

# Legal Brief Key Items Extractor

A dynamic NLP-powered tool for automatically extracting and analyzing key legal arguments from court briefs with precise page references and importance scoring.

## Overview

This Jupyter notebook application uses natural language processing to identify the top 10 most significant legal arguments from amicus briefs and other legal documents. Unlike manual extraction methods, this tool dynamically scores and ranks arguments based on multiple legal importance indicators.

## Problem Statement

Legal proceedings require thorough document analysis. Attorneys need tools that can efficiently:

- Extract critical arguments from lengthy legal briefs
- Provide precise page references for quick access
- Score arguments by legal importance
- Present both supporting and contextual information
- Save time in case preparation

This tool addresses these needs by automating the extraction and analysis process.

## Features

### Dynamic Extraction

- **No Hardcoding:** Automatically identifies key arguments using NLP
- **Multi-Factor Scoring:** Evaluates sentences based on:
  - Legal citations (statutes, regulations, case law)
  - Modal verb strength (violate, prohibit, require)
  - Argument indicators (FDA, states, public interest)
  - Subject relevance (case-specific terms)
  - Procedural importance (injunction, relief)

### Comprehensive Analysis

- Extracts top 10 key items with importance rankings
- Automatic categorization by legal topic
- Page-level reference tracking
- Score transparency and breakdown

- Statistical summaries

## Multiple Export Formats

- Excel workbook (multi-sheet with summary and categories)
- CSV files for critical arguments
- JSON export capability
- Pandas DataFrames for further analysis

## Technology Stack

- **Python 3.12+**
- **PyPDF2**: PDF text extraction
- **Pandas**: Data organization and analysis
- **NLTK**: Natural language processing and sentence tokenization
- **OpenPyXL**: Excel file generation

## Installation

### Prerequisites

```
bash
```

```
pip install PyPDF2 pandas openpyxl nltk
```

## NLTK Data

The notebook automatically downloads required NLTK resources:

- `punkt_tab`: Sentence tokenizer
- `stopwords`: English stopwords corpus

## Usage

### Step 1: Prepare Your PDF

Place your legal brief PDF in the same directory as the notebook, or provide the full path:

```
python
```

```
PDF_FILE_PATH = "Amicus Brief on Behalf of Mississippi, Alabama, Alaska, Arkansas etc....pdf"
```

### Step 2: Run Cells Sequentially

Execute cells 1-10 in order:

**Cell 1:** Import libraries and download NLTK data

**Cell 2:** Load PDF and extract text

**Cell 3:** Define DynamicLegalExtractor class

**Cell 4:** Initialize the extractor

**Cell 5:** Extract top 10 key items

**Cell 6:** View score breakdown

**Cell 7:** Analyze category distribution

**Cell 8:** Review page distribution

**Cell 9:** Export results to Excel/CSV

**Cell 10:** View final summary

## Step 3: Review Results

The tool generates:

- Console output with ranked arguments
- `dynamic_legal_analysis.xlsx` - Complete analysis workbook
- `key_items_dynamic.csv` - Quick reference CSV

## Output Structure

### Key Items DataFrame

Each extracted item includes:

- `rank`: Position (1-10)
- `category`: Auto-assigned legal category
- `importance`: Critical/High/Medium
- `page`: PDF page number
- `total_score`: Aggregated importance score
- `text`: Full argument text
- Score breakdowns by component

### Excel Workbook Sheets

1. **Key Items:** All extracted arguments with metadata
2. **Summary:** Statistics and metrics
3. **Categories:** Distribution breakdown

## Scoring Algorithm

Arguments are scored using weighted factors:

Factor	Weight	Description
Legal Citations	2.0	Presence of statutes, regulations, case law
Modal Strength	2.0	Strong legal verbs (violate, prohibit)
Argument Indicators	1.5	Key legal actors (FDA, states, Congress)
Subject Relevance	1.0	Case-specific terminology
Procedural Terms	0.5	Process words (injunction, motion)

