`Supreme Court Documentation

Kory Rosen

Due on 2024-03-11

Introduction

The **Supreme Court Opinion Analysis** Shiny App is an R Shiny Application that provides an interactive platform for exploring and analyzing Supreme Court case opinions. The app is geared towards an audience of legal scholars and lawyers specializing in the field of Constitutional Law, researchers, and students who have an interest in analyzing Supreme Court opinions. The main goal of the project is to enable users to interactively explore Supreme Court case opinions, in a **user-friendly** environment, through using **data visualizations** that are based and filter on various variables such as as year range, case name, author, decision type, text type, political leaning of the opinion writer, word count, opinion text, etc.

Data Description

The Supreme Court Opinion Analysis Shiny App utilizes a dataset containing information on Supreme Court cases. The data was collected from the CourtListener API, which provides access to legal opinions and related information.

Data Retrieval

The dataset was obtained by making API requests to the CourtListener API endpoints for clusters, opinions, people (authors), and dockets. The data was retrieved in JSON format and processed using Python scripts, as R is not very friendly to requests that have over tens of thousands of rows. I had to use multiprocessing in order to download the data in a feasible time. The specific steps for data retrieval included:

1. Fetching cluster data: The script retrieved cluster data from the clusters endpoint in the CourtListener API, getting fields such as case names, date filed, decision direction, vote counts, and sub-opinion IDs.

2. Fetching opinion data: Using the sub-opinion IDs obtained from the cluster data, the script made requests to the opinions endpoint in the CourtListener API to retrieve that data, getting fields such as the opinion text, author ID, and cluster ID.

3. Fetching author data: Using the unique author IDs obtained from the opinion data, the script made requests to the person endpoint of the CourtListener API to retrieve author information, such as name, date of birth, political affiliation, and tenure on the Supreme Court details.

4. Fetching docket data: The script used the cluster IDs to retrieve docket IDs from the dockets endpoint of the CourtListener API.

The fetched data was stored in pickle files for further processing and analysis.

Data Cleaning and Preprocessing

The dataset underwent initial preprocessing steps before being stored in a PostgreSQL database. Some of these steps include:

1. The columns of the DataFrames that were pickled are renamed for consistency and clarity.
2. Drop unnecessary columns in these dataframes.
3. Filtered to remove entries for individuals who are not Supreme Court justices.
4. Create columns for "political_party", "date_start", and "date_left" by extracting relevant information from the "political_affiliations" column and then I dropped "political_affiliations".
5. Update entries for justices with correct dates for birth, start, and end of tenure.
6. Parsing the opinion text from various formats (plain_text, html, html_lawbox, html_columbia, html_anon_2020, xml_harvard, html_with_citations) using BeautifulSoup to extract the text content.
7. Consolidating the parsed opinion text into a single column (opinions_text) as a list of unique, non-empty sections.
8. The cleaned DataFrames are merged based on common columns such as "cluster_id" and "author_id" to create a single consolidated DataFrame called new_data.
9. Cases are updated with information on majority and minority votes, as well as the final decision (e.g., "reversed", "affirmed", "vacated and remanded", etc.).
10. Converting opinion text to UTF-8 encoding.
11. Replacing empty lists in the opinions_text column with None.
12. Handling missing data by replacing empty values with values that would be able to transform into suitable values in R.

The data was then stored in a PostgreSQL database for efficient storage and retrieval.

After retrieving the data from the PostgreSQL database using the RPostgres server, I performed some preprocessing and cleaning in R to ensure data quality and consistency. The key steps included:

1. Preprocessing opinion text: I removed HTML tags and special characters from the opinions using regular expressions and string manipulation techniques in order to try to standardize opinions.
2. Extracting author information: I extracted author names from the opinion text using regular expressions and matched them with the author data obtained from the API in order to fill in author-related fields.

