

Brief Analyzer Codebase Summary

Overview

Core Components

- Citation Extraction
- Citation Lookup and Enrichment
- API Interaction
- Data Processing and Formatting
- Utilities
- Tests

Component Relationships

Key Workflows

1. Citation Extraction from Legal Briefs
2. Citation Lookup via CourtListener API
3. Opinion Retrieval and Formatting
4. Citation Data Enrichment
5. Report Generation

Detailed Component Analysis

- Citation Extraction Tools
- Citation Lookup and Enrichment
- Data Processing and Formatting

Brief Analyzer Codebase Summary

Overview

The Brief Analyzer is a collection of Python tools designed to extract, analyze, and enrich legal citations from briefs and other legal documents. The codebase leverages the eyecite library for citation extraction and the CourtListener API for citation lookup and opinion retrieval.

Core Components

Citation Extraction

- **extract_authorities.py**: Extracts case citations from PDF briefs, identifies the Table of Authorities and Argument sections, and saves the extracted information to a database.
- **eyecite_extractor.py**: General-purpose citation extractor that uses the eyecite library to find and parse legal citations from various document formats.
- **extract_minimal.py**: Minimal version of citation extraction that only extracts volume, reporter, and page information.
- **extract_text.py**: Simple utility to extract text from PDF files and save raw and cleaned versions.

Citation Lookup and Enrichment

- **citation_lookup.py**: Performs citation lookups against the CourtListener API.
- **citation_lookup_test.py**: Interactive tool to test citation lookups with detailed output.
- **citation_info.py**: Command-line tool to retrieve and display citation information.
- **citations_enricher.py**: Enriches citation data from CSV files with additional metadata from CourtListener.

API Interaction

- **courtlistener_api.py**: Complete API client for CourtListener with various functions (search, citation lookup, opinion retrieval).
- **fetch_opinion.py**: Interactive tool to fetch and save case opinions by citation or ID.
- **get_opinion_text.py**: Retrieves full text of opinions from CourtListener by citation.

Data Processing and Formatting

- **format_opinion.py**: Converts opinion data to various formats (HTML, PDF, TXT, Markdown).
- **generate_report.py**: Creates reports from citation data in different formats.
- **citations_db.py**: Provides database functionality for storing and retrieving citation data.

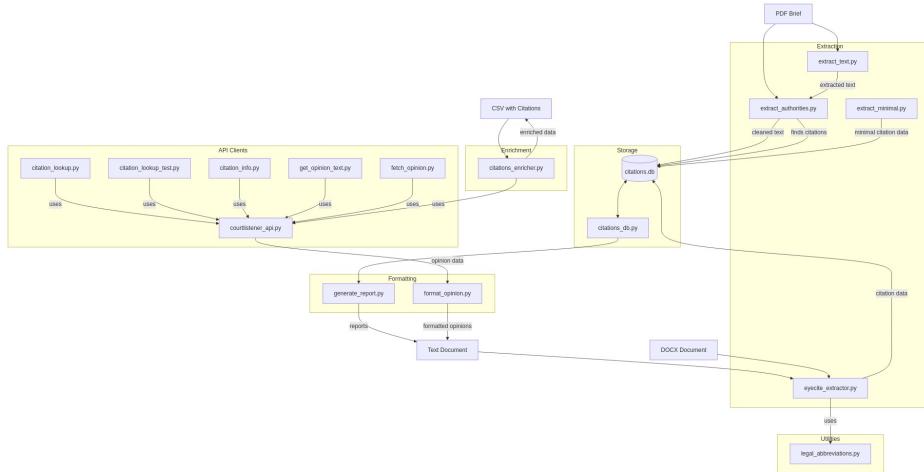
Utilities

- **legal_abbreviations.py**: Handles expansion of legal abbreviations to improve citation recognition.
- **debug_toa.py**: Helps with debugging Table of Authorities extraction issues.

Tests

- **test_citation.py**: Tests for citation parsing functionality.
 - **test_extraction.py**: Tests for text extraction functionality.
 - **test_eyecite_fields.py**: Tests for eyecite field extraction.
 - **test_party_extraction.py**: Tests for party name extraction from citations.
 - **test_party_name_logic.py**: Tests for logic used in party name processing.
-

Component Relationships



diagram_0

Key Workflows

1. Citation Extraction from Legal Briefs

The primary workflow in the Brief Analyzer codebase is extracting citations from legal briefs, particularly from the Table of Authorities section.

