

AI Humanities Report

Spring Quarter 2024

Prepared by: Pavel Martinez, Greg Chang, Nidhi Vadulas on May 28, 2024.

Reviewed by: [name of any reviewers]

## Background

The College of Letters and Sciences at the University of California, Davis, proposed for a Center for AI and Experimental Futures (CAIEF). The motivation for the center is to highlight the impacts, ethics, and humanities of Artificial Intelligence (AI) and how it affects us. Our goal for the project is to collect various relevant journal articles to serve as the foundational body of literature that future researchers will use to study the realities and potentialities of the interface between human and artificial intelligence. To access the journal articles, we first need their ISSN (International Standard Serial Number). Gathering the ISSN is

Goals for the current Datalab research students:

1. Scrape UCD Library for Journal Title, Link to Journal (URL), Online availability, Publisher, Subjects, Languages, ISSN
2. Use ISSN to collect a pool of Journal contents (.txt & .pdf)
3. Create corpus of collected journals
4. Analyze metadata and journal contents
5. Populate final database with relevant journals

Time frame: 10 weeks duration of Spring Quarter 2024

**Partners**

Data Lab Contributors

- Carl Stahmer (DataLab Sponsor) [[cstahmer@ucdavis.edu](mailto:cstahmer@ucdavis.edu)]

- Elise Hellwig (DataLab Technical Lead) [[echellwig@ucdavis.edu](mailto:echellwig@ucdavis.edu)]

- Greg Chang (Student Contributor) [[gjchang@ucdavis.edu](mailto:gjchang@ucdavis.edu)]

- Pavel Martinez (Student Contributor) [[pimartinez@ucdavis.edu](mailto:pimartinez@ucdavis.edu)]

- Nidhi Vadulas (Student Contributor) [[nvadulas@ucdavis.edu](mailto:nvadulas@ucdavis.edu)]

Domain Leads

- Colin Milburn (Primary Partner, English Department) [[cnmilburn@ucdavis.edu](mailto:cnmilburn@ucdavis.edu)]

- Emily Merchant (STS Department Partner) [[ekmerchant@ucdavis.edu](mailto:ekmerchant@ucdavis.edu)]

- Finn Brunton (STS Department Partner) [[fpbrunton@ucdavis.edu](mailto:fpbrunton@ucdavis.edu)]

## Approach

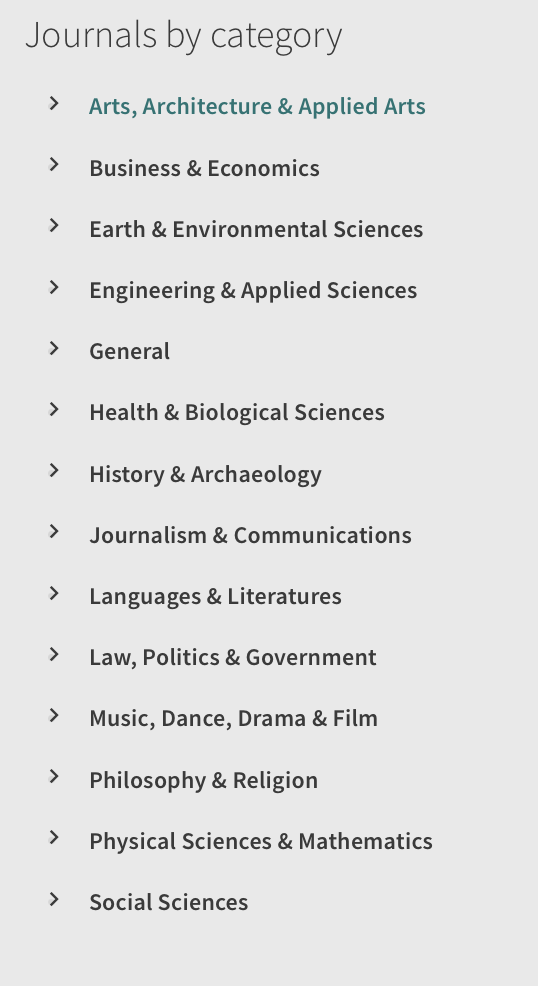
Our approach was to scrape the [UC Davis Library](https://search.library.ucdavis.edu/discovery/jsearch?vid=01UCD_INST:UCD&lang=en) website and we started using Rvest and xml2 libraries. Due to html accessibility restrictions due to javascript running on the UC Davis Library website and it being a complex dynamic website with pop-up pages, we pivoted to using the RSelenium library to aid us in our web scraping which was a new package to us. However unfamiliar we were with RSelenium, we saw great success scraping html tag data with that r library.