3. Splitting opinions into sections: I split the opinion text into its specific sections like "Syllabus," "Opinion of the Court," "Concurring Opinion," and "Dissenting Opinion" using regular expressions in order to produce an in-depth analysis.
4. Calculating word counts: I calculated the word count for each opinion section, which would help me in producing data visualizations based on this metric.
5. Removing almost-similar rows: I calculated by using threshold almost similar rows that were extremely similar to another row, like if there were two syllabus for a case that were nearly identical, one would get removed.

Data Description and Structure

The Supreme Court case data used in this project covered cases from January 2019 to Feburary 2024. The final dataset represents a total of 278 Supreme Court cases, with 842 rows of data. Each row corresponds to a specific opinion section within a case, allowing for detailed analysis at the opinion level. The final dataset used in the Shiny App consists of 15 variables:

- case_name (character type): The name of the Supreme Court case.
- date_filed (Date type): The date the case was filed, which is in the format "YYYY-MM-DD"
- decision (character type): The decision reached by the Supreme Court (i.e., affirmed, reversed).
- scdb_votes_majority (numeric type): The number of votes in the majority decision. This could be a value from 5-9.
- scdb_votes_minority (numeric type): The number of votes in the minority decision. This could be a value from 0-4.
- opinions_cited (numeric type): The number of opinions cited.
- text_type (character type): The type of opinion sections. The choices include Syllabus, Opinion of the Court, Concurring Opinion, Dissenting Opinion, and Other.
- opinions_text (character type): The text of a section of an opinion.
- word_count (numeric type): The number of words in the opinions_text.
- full_name (character type): The full name of the opinion author.
- author_id (numeric type): The unique identifier of the opinion author.
- political_party (character type): The political party affiliation of the opinion author, which is either "D" or "R".
- date_dob (Date type): The date of birth of the opinion author.
- date_start (Date Type): The start date of the opinion author's tenure on the Supreme Court.
- date_left (Date Type): The end date of the opinion author's tenure on the Supreme Court (if applicable). There are only two justices who fit this category, Justice Breyer and Justice Ginsburg.

The dataset provides a comprehensive representation of Supreme Court opinions, which allows me to conduct analysis based on case details, opinion content, and author characteristics.

Questions for Analysis

The Shiny app aims to address the following questions: 1. How has the length of Supreme Court opinions varied over time? 2. What are the most frequent vote splits in Supreme Court decisions, and what is their distribution amongst cases? 3. What are the common themes and topics addressed in the opinions? 4. How strongly are sentiments expressed in the opinions? 5. Do sentiments correlate with the decisions or the authors of the opinions?

Insights from Data Analysis

Through exploratory data analysis and visualization using the Shiny App, several interesting insights were uncovered:

1. An examination of the Supreme Court opinion lengths from 2019 to 2024 when using default settings in the Shiny App indicates a relatively consistent distribution of opinion lengths over time. This suggests that, even though outliers are present, the majority of opinions cluster within a moderate word count range.
2. The app reveals a notable concentration of both unanimous (9-0) and 5-4 decisions. This shows that there are areas of both strong judicial consensus and ideological divide within the Court.
3. Sentiment Expression: The app's sentiment analysis indicates a range of emotions, with a prevalence of negative to neutral to positive sentiments, suggesting a consistent tone in judicial writing. The sentiment bar corresponding to 'trust' often stood out to me, as I was surprised to see that much "trust" present.

These findings provide a comprehensive understanding of Supreme Court decisions and how they have evolved over time.

Reproducibility

To guarantee the reproducibility of my work, I have documented each step of the data collection, cleaning, and analysis pipeline. Here is a breakdown:

Data Collection with Google Colab+

1. Set up Google Colab+: I used Google Colab+ for its extended runtime and background execution capabilities. This allowed me to run my data scraping .ipynb scripts overnight without interruptions due to network connectivity or hardware limitations.
2. API Requests: In the jupyter notebook scripts, I made requests to the CourtListener API to fetch data from various endpoints (clusters, opinions, people, dockets) using the requests library in Python. Then, I stored the fetched data was stored in pickle files for later use. This ensured that the raw data could be accessed at a later time, so I would not have to continously request from the API given the amount of data scraped.