**Minimum threshold:** 3.0 total score

**Critical threshold:** 8.0+ total score

**High priority:** 5.0-7.9 total score

## Code Examples

### Loading and Extracting PDF Text

```
python
```

```

import PyPDF2
import pandas as pd

def load_pdf_with_pypdf2(file_path: str) -> Tuple[str, pd.DataFrame]:
    """Extract text and create page-level DataFrame"""
    with open(file_path, 'rb') as file:
        pdf_reader = PyPDF2.PdfReader(file)
        total_pages = len(pdf_reader.pages)

        full_text = ""
        page_data = []

        for page_num, page in enumerate(pdf_reader.pages, 1):
            page_text = page.extract_text()
            full_text += f"\n--- PAGE {page_num} ---\n{page_text}"

            page_data.append({
                'page_number': page_num,
                'text_content': page_text,
                'word_count': len(page_text.split()),
                'char_count': len(page_text)
            })

        pages_df = pd.DataFrame(page_data)
        return full_text, pages_df

# Usage
document_text, pages_df = load_pdf_with_pypdf2('your_brief.pdf')

```

## Sentence Extraction with NLTK

```
python
```

```
from nltk.tokenize import sent_tokenize

def extract_sentences(pages_df: pd.DataFrame) -> List[Dict]:
    """Extract all sentences with metadata"""
    sentences_data = []

    for idx, row in pages_df.iterrows():
        page_num = row['page_number']
        page_text = row['text_content']

        # Skip header/footer pages
        if len(page_text) < 100:
            continue

        # Tokenize into sentences
        sentences = sent_tokenize(page_text)

        for sent in sentences:
            if len(sent) > 50: # Filter very short sentences
                sentences_data.append({
                    'text': sent.strip(),
                    'page': page_num,
                    'length': len(sent),
                    'word_count': len(sent.split())
                })

    return sentences_data
```

## Dynamic Scoring Algorithm

python

```
import re
```

```
def score_sentence(sentence: str) -> Dict[str, float]:
```

```
    """Score a sentence based on legal importance indicators"""
```

```
    scores = {
```

```
        'legal_citation': 0,
```

```
        'modal_strength': 0,
```

```
        'argument_indicator': 0,
```

```
        'subject_relevance': 0,
```

```
        'procedural': 0
```

```
    }
```

```
    sentence_lower = sentence.lower()
```

```
    # Legal citations (statutes, cases, regulations)
```

```
    citation_patterns = [
```

```
        r'\d+\s+U\.S\.C\. \s+\d+', # Federal statutes
```

```
        r'\d+\s+C\.F\.R\. \s+\d+', # Federal regulations
```

```
        r'\d+\s+S\. \s+Ct\. \s+\d+', # Supreme Court
```

```
        r'\d+\s+F\. \s+\d+th\s+\d+', # Federal courts
```

```
        r'\s+[A-Z][\w\s]+\s+\d+', # Case names
```

```
    ]
```

```
    for pattern in citation_patterns:
```

```
        if re.search(pattern, sentence):
```

```
            scores['legal_citation'] += 2
```

```
    # Strong modal verbs
```

```
    strong_modals = ['violate', 'defy', 'contradict', 'require',
```

```
                    'mandate', 'prohibit', 'unlawful', 'invalid']
```

```
    scores['modal_strength'] = sum(2 for word in strong_modals
```

```
        if word in sentence_lower)
```

```
    # Argument indicators
```

```
    argument_phrases = ['the fda', 'states have', 'public interest',
```

```
                        'court held', 'congress', 'administration']
```

```
    scores['argument_indicator'] = sum(1.5 for phrase in argument_phrases
```

```
        if phrase in sentence_lower)
```

```
    return scores
```

```
    # Example usage
```

```
    sentence = "The FDA's actions violate 21 U.S.C. § 355 and harm the public interest."
```

```
    scores = score_sentence(sentence)
```

```
    total_score = sum(scores.values())
```

```
    print(f"Total Score: {total_score}") # Output: Total Score: 7.5
```

## Extracting Top Key Items

python

```
def extract_key_items(sentences_data: List[Dict], top_n: int = 10) -> pd.DataFrame:
    """Dynamically extract top N key items"""
    scored_items = []

    for sent_data in sentences_data:
        scores = score_sentence(sent_data['text'])
        total_score = sum(scores.values())

        # Only include sentences with meaningful scores
        if total_score >= 3:
            scored_items.append({
                'text': sent_data['text'],
                'page': sent_data['page'],
                'total_score': total_score,
                **scores, # Unpack individual scores
                'word_count': sent_data['word_count']
            })

    # Sort by total score and take top N
    scored_items.sort(key=lambda x: x['total_score'], reverse=True)
    top_items = scored_items[:top_n]

    # Add rank and categorization
    for i, item in enumerate(top_items, 1):
        item['rank'] = i
        item['category'] = categorize_sentence(item['text'])
        item['importance'] = determine_importance(item['total_score'])

    return pd.DataFrame(top_items)
```