The following [RSelenium video tutorial](https://www.youtube.com/watch?v=U1BrIPmhx10) was used to get us started with learning how to select html elements, clicking items, scraping text, etc. The tutorial uses google chrome as the browser of choice, however, we recommend using firefox instead as chrome’s latest version seems to be unsupported by the current RSelenium library.

Our goal was to create a data frame with the following information for every journal under the UC Davis library website: Journal Title, Link to Journal (URL), Online availability, Creator, Subjects, Genre, Description, alternative title(s), related titles, Publisher, Creation Date, Journal Format, Frequency, Language of Journal, source and ID/ISSNs. We will refer to this journal as the “full detailed journal” data frame.

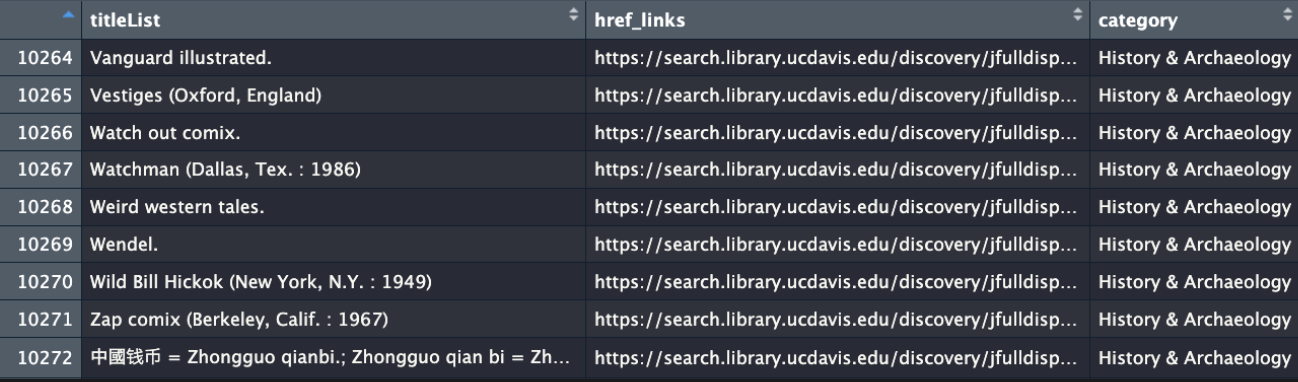
### Journal categories

Before scraping all the above journal details, we first scraped only the title and individual journal link for every of the 14 subject categories under “Journal search” in the website top bar. We had to scrape every category separately since we did not have an A-Z list where all the journals were displayed.