Data Cleaning and Preprocessing in Python

1. Loading Data: I loaded the raw data from pickle files from my data collection processes using the pickle library.
2. Dataframe Manipulation: I used the pandas library in Python to clean and preprocess the data. This involved renaming columns, dropping unnecessary columns, and handling missing values.
3. Parsing: For columns containing opinion text that had HTML or XML content, I used BeautifulSoup to extract the plain text and then coalesced them into a column.
4. Date Formatting: I standardized the format of date columns to ensure consistency.
5. Text Encoding: I encoded the opinions_text data in UTF-8.
6. Export to a feather: I exported my data into a feather file.

Data Storage in PostgreSQL

1. Database Setup: I set up a PostgreSQL database to store the cleaned and processed data, which provided a structured and fast way of storing and querying the data.
2. Data Insertion: The cleaned data was inserted after doing some data type manipulations to the feather file into the database using SQL commands. The database schema can be found in "PuttingFileIntoDataBase.ipynb" so that it could be reproduced.

Data Retrieval in R

1. Database Connection: I established a connection to our PostgreSQL database using the RPostgres package, fetching the cleaned data into an R dataframe for analysis, as the feather file was too slow and was making R lag out.

Data Preprocessing in R

1. Preprocessing opinion text: I removed HTML tags and special characters from the opinions using regular expressions and string manipulation techniques in order to try to standardize opinions.
2. Extracting author information: I extracted author names from the opinion text using regular expressions and matched them with the author data obtained from the API in order to fill in author-related fields.
3. Splitting opinions into sections: I split the opinion text into its specific sections like "Syllabus," "Opinion of the Court," "Concurring Opinion," and "Dissenting Opinion" using regular expressions in order to produce an in-depth analysis.
4. Calculating word counts: I calculated the word count for each opinion section, which would help me in producing data visualizations based on this metric.
5. Removing almost-similar rows: I calculated by using threshold almost similar rows that were extremely similar to another row, like if there were two syllabus for a case that were nearly identical, one would get removed.

Analysis and Visualization in R

1. Shiny App Development: I developed a Shiny app in R to visualize and analyze the data. I used plotly for creating interactive plots that allow users to explore the data

dynamically and filters to select specific cases, authors, and date ranges, enabling customized analysis.

Detailed Documentation and Code Sharing

1. Code Comments: I added comments to my code for explainability purposes.
2. GitHub Repository: The entire codebase, including the ipynb scripts, Python cleaning scripts, and R Shiny app code, was shared in my GitHub repo.

Design Decisions (What/Why/How Framework & Encodings/Mappings)

The Shiny app's data visualizations were chosen based on the "what-why-how" framework, which could be found below:

Scatter Plot: Opinion Length Over Time

What

Datasets: The dataset is a table where each row is an item (court case) and columns are attributes.

Attributes: The attributes for this visualization are quantitative and ordered, representing opinion length, and temporal, representing filing dates.

Why

Actions: I want to consume this data by discovering and presenting it visually.

Target: My target is to identify trends and outliers.

How

Encode: I map opinion length to vertical position and filing date to horizontal position. Color hue indicates opinion density, which provides information about data distribution over time.

Reduce: Users can filter data based on a date range to focus on specific periods.

Bar Chart: Vote Splits Distribution

What

Datasets: The dataset for the vote splits visualization can be considered a table, with items (cases) having attributes (vote split categories and counts).

Attributes: The vote splits are categorical attributes that represent different types of decision outcomes in the Supreme Court cases. The number of cases per vote split is a quantitative attribute.

Why

Actions: I present the distribution of vote splits and enable users to compare the frequency of different types of vote splits.

Target: The distribution of cases across various vote splits will allow users to identify common and uncommon splits, as well as trends and outliers.

