the original instruction schema by removing descriptive elements that did not contribute to extraction accuracy, reducing unnecessary token usage. Instead of returning the full evidence text, the extractor now outputs only the first five words along with the total character length, which can later be used to locate the exact snippet. This design significantly lowers token consumption while ensuring that re-running the extractor over the same report$_s ections rows still produces stable EFV records with reliable p$

- **Recommendation Logic:** The system generates recommendations for EFVs based on user inputs:
  - *Section only:* If only a section is selected, recommendations are generated from the global dataset across all companies.
  - *Section + Company:* If both section and company are selected, two modes are available:
    1. **Company-only Mode** – Uses only the company's historical data.
    2. **Hybrid Mode** – Combines company-specific and global data. The blending ratio is controlled by the weight parameters:
       - w_{comp} for company-level data.
       - w_{glob} for global cross-company data.

- **Hyperparameter Configuration:** The user can specify the maximum number of top results (k) to display for each entity type via the settings panel:
  - **Variables (k)** – Default: 20
  - **Factors (k)** – Default: 10
  - **Events (k)** – Default: 8

  These parameters balance breadth and precision of the recommendations.

- **Score Calculation:** Each EFV is scored using a combination of:
  1. **Frequency Score** – Measures how often the EFV appears historically.
  2. **Recency Score** – Gives higher weight to recent occurrences.
  3. **Link Bonus (Events only)** – Rewards events that are strongly connected to factors and variables.

  The final hybrid score is computed as:

$$Score = w_{comp} \times Score_{company} + w_{glob} \times Score_{global} + w_{freq} \times Frequency + Bonus$$

1.2.2 Ranking Strategies

I model the extracted EFV items as a lightweight heterogeneous graph stored in the `event_relation` table. Each row represents a sentence–to–node edge within a given `section_id`, carrying an importance weight `score`, and pointing to exactly one EFV node via `event_id` or `factor_id` or `variable_id`. For ranking, I aggregate edge weights over a set of target sections and then (optionally) canonicalize names before selecting Top–k.

Scopes and section resolution. Given a UI selection (*company*, *section_names*), I resolve two disjoint scopes of `section_id`s: (i) the *company scope*—sections from that company's reports (optionally filtered by `year_min`/`year_max`/`report_limit`); and (ii) the *global scope*—all companies' sections that share the same *section_names*. This is done via helper queries over `report` and `report_sections`.

Per-type score accumulation. For each node type $t \in \{\texttt{event}, \texttt{factor}, \texttt{variable}\}$, I join `event_relation` with the node table t inside the chosen scope and sum `er.score` by node:

$$raw\_score_i^{(t)} = \sum_{edges: e \to i,\ e.section \in \mathcal{S}} \texttt{er.score}(e),$$

yielding a list of $(raw\_id, raw\_name, raw\_score)$. I optionally deduplicate by name and keep the highest-scoring instance.

Company / Global / Hybrid views. I produce three views per type:

- **Company view** (*raw granularity*): aggregate within the company scope without canonicalization (`aggregate=False`) to preserve company-specific naming.
- **Global view** (*canonical granularity*): aggregate across all companies with canonicalization (`aggregate=True`) to merge aliases and clean names.

- **Hybrid view**: first canonicalize the company scope, then blend it with the global view using rank-based fusion.

Rank-based fusion with frequency bonus. Within each scope I convert raw node scores into rank-normalized values For each scope $s \in \{\text{comp}, \text{glob}\}$, let L_s be the sorted list with size $n_s = |L_s|$, and let $\pi_s(k)$ be the rank of item k (best = 1). I set

$$r_s(k) = \frac{n_s - \pi_s(k) + 1}{n_s}.$$

If $k \notin L_s$, define $r_s(k) = 0$. so the top item gets 1.0, the second gets $(n_s - 1)/n_s$, ..., and the last gets $1/n_s$. (This corresponds to the code that enumerates $i = 1, \ldots, n$ and sets $(n - i + 1)/n$.) I also normalize frequencies to $f_{\text{comp}}, f_{\text{glob}} \in [0, 1]$ by dividing by the scope-wise maximum (absent $\to 0$). The fused score is

$$\text{Score}(k) \;=\; w_{\text{comp}}\, r_{\text{comp}}(k) + w_{\text{glob}}\, r_{\text{glob}}(k) + w_{\text{freq}}\, \max\{f_{\text{comp}}(k), f_{\text{glob}}(k)\} + \mathbf{1}_{\text{both}(k)} \cdot \text{both\_bonus},$$

Top-k selection and rounding. Within each view and type, I deduplicate by (`canonical_id`, *normal* sort by (*Score*, `freq`) descending, and keep Top-k. Scores are rounded to a fixed number of decimals for display stability.

2 Challenges and Solutions

This section discusses the main challenges encountered during the project and how they were addressed.

1. **Variables recall.** I refined the *variables* specification to mandate a short canonical name (noun phrase) plus verbatim `value`, `unit`, and `period`, which makes the extractor reliably enumerate each occurrence in the passage.