Step-by-Step Process:

- 1. Text Extraction:** `extract_authorities.py` uses `pdfminer.six` to extract raw text from the PDF brief
- 2. Section Detection:** The script identifies the Table of Authorities (TOA) section using pattern matching and structural analysis
- 3. Text Cleaning:** The raw TOA text is cleaned to remove page numbers and normalize formatting
- 4. Citation Extraction:** The `eyecite` library identifies legal citations in the text
- 5. Data Enrichment:** Additional metadata is extracted (case title, court, year, etc.)
- 6. Database Storage:** Citations are stored in a SQLite database for further analysis
- 7. Output Generation:** Various text files are created (raw text, cleaned TOA, argument section, etc.)

Input: `brief.pdf`

↓
`extract_authorities.py`

↓
Output:

- `brief_debug_raw.txt` (raw text)
- `brief.txt` (full text)
- `brief_toa_raw.txt` (raw Table of Authorities)
- `brief_toa_cleaned.txt` (cleaned Table of Authorities)
- `brief_argument.txt` (argument section)
- `brief_citations_db.txt` (formatted citations)
- `citations.db` (SQLite database)

2. Citation Lookup via CourtListener API

This workflow allows users to look up legal citations to get comprehensive case information.

Step-by-Step Process:

1. **Citation Input:** User provides a legal citation (e.g., “410 U.S. 113”)
2. **API Query:** The citation is formatted and sent to the CourtListener API
3. **Response Processing:** The API response is parsed to extract relevant information
4. **Display/Storage:** Results are displayed to the user and/or saved to a file

This functionality is provided through multiple interfaces: - `citation_lookup.py`: Simple command-line interface - `citation_lookup_test.py`: Interactive interface with comprehensive output - `citation_info.py`: Advanced command-line tool with multiple options - `courtlistener_api.py`: Direct API client with multiple endpoints

```
Input: "410 U.S. 113"
↓
courtlistener_api.py (citation_lookup method)
↓
Output:
- JSON response with case metadata
- Formatted text output (optional)
- Saved output file (optional)
```

3. Opinion Retrieval and Formatting

This workflow retrieves full legal opinions and formats them in various ways.

Step-by-Step Process:

1. **Citation Input:** User provides a legal citation or opinion ID
2. **API Query:** The citation is used to query the CourtListener API
3. **Text Retrieval:** The full opinion text is retrieved
4. **Format Conversion:** The opinion is formatted in multiple formats
5. **Storage:** Formatted opinions are saved to files

This functionality is provided through: - `fetch_opinion.py`: Interactive tool to fetch and save opinions - `get_opinion_text.py`: Library to retrieve opinion text by citation - `format_opinion.py`: Converts opinions to various formats

```
Input: "410 U.S. 113" or opinion ID
↓
fetch_opinion.py → get_opinion_by_citation() or
get_opinion_by_id()
↓
format_opinion.py
↓
Output:
- opinion_xxx.html (HTML format)
```

- opinion_xxx.txt (plain text)
- opinion_xxx.md (Markdown)
- opinion_xxx.pdf (PDF)

4. Citation Data Enrichment

This workflow takes a CSV file containing citation data and enriches it with metadata from the CourtListener API.

Step-by-Step Process:

1. **CSV Input:** User provides a CSV file with citation data
2. **CSV Parsing:** Citations are extracted from the CSV
3. **API Lookup:** Each citation is looked up via the CourtListener API
4. **Data Enrichment:** Additional metadata is added to each citation record
5. **CSV Output:** Enriched data is saved to a new CSV file

```
Input: citations.csv
↓
citations_enricher.py
↓
Output:
- enriched_citations.csv (with additional metadata)
- opinion_texts/ (optional directory with full opinion texts)
- lookup_errors.log (record of failed lookups)
```

5. Report Generation

This workflow generates formatted reports from citation data in the database.