How

Encode: The bars are ordered categorically along the x-axis according to vote splits. The height of each bar corresponds to the count of cases for each vote split. Color hue is an aspect that helps differentiate different vote splits from each other.

Reduce: Users can filter data based on a date range to focus on specific periods.

Word Cloud: Thematic Content

What

Datasets: This visualization handles a table dataset where items (words) are rows with attributes like frequency of each word.

Attributes: The words in the opinion texts are categorical attributes, and their frequency of occurrence is a quantitative attribute.

Why

Actions: I want to discover and present my findings as I am visually representing the most frequently occurring words within the Supreme Court opinions. This will allow users to grasp the main themes and topics discussed in an opinion(s).

Target: The targets are the features and outliers in the data, specifically the words that appear most and least frequently.

How

Encode: Word frequency is mapped to size, making more frequent terms larger and less frequent terms smaller. The color is used aesthetically to differentiate words and does not encode any particular data attribute but does enhance readability.

Manipulate: Users can interact with the word cloud, hovering over words to get specific frequency counts.

Reduce: Users can filter data based on some filters such as a date range to focus on specific periods.

Bar Plot: Sentiment Analysis

What

Datasets: This is a table dataset where items (individual opinion texts) are associated with attributes (sentiment scores).

Attributes: The sentiment scores are quantitative attributes derived from the opinion texts.

Why

Actions: By presenting and comparing, users are presented with sentiment scores to consume and enjoy the information visually.

Target: The target is the distribution of sentiment scores across opinions, which might reveal trends, extremes, and outliers in the data.

How

Encode: We map the categorical data of sentiments to the color of the bars, and quantitative data of sentiment frequency to the length of the bars.

Reduce: Users can filter data based on some filters such as a date range to focus on specific periods.

Improvements for the Future

While the Shiny app already does a good job analyzing Supreme Court case opinions, there is still room for improvement. Allowing users to traverse through decades of legal decisions would help detect more apparent shifts in judicial perspectives and policy impacts. Further, leveraging natural language processing and machine learning could unravel deeper thematic and sentiment trends. Lastly, optimizing the app for speed is crucial—improving data processing and loading times would not only streamline user interaction but also enable the handling of larger, more complex datasets without compromising performance. These advancements could transform the app into an even more powerful tool for legal scholars, constitutional law lawyers, students, academics, historians, and the public.

References

Munzner, T. (2015). Visualization analysis & design. CRC Press/Taylor & Francis Group. https://doi.org/10.1201/b17511

https://www.courtlistener.com/help/api/rest/

Appendix

Time Log

Data Table

```r
timelog <- "C:\\Users\\Kory\\OneDrive\\Documents\\timelog.RData"

load(timelog)
timelog <- timelog

library(knitr)

kable(timelog, format = "pandoc", caption = "Timelog Dataset")
```

Timelog Dataset

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
Activ	1	2024-	2024-	Researc	Thinkin	20	20

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
ity 1		01-24	01-24	h	g about what dataset I wanted to choose during class and how I wanted to approach it.		
Activity 2	1	2024-01-24	2024-02-01	Research	Note from 01/24/2024: I did this part in class and after class on this date. Given that I wanted to do Supreme Court data, I started to search for how to find a dataset online. I first found a dataset	180	300

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					on Kaggle, but the issue would be that I would have to scrape the actual opinions from the website, which would be a pain. Then, I found CourtListener, which would be a great choice, but I had to figure out how I would download the data since the bulk data would be huge to downloa		

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					d. Update from 02/01/2024: I gave up using CourtListener bulk data because it would come to over 50 GB of data and my comptuer cannot hold that, so had to find a different way.		
Activity 3	3	2024-02-04	2024-02-04	Research	Note from 02/04/2024: I tried to use this dataset on Hugging Face regarding Supreme Court Cases, but	180	180

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					could not get it to work. Tried for about 3 hours		
Activity 4	3	2024-02-09	2024-02-19	Research	Looked for another way to do this because I was keep stricking out without luck. Was Googling for APIs that had this for about an hour and a half. Update from 02/19/2024: I found the CourtListener API, which seemed like the least	120	210

