

Table of Contents

1. Introduction
2. File-by-File Documentation
 - 2.1 `fetch_data.ipynb`
 - 2.2 `chunkdivide.ipynb`
 - 2.3 `fetch_citations.ipynb`
 - 2.4 `fetch_labels.ipynb`
 - 2.5 `opinion_analysis.ipynb`
 - 2.6 `thesis_main.ipynb`
3. Usage and Workflow
4. Dependencies and Environment

1. Introduction

This document provides detailed technical specifications for each Jupyter notebook in the analysis pipeline. It describes the end-to-end process of retrieving, cleaning, annotating, analyzing, and reporting on U.S. federal court opinions related to AI governance (2010–2025).

2. File-by-File Documentation

2.1 `fetch_data.ipynb`

Overview: Retrieves raw court opinions from the CourtListener API using AI-related keywords, with retry and backoff strategies, and saves the results to Excel or CSV.

Implementation Details:

- **Async Configuration:** `nest_asyncio.apply()` enables nested event loops in Jupyter.
- **HTTP Requests:** `aiohttp.ClientSession` issues GET requests in `fetch_page`, with parameters for retry count and exponential backoff.
- **Pagination:** `fetch_all_pages_concurrently` follows the next cursor until depletion, collecting JSON results and logging progress.
- **Error Handling:** Failed URLs are retried once via `retry_failed_urls`; exceptions are logged with timestamps.
- **Data Sanitization:** `process_and_save_data` constructs a `pandas.DataFrame`, expands nested `recap_documents`, strips control characters using a compiled regex, and writes to `.xlsx` (fallback to `.csv`).
- **Entry Point:** `main` assembles `api_url` with AI keywords and date filters, invokes the fetch pipeline, and writes output to user-specified path. The script is executed via `asyncio.run(main())`.

Inputs:

- `api_url` string including keyword logic and date/court filters.

- headers dict with API token.
- Optional parameters: retries, backoff_factor, batch_size, save_path.

Outputs:

- Excel/CSV file with opinion metadata (ID, date, court, text, recaps).
- Console log detailing fetch duration and error counts.

2.2 chunkdivide.ipynb

Overview: Splits a large JSON file of opinions into fixed-size chunks for downstream processing.

Implementation Details:

- Directory creation via `os.makedirs` ensures the output path exists.
- Input JSON is loaded into a Python list. The script iterates in increments of `chunk_size`, extracting sublists.
- Each chunk is serialized with `json.dump(..., indent=2)` to files named `chunk_{index}.json`.
- Progress statements report record counts and index ranges.

Inputs:

- `input_file`: source JSON path.
- `output_dir`: destination directory.
- `chunk_size`: maximum records per chunk.

Outputs:

- Multiple JSON chunk files in output_dir.
- Summary logs printed to the console.

2.3 fetch_citations.ipynb

Overview: Extracts legal citations from text chunks, normalizes them, and computes metrics for citation density, lexical complexity, hedging frequency, and doctrinal counts.

Implementation Details:

- **Data Ingestion:** load_data supports JSON Lines and JSON formats, validating plain_text presence.
- **Text Cleaning:** fast_clean_text applies BeautifulSoup to remove HTML tags.
- **Citation Parsing:** extract_citations uses eyecite.get_citations, groups by type, and applies regex rules for Federal Rules and Acts.
- **Normalization:** Helper functions (get_statute_codes, get_act_names_from_text, etc.) format citations into human-readable tokens.
- **Metrics Calculation:** Functions compute citation density (citations per sentence), lexical complexity (type-token ratio), hedging frequency (modal lexicon), and counts of 'ultra vires' and Chevron references.
- **Parallel Processing:** process_all_rows maps process_row across CPU cores using multiprocessing.Pool and displays a tqdm progress bar.
- **Output:** Results are written to a CSV file named <chunk>_output.csv.

Inputs:

- Path to a chunk JSON file containing plain_text.
- Optional output_csv_path.

Outputs:

- CSV with one row per opinion and columns for all citation and text metrics.

2.4 fetch_labels.ipynb

Overview: Uses Google's Gemini API to classify opinions by sector and judgment outcome, then merges these labels with citation metrics and case metadata to create a unified dataset.

Implementation Details:

- **Model Configuration:** The LegalAnalysisResult defines allowed sectors and outcomes. The system_prompt ensures JSON-only responses.
- **API Calls:** extract_sector_and_outcome invokes Gemini with zero temperature for determinism and handles rate limits via recursive retries.
- **Excel I/O:** write_result_to_excel initializes or appends to analysis(remaining).xlsx.
- **Data Integration:** Reads caselaw_meta.xlsx, citations_analysis_final.xlsx, and label_analysis_final.xlsx. Performs outer merges on id. Computes unique citation counts with pandas.unique. Incorporates Act names from an auxiliary Excel.
- **Persistence:** Saves the consolidated final_df.xlsx to Google Drive.

Inputs:

- Intermediate CSV/Excel files from prior steps.
- Environment variable GOOGLE_API_KEY.

Outputs:

- analysis(remaining).xlsx with sector and outcome labels.

- final_df.xlsx containing all features required for index computation and analysis.

2.5 opinion_analysis.ipynb

Overview: Finalizes the extraction of text and citation features on each chunk, preparing the complete feature set for the Regulatory Gap Index computation.

Implementation Details:

- Reuses cleaning, citation extraction, and metrics functions from 2.3.
- Bundles feature computation in process_row; parallelism over chunks in process_all_rows.
- Writes comprehensive metrics CSVs for use by thesis_main.ipynb.

Inputs:

- JSON chunk files.

Outputs:

- CSVs with per-opinion metrics: citation density/diversity, complexity, hedging, doctrinal counts.

2.6 thesis_main.ipynb

Overview: Aggregates all inputs to compute the Regulatory Gap Index (RGI), generate descriptive statistics, produce visualizations (histograms, bar charts, time series, heatmaps), and create geospatial maps of regulatory strain.

Implementation Details:

- **Data Loading:** Reads final_df.xlsx, preprocesses date fields, adjusts citation diversity and computes opinion_count.
- **RGI Computation:** Implements winsorization, log-transform, robust scaling, min-max normalization, feature directionality inversion, and computes the

mean across features in `compute_rgi`.

- **Visualization:** Uses Matplotlib and Seaborn to produce distribution plots, sector and court comparisons, temporal trend lines, and heatmaps.
- **Geospatial Mapping:** Reads U.S. states shapefile with GeoPandas, merges state-level RGP or median RGI, computes centroids for labeling, and exports GeoJSON.
- **Statistical Modeling:** Installs and utilizes `factor_analyzer` and `pingouin` for PCA validation and factor analysis.

Inputs:

- `final_df.xlsx` and shapefiles (`.shp`).

Outputs:

- Inline visualizations and GeoJSON for embedding in the thesis report.

3. Usage and Workflow

- Setup the system according to the configs
- Execute notebooks sequentially: `fetch_data` → `chunkdivide` → `fetch_citations` → `fetch_labels` → `opinion_analysis` → `thesis_main`.
- Verify intermediate files in `data/` and final outputs (reports and figures) in `reports/` or Google Drive.

4. Dependencies and Environment

- Python ≥3.8
- Libraries: `aiohttp`, `pandas`, `numpy`, `nlTK`, `eyecite`, `bs4`, `google-genai`, `matplotlib`, `seaborn`, `sklearn`, `statsmodels`, `geopandas`, `factor_analyzer`, `pingouin`
- Services: CourtListener API token, `GOOGLE_API_KEY`, Google Drive mount for persistent I/O.