Journals displayed in Arts, Arc. & Applied Arts Category


*Journals by category Journals displayed in Arts, Arc. & Applied Arts Category*

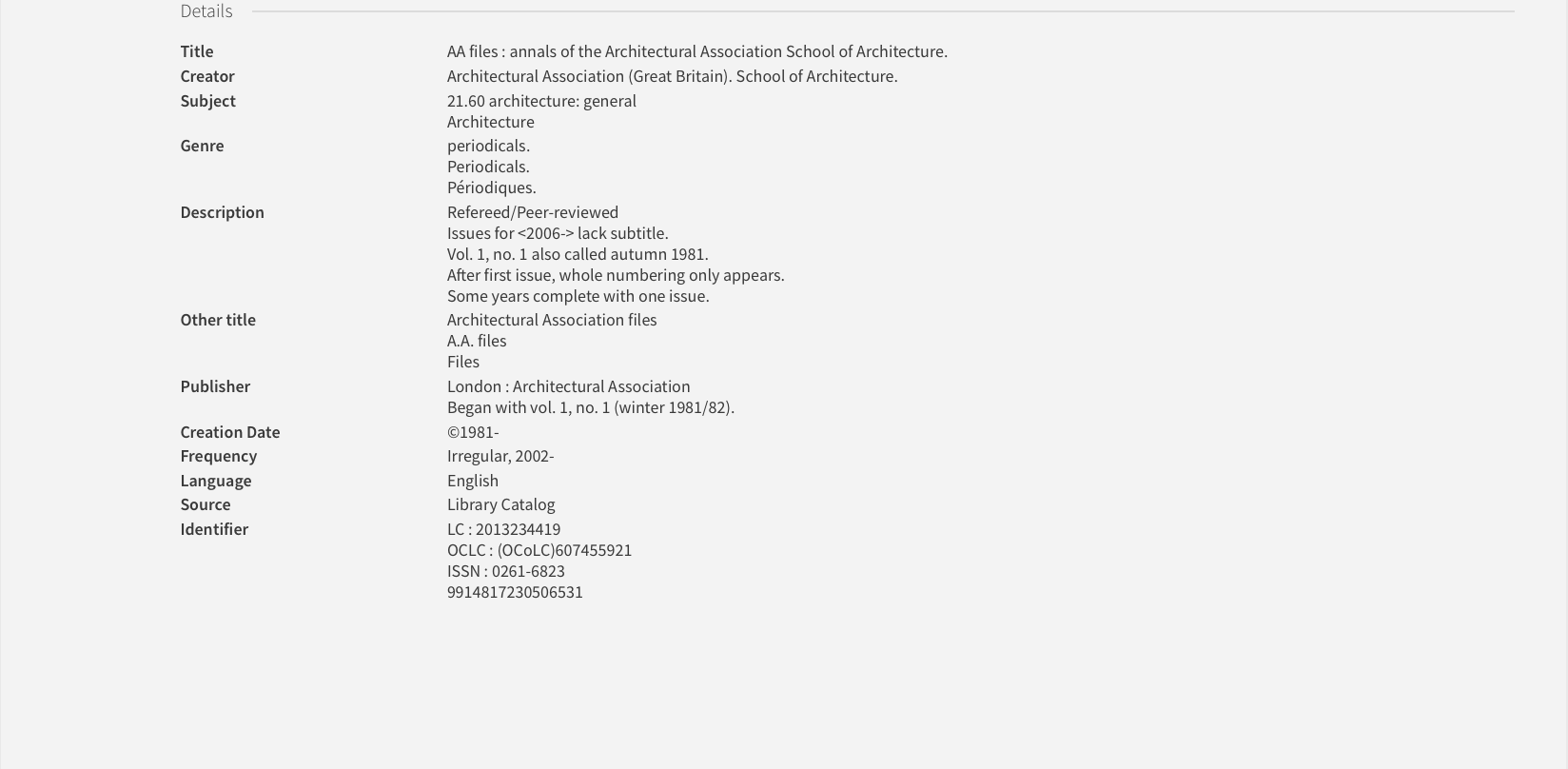
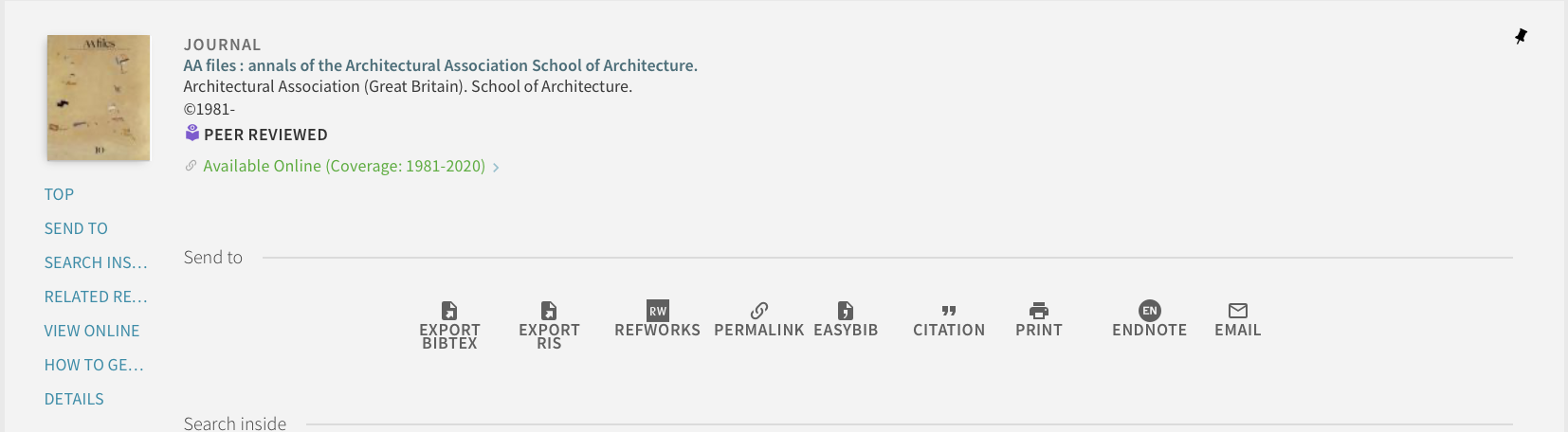
We only collected the **title** and **journal link** first due to the limited details displayed per journal when bulk scraping journals per category. This led us to have 14 different data frames for every category that only included title and individual journal link. In some data frames, the name of the category was appended.