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					memory intensive and computationally intensive way of doing this so far, so I will use this it seems. Based on testing for about 2 hours, the API does what I want it to do.		
Activity 5	6	2024-02-20	2024-02-20	Email Correspondence	I was corresponding with Mike Lissener, the cofounder of the project. My email was the following: Hope all is well. I am a 4th	60	60

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					year at Grinnell College majoring in computer science and political science and I need your help for my project in software development class. I am planning on doing a Natural Language Processing project on Supreme Court Cases from 2019-2022 and was wondering if there is		

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					any way to scrape more than 10 pages worth of data using the API because 10 pages isn’t enough to get all the information. I tried with bulk data but that is too much of a hassle. Thank you so much. He responded that is all he has as of now, so I went to find ways to use the API.		
Activ	6	2024-	2024-	Researc	For 3	180	180

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
ity 6		02-21	02-21	h	hours during the late afternoon (between 4 and 7), I was exploring how the DRF worked and how queries work within it.		
Activity 7	6	2024-02-23	2024-02-23	Data Collection	Created a small snippet of the dataframe that I wanted to create. (Did this between 4pm and 8:30pm)	270	270
Activity 8	7	2024-02-26	2024-02-26	Visualization Creation	Created some sample visualizations that I could do and saw what worked and what did	240	240

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					not work (Did this between 5pm and 9:00pm)		
Activity 9	7	2024-03-02	2024-03-05	Data Collection	Created a script that scraped the whole amount of data we needed. First, I had to conceptually think of how I could do this without making that many API calls. This was the hardest part because I did not want to overrun the nonprofits servers,	60	300

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					which I had done during my testing (Worked on it between 12:30pm and 5pm). Planning on doing testing later. Update from 03/05/2024: Trials keep failing during the middle of the test run. Keep getting annoyed. Have to figure out a way to stop this from happening. I am implementing pickle		

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					files now and going to split it into shorter files. I worked on it today for about an hour so far. Update from 03/05/2024 (Later at night): I was so far in and my internet got disrupted. Need to think of a workaround for this.		
Activity 10	8	2024-03-06	2024-03-06	Data Collection	From 6pm to 11pm, I created the script with multiprocessing and	300	300

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					failsafes using Google Collab+ for Clusters and recieved results.		
Activity 11	8	2024-03-07	2024-03-08	Data Collection	From 6pm to 11pm, I created the script with multiprocessing and failsafes using Google Collab+ for Opinions and did not recieve results because there was an error. Update from 03/08/2024: I fixed the error and got results. Took	210	480

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					about 3 hours to run and 30 minutes to fix and it was done sporadically throughout the day.		
Activity 12	8	2024-03-08	2024-03-08	Data Collection	From 7pm-7:30pm, I created the script with multiprocessing and failsafes using Google Collab+ for Authors and did recieved results	30	30
Activity 13	8	2024-03-08	2024-03-08	Data Collection	From 7:30pm-8:00pm, I created the script with multiprocessing	30	30

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					and failsafes using Google Collab+ for Docket ID and did recieved results		
Activity 14	8	2024-03-09	2024-03-09	Data Transformation	From 11am-8pm, Fixed the data and make it ‘clean’ by combining datasets, changing types, and getting it suitable to work in R.	540	540
Activity 15	9	2024-03-10	2024-03-10	Shiny App Creation	From 1pm-9pm, I used the visualizations that I already had and adapted them	480	480

activityID	week	dateStarted	dateCompleted	activityType	description	numberOfMinutesWorkedThatDate	totalNumberOfMinutesWorked
					into Shiny format and also created some new visualizations.		
Activity 16	9	2024-03-11	2024-03-11	Data Transformation	From 6am to 8am, I removed duplicates in my dataset using a function I had created that was inspired by fuzzy matching.	120	120