Step-by-Step Process:

1. **Database Query:** Citation data is retrieved from the SQLite database
2. **Data Organization:** Citations are grouped by reporter
3. **Formatting:** Data is formatted as text or CSV
4. **Output:** Formatted report is displayed or saved to a file

```
Input: citations.db
↓
generate_report.py
↓
Output:
- Text report (console output or saved file)
- CSV report (saved file)
```

Detailed Component Analysis

Citation Extraction Tools

`extract_authorities.py`

- **Primary Purpose:** Extract citations from legal briefs' Table of Authorities
- **Key Functions:**
 - `clean_text()`: Removes page numbers and normalizes formatting
 - `find_toa_boundaries()`: Locates Table of Authorities section in document
 - `find_argument_boundaries()`: Locates Argument section in document
 - `find_citations()`: Uses eyecite to extract citations and store in database
- **Input:** PDF legal brief
- **Output:**
 - Text files with extracted content
 - Citations stored in SQLite database
- **Dependencies:** eyecite, pdfminer.six, sqlite3

`eyecite_extractor.py`

- **Primary Purpose:** General-purpose legal citation extraction
- **Key Functions:**
 - `extract_text_from_pdf()`: Extract text from PDF files
 - `extract_text_from_docx()`: Extract text from DOCX files
 - `extract_case_citations()`: Parse and extract citations
 - `process_file()`: Process files in various formats
 - `generate_html_output()`: Create HTML with citation links
- **Input:** Text, PDF, or DOCX documents
- **Output:** Structured citation data or HTML with linked citations
- **Dependencies:** eyecite, pdfminer.six, beautifulsoup4

`extract_text.py`

- **Primary Purpose:** Simple text extraction from PDFs
- **Key Functions:**
 - `extract_text()`: Extract and save both raw and normalized text
- **Input:** PDF file
- **Output:** Two text files - raw extracted text and cleaned text
- **Dependencies:** eyecite_extractor module

`extract_minimal.py`

- **Primary Purpose:** Minimal citation extraction example
- **Key Functions:**
 - `extract_minimal_citation_data()`: Extract basic citation components
- **Input:** Text with citations
- **Output:** List of citation dictionaries with volume, reporter, and page
- **Dependencies:** eyecite

Citation Lookup and Enrichment

courtlistener_api.py

- **Primary Purpose:** Comprehensive CourtListener API client
- **Key Functions:**
 - `search()`: Search for opinions
 - `get_opinion()`: Get opinion by ID
 - `get_docket()`: Get docket by ID
 - `citation_lookup()`: Look up cases by citation
 - `get_text_by_citation()`: Get full text by citation
 - `direct_citation_lookup()`: Look up using direct citation format
- **Input:** Search terms, citation strings, or IDs
- **Output:** JSON data from API or formatted text
- **Dependencies:** requests, dotenv

citation_lookup.py

- **Primary Purpose:** Simple citation lookup utility
- **Key Functions:**
 - `lookup_citation()`: Look up citation via CourtListener
 - `save_to_file()`: Save API response to file
- **Input:** Volume, reporter, page
- **Output:** JSON response saved to file
- **Dependencies:** requests, json

citation_lookup_test.py

- **Primary Purpose:** Interactive testing tool
- **Key Functions:**
 - Interactive prompts for citation lookup
 - Comprehensive results display
- **Input:** User-provided citation
- **Output:** Formatted text file with citation details
- **Dependencies:** courtlistener_api, json

citations_enricher.py

- **Primary Purpose:** Enrich citation data from CSV
- **Key Functions:**
 - `read_citations()`: Read citation data from CSV
 - `enrich_citation()`: Add metadata from CourtListener
 - `write_enriched_citations()`: Save enriched data to CSV
- **Input:** CSV file with citation data
- **Output:** Enriched CSV with additional metadata
- **Dependencies:** courtlistener_api, csv, json

Data Processing and Formatting

format_opinion.py

- **Primary Purpose:** Convert opinion data to multiple formats
- **Key Functions:**
 - `format_html()`: Format as HTML
 - `format_txt()`: Format as plain text
 - `format_md()`: Format as Markdown
 - `save_formats()`: Save in all formats
- **Input:** JSON opinion data
- **Output:** HTML, TXT, MD, and PDF files
- **Dependencies:** weasyprint, markdown, beautifulsoup4

generate_report.py

- **Primary Purpose:** Generate reports from citation database
- **Key Functions:**
 - `get_citation_data()`: Extract citation data from database
 - `group_by_reporter()`: Organize citations by reporter
 - `generate_report()`: Create formatted report
- **Input:** SQLite database
- **Output:** Text or CSV report
- **Dependencies:** sqlite3, tabulate

citations_db.py

- **Primary Purpose:** Database operations for citations
- **Key Functions:**
 - `CitationDB` class: Database interface
 - `ExtendedCitation` class: Enhanced citation object
- **Input:** Citation data
- **Output:** Database records
- **Dependencies:** sqlite3