*Part 1 of scraped journal details*

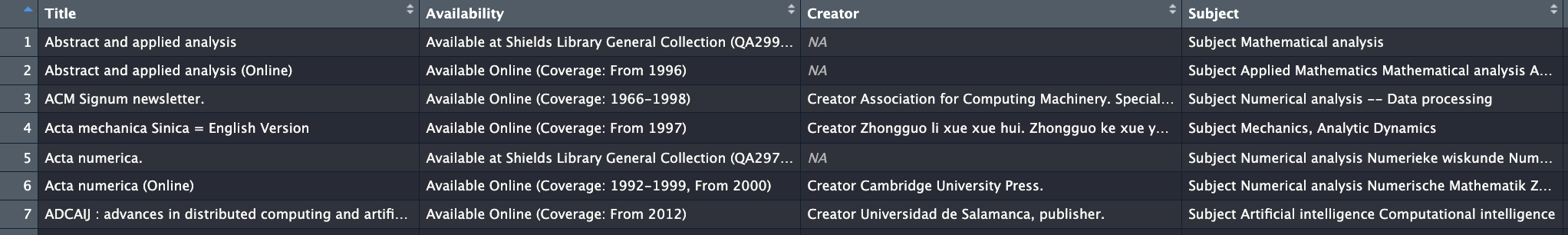
### Individual Journal details

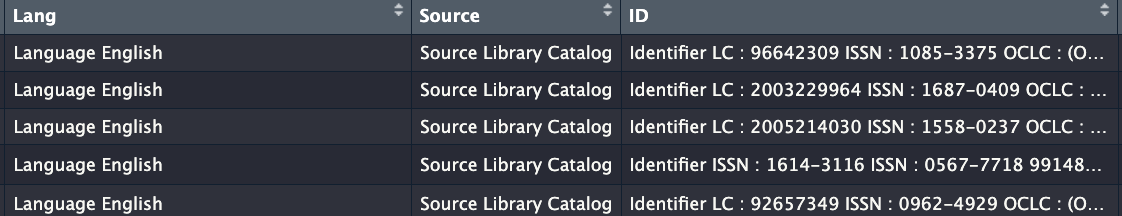
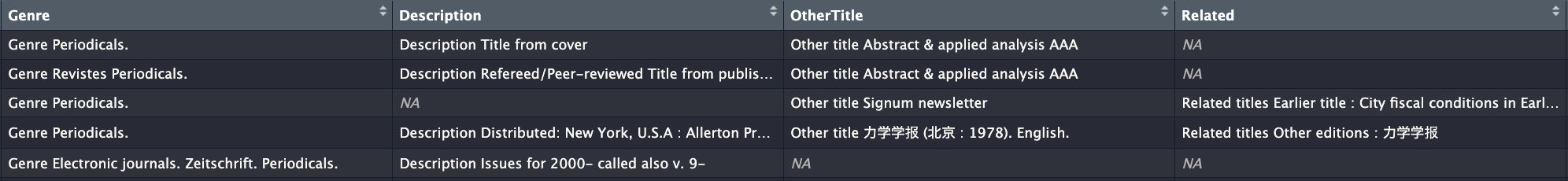
The individual journal link is important since the individual journal page displays the detailed information we desire for the journal. It is from the journal page where we extracted the info for the full detailed journal data frame.



*Individual journal page (redacted)*

Now, for the full detailed journal data frame, we coded a for-loop which loaded every individual journal link and scraped the desired journal details, the most important being journal availability (online/in-person), ISSNs, and subject. Look below at a sample of the collected data from the Engineering Category.





*Part 1 of scraped journal details*

Despite our code working for the full detailed journal data frame, we limited ourselves to scraping only one out of the 14 journal categories, the aforementioned engineering category. This was because scraping only 4,000 out of the 10,000 journals in the engineering category took an enormous amount of time since the code only scrapes one journal at a time, each taking around 12-16 seconds to complete. Nonetheless, we were still able to get important information from 4,000 journals.