2. **Events/Factors quality.** I adopted *per-section, short-passage* prompts: extracting one section at a time yields high recall with minimal duplication, while combining multiple paragraphs—especially across sections—degrades quality and causes omissions/near-duplicates.

3. **Evidence & determinism.** To suppress hallucinations and ease auditing, the model outputs `evidence_start` (first five words verbatim) and `evidence_offset` (full-sentence character length) instead of full sentences; with temperature = 0.0, this provides deterministic, verifiable anchors and I reject items without a resolvable in-text anchor.

4. **Unified EFV normalization.** Rather than rule-based cross-mapping, I ask the LLM to *jointly normalize* events, factors, and variables within the same task; given the section structure it produces concise action phrases and canonical names that support frequency statistics and name linking for the recommender.

5. **Cost control.** I simplified the schema by removing descriptors that do not improve accuracy, avoided chat-style "schema caching" and multi-section batching (both harmed precision/recall), and settled on a *per-section, single-shot* extractor with a compact `EFV_SCHEMA`, reducing tokens while preserving precision, reproducibility, and provenance.

A EFV Schema

```
EFV_SCHEMA: Dict[str, Any] = {
    "type": "object",
    "additionalProperties": False,
    "properties": {
        "events": {
            "type": "array",
            "description": (
                "Actions or happenings explicitly stated in THIS passage that hav
                "and that may affect the company's stock price or be credit-relev
                "include a time/period if shown. If one sentence mentions multipl
                "you may keep it as one event rather than force-splitting."
            ),
            "items": {
                "type": "object",
                "additionalProperties": False,
                "properties": {
                    "name": {
                        "type": "string",
                        "description": "Short, standardized label (5 words) for t
                    },
                    "contents": {
                        "type": "string",
                        "description": "Normalized action phrase in past/clear pr
                        "maxLength": 120
                    },
                    "event_type": {
                        "type": "string",
                        "enum": _EVENT_TYPE_ENUM,  # machine validation
                        "description": "One of the 18 controlled event categories
                    },
                    "period": {
                        "type": ["string", "null"],
                        "description": "Verbatim time/period string from THIS pas
                    },
                    "evidence_start": {
                        "type": "string",
                        "description": (
                            "Verbatim first five words of the sentence that conta
                        )
                    },
                    "evidence_offset": {
                        "type": "integer",
                        "description": (
                            "Total number of characters in the full sentence cont
                        )
                    }
```

```
            },
            "required": ["name", "contents", "event_type", "period", "evidenc
        }
    },

    "factors": {
        "type": "array",
        "description": (
            "Specific, reusable credit/rating considerations stated in THIS p
            "Include explicit forward-looking statements; keep the name a con
        ),
        "items": {
            "type": "object",
            "additionalProperties": False,
            "properties": {
                "name": {
                    "type": "string",
                    "description": "Short, standardized label (6 words) for t
                },
                "contents": {
                    "type": "string",
                    "description": (
                        "Generate a concise, standardized factor description
                        "constraint, policy, or risk stated in THIS passage.
                        "Do NOT include numeric values, dates, or subjective
                    )
                },
                "period": {
                    "type": ["string", "null"],
                    "description": "Verbatim time/period string if present; n
                },
                "evidence_start": {
                    "type": "string",
                    "description": (
                        "Verbatim first five words of the sentence that conta
                    )
                },
                "evidence_offset": {
                    "type": "integer",
                    "description": (
                        "Total number of characters in the full sentence cont
                    )
                }
            },
            "required": ["name", "contents", "period", "evidence_start", "evi
        }
    },
```

```
"variables": {
    "type": "array",
    "description": (
        "All observable, measurable quantities mentioned in THIS passage
        "Each item captures one metric/value mention with its unit and pe
        "If the same metric appears with different values or periods, out
    ),
    "items": {
        "type": "object",
        "additionalProperties": False,
        "properties": {
            "name": {
                "type": "string",
                "description": (
                    "A concise, standardized label for the measurable met
                    "It should be a short noun phrase (5 words), capturin
                    "without numeric values or periods."
                )
            },
            "contents": {
                "type": "string",
                "description": "Verbatim metric description as written in
            },
            "value": {
                "type": "string",
                "description": "Verbatim numeric/date/ratio text (e.g., '
            },
            "unit": {
                "type": ["string", "null"],
                "description": "Verbatim unit token as it appears (e.g.,
            },
            "period": {
                "type": ["string", "null"],
                "description": "Verbatim time/period expression (e.g., 'a
            },
            "evidence_start": {
                "type": "string",
                "description": (
                    "Verbatim first five words of the sentence that conta
                )
            },
            "evidence_offset": {
                "type": "integer",
                "description": (
                    "Total number of characters in the full sentence cont
                )
            }
        },
```

```
                    "required": ["name", "contents", "value", "unit", "period", "evid
                }
            }
        },
        "required": ["events", "factors", "variables"]
    }
```