## Automatic Categorization

python



```
def categorize_sentence(text: str) -> str:
    """Automatically categorize based on content"""
    text_lower = text.lower()

    if 'u.s.c.' in text_lower or 'c.f.r.' in text_lower:
        return 'Legal Violation'
    elif 'dobbs' in text_lower or 'state' in text_lower:
        return 'Constitutional Authority'
    elif 'fda' in text_lower and ('approve' in text_lower or 'action' in text_lower):
        return 'FDA Actions'
    elif 'public interest' in text_lower or 'harm' in text_lower:
        return 'Public Interest'
    elif 'enforce' in text_lower or 'resource' in text_lower:
        return 'State Enforcement'
    else:
        return 'General Legal Argument'

def determine_importance(score: float) -> str:
    """Determine importance based on score"""
    if score >= 8:
        return 'Critical'
    elif score >= 5:
        return 'High'
    else:
        return 'Medium'
```

## Exporting Results to Excel

python

```

def export_to_excel(df: pd.DataFrame, pages_df: pd.DataFrame,
                   filename: str = 'legal_analysis.xlsx'):
    """Export analysis to Excel with multiple sheets"""
    with pd.ExcelWriter(filename, engine='openpyxl') as writer:
        # Key items sheet
        df.to_excel(writer, sheet_name='Key Items', index=False)

        # Summary statistics sheet
        summary = pd.DataFrame({
            'Metric': ['Total Items', 'Critical', 'High', 'Medium',
                      'Avg Score', 'Max Score', 'Total Pages'],
            'Value': [
                len(df),
                len(df[df['importance'] == 'Critical']),
                len(df[df['importance'] == 'High']),
                len(df[df['importance'] == 'Medium']),
                round(df['total_score'].mean(), 2),
                df['total_score'].max(),
                len(pages_df)
            ]
        })
        summary.to_excel(writer, sheet_name='Summary', index=False)

        # Category breakdown
        category_counts = df['category'].value_counts().reset_index()
        category_counts.columns = ['Category', 'Count']
        category_counts.to_excel(writer, sheet_name='Categories', index=False)

    print(f'Exported to {filename}')

```

## Complete Usage Example

```
python
```

*# Step 1: Load PDF*

```
document_text, pages_df = load_pdf_with_pypdf2('amicus_brief.pdf')
```

*# Step 2: Extract sentences*

```
sentences = extract_sentences(pages_df)
print(f'Extracted {len(sentences)} sentences')
```

*# Step 3: Extract and score key items*

```
key_items_df = extract_key_items(sentences, top_n=10)
```

*# Step 4: Display results*

```
print(key_items_df[['rank', 'category', 'importance', 'page', 'total_score']])
```

*# Step 5: Export to Excel*

```
export_to_excel(key_items_df, pages_df, 'analysis_results.xlsx')
```

*# Step 6: Filter critical items*

```
critical_items = key_items_df[key_items_df['importance'] == 'Critical']
print(f'\nFound {len(critical_items)} critical arguments')
```

## Analyzing Results with Pandas

python

*# Group by category*

```
category_analysis = key_items_df.groupby('category').agg({
    'total_score': ['mean', 'max', 'count'],
    'page': 'nunique'
}).round(2)
print("Category Analysis:")
print(category_analysis)
```

*# Find highest scoring items per category*

```
top_per_category = key_items_df.loc[
    key_items_df.groupby('category')['total_score'].idxmax()
]
print("\nTop item per category:")
print(top_per_category[['category', 'total_score', 'text']])
```

*# Page distribution analysis*

```
page_dist = key_items_df['page'].value_counts().sort_index()
print(f'\nKey items span pages {page_dist.index.min()} to {page_dist.index.max()}')
```