### Extracting ISSNs

Continuing, with the full detailed journal data frame, we decided to add an extra column with just the ISSNs of every journal. For doing so, we used the “ID” variable in the data frame which contains all various types of identification, including for not limited to the ISSN.

The format of every ID entry looks something like below:

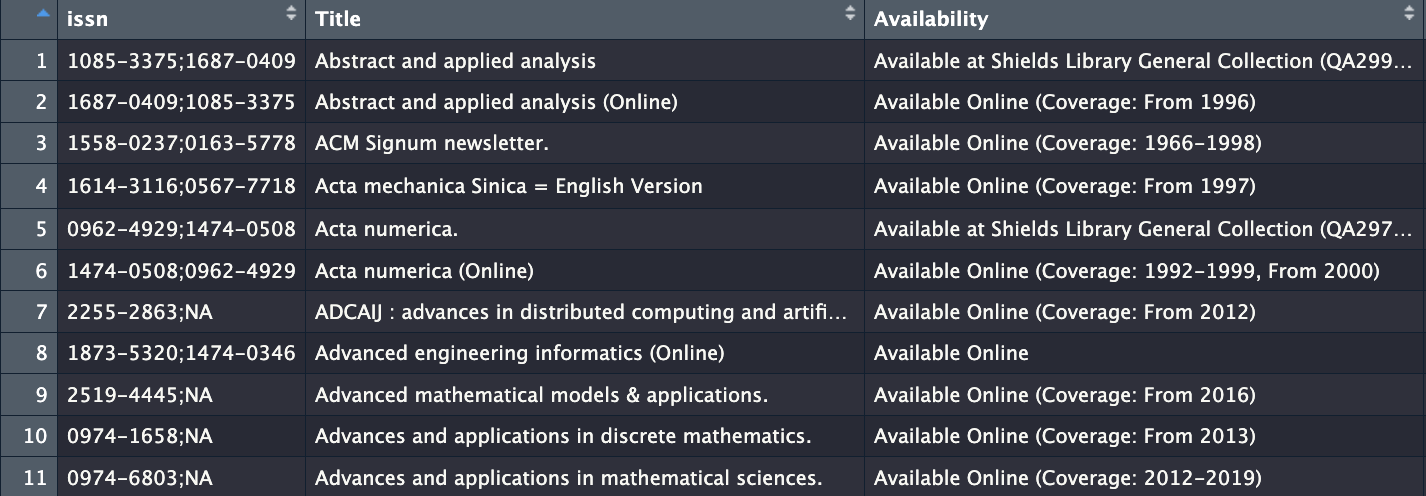
**"Identifier\nLC : 96642309\nISSN : 1085-3375\nOCLC : (OCoLC)33333271\nISSN : 1687-0409\n990022160050403126"**

Using a function, we extracted the digits in front of the “ISSN” string until the “\n” line break and added a “;” delimiter between ISSNs if the journal had more than one.

**"Identifier\nLC : 96642309\nISSN : 1085-3375\nOCLC : (OCoLC)33333271\nISSN : 1687-0409\n990022160050403126"**

We countered issues when the last digit of the ISSN was an “X”. Whenever the ISSN ends in an “X”, the code drops the letter and outputs a 7-digit ISSN instead of an 8-digits. We need to fix this in the future.

Afterwards, we appended a vector with the ISSNs to the full detailed journal data frame. We recommend appending a column as a vector and not a data frame because this can cause issues displaying the data frame correctly.



*ISSN column appended to the left*

The 14 categories data frames and the full detailed journal data frame for the engineering journals were uploaded to the UC Davis datalab CAIEF google drive for future reference.

### Gathering the Journal Corpus

Using the ISSNs collected from the journals available online, we started collecting the text data of every journal. We did so with the help of one of our Domain Lead, Professor Emily Merchant, who provided us with an Rmarkdown file that allows us to gather text data from Elsevier, and other journals, using the ISSNs. Due to the surplus of scraped journals from the UC Davis library, it would be unrealistic to run the code on our personal computers, which is how we decided to use the Data Science server to run our code.

Since a server is unable to read Rmarkdown files (similar to Jupyter notebooks), two of our undergraduate student researchers translated the code into an R-Script so it could run on Bash. We began by reading through the provided Rmarkdown files and extracting the lines of code relevant for our task. We created an R script with the functions and commands, which enabled us to download the metadata associated with each journal and the articles themselves.