## Customization

### Adjust Scoring Weights

Modify the `score_sentence()` method:

```
python

# Increase importance of legal citations
scores['legal_citation'] += 3 # Changed from 2

# Add more weight to modal verbs
scores['modal_strength'] = sum(3 for word in strong_modals if word in sentence_lower)
```

### Add Domain-Specific Terms

Update keyword lists for your specific case:

```
python

# For pharmaceutical cases
key_subjects = ['mifepristone', 'abortion', 'rems', 'fda approval',
                'clinical trials', 'adverse events']

# For environmental cases
key_subjects = ['epa', 'pollution', 'emissions', 'clean air act',
                'environmental impact']
```

### Change Extraction Count

Extract more or fewer items:

```
python

# Extract top 20 items instead of 10
key_items_df = extractor.extract_key_items(top_n=20)

# Extract only critical items (score >= 8)
critical_only = key_items_df[key_items_df['importance'] == 'Critical']
```

### Custom Filtering

```
python
```

```

# Extract items from specific page range
pages_10_to_20 = key_items_df[
    (key_items_df['page'] >= 10) & (key_items_df['page'] <= 20)
]

# Get items with specific keywords
fda_related = key_items_df[
    key_items_df['text'].str.contains('FDA', case=False)
]

# Combine multiple filters
high_priority_fda = key_items_df[
    (key_items_df['importance'].isin(['Critical', 'High'])) &
    (key_items_df['category'] == 'FDA Actions')
]

```

## Example Output

### DYNAMIC EXTRACTION - TOP 10 KEY ITEMS

rank	category	importance	page	total_score	text
1	Legal Violation	Critical	8	12.5	The FDA's challenged...
2	Constitutional Auth	Critical	1	11.0	Dobbs v. Jackson...
3	Federal Criminal Law	Critical	10	10.5	Federal law (18 U.S.C...
...					

## Case Study: Alliance for Hippocratic Medicine v. FDA

This tool was developed to analyze the amicus brief in this case, which involves:

- FDA approval of mifepristone
- State authority post-Dobbs decision
- Federal preemption issues
- Public interest considerations

The dynamic extraction successfully identified critical arguments regarding FDA regulatory violations, Comstock Act provisions, and state enforcement burdens.

## Limitations

- Requires PDF text extraction (scanned PDFs need OCR preprocessing)
- English language only

- Optimized for U.S. legal documents
- Scoring algorithm may need domain-specific tuning
- Cannot interpret legal merit, only importance indicators

## Future Enhancements

- ☐ Support for multiple brief comparison
- ☐ Argument relationship mapping
- ☐ Citation network analysis
- ☐ Machine learning model training on legal corpus
- ☐ Interactive visualization dashboard
- ☐ Support for additional document formats (DOCX, TXT)

## Project Structure

```
legal-brief-analyzer/  
├── README.md  
├── legal_brief_analyzer.ipynb  
├── Amicus Brief on Behalf of Mississippi....pdf  
├── dynamic_legal_analysis.xlsx (generated)  
├── key_items_dynamic.csv (generated)  
└── requirements.txt
```

## Requirements File

```
txt  
  
PyPDF2>=3.0.0  
pandas>=2.0.0  
openpyxl>=3.1.0  
nltk>=3.8.0
```

## Troubleshooting

### PDF Not Found

Ensure the PDF filename matches exactly, including spaces and special characters.

### NLTK Download Issues

Manually download NLTK data:

```
python
```

```
import nltk
nltk.download('punkt_tab')
nltk.download('stopwords')
```

## Empty Results

Check if PDF text extraction was successful. Some PDFs may be image-based and require OCR.

## Score Too Low

Adjust the minimum threshold in `extract_key_items()`:

```
python

if total_score >= 2: # Lower threshold
```

## License

This project is provided for educational and professional legal analysis purposes.

## Contributing

Contributions are welcome. Key areas for improvement:

- Additional legal domain patterns
- Alternative scoring algorithms
- Support for other jurisdictions
- Performance optimization for large documents

## Contact

For questions or suggestions about this legal analysis tool, please open an issue in the project repository.

## Acknowledgments

Developed to assist attorneys in case preparation by automating the extraction and analysis of key legal arguments from court briefs and legal documents.

---

**Note:** This tool is designed to assist legal professionals but does not replace human legal analysis and judgment. Always verify extracted arguments and page references against source documents.