Technologies / Softwares:

1. R
2. DataSci Server
3. Notion
4. Google Drive
   1. Google Sheets
   2. Google Docs
5. Github

Libraries

1. Tidyverse
2. httr
3. jsonlited
4. RSelenium
5. wdman
6. netstat

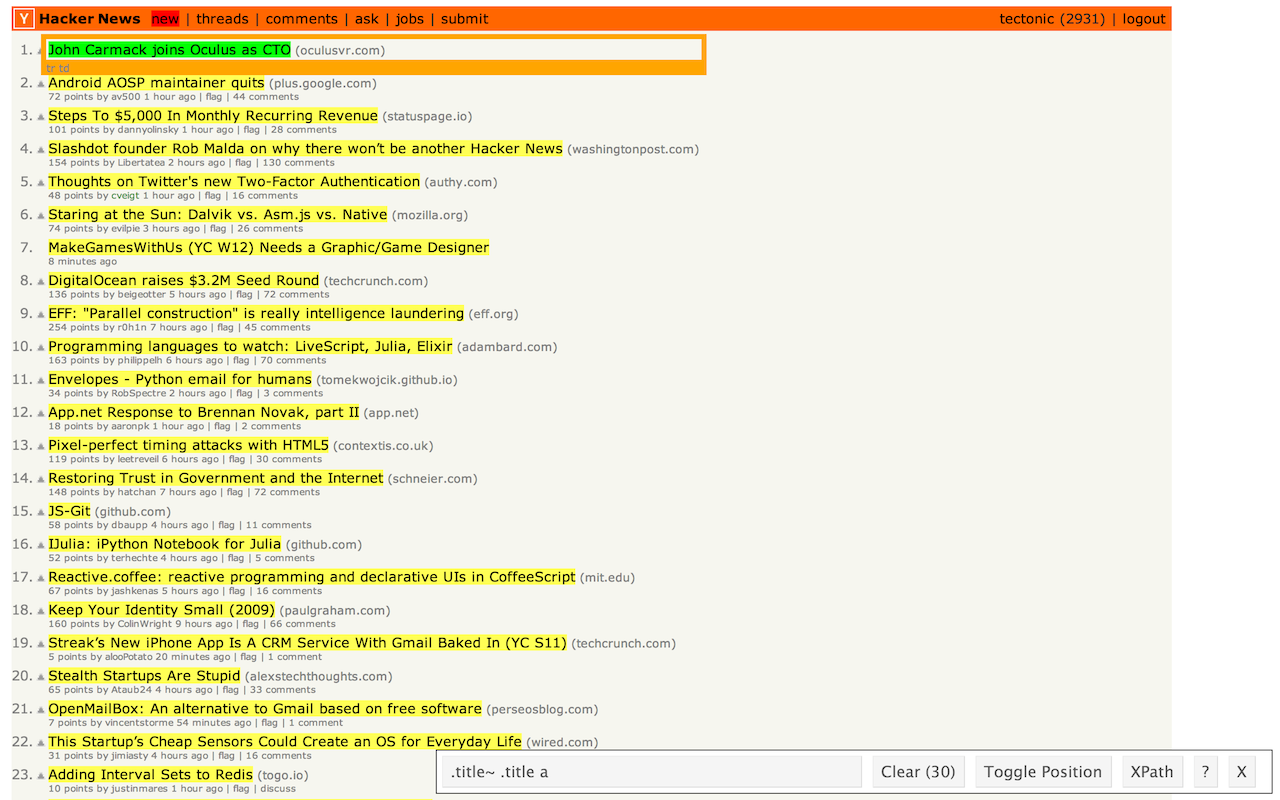
## Research Outcomes

1. Created Web Crawler to collect journal metadata:
   1. Title, Journal URL, Online Availability, Publisher, Subjects, languages, and ISSNs.
2. Used collected ISSNs to get the actual contents of the journal articles using CrossRefs API and other publisher’s APIs

## Discussion

### Locating specific tags and other HTML elements

A key part of the project was identifying the tags within the html elements we wanted to scrap. A common practice is using the “inspect element” that lets you identify the xpath and tags of a specific element. However, we found the google chrome extension called [SelectorGadget](https://chromewebstore.google.com/detail/selectorgadget/mhjhnkcfbdhnjickkkdbjoemdmbfginb?hl=en) to be extremely useful when identifying xpaths. With SelectorGadget you can just press on the desired element and it will highlight all similar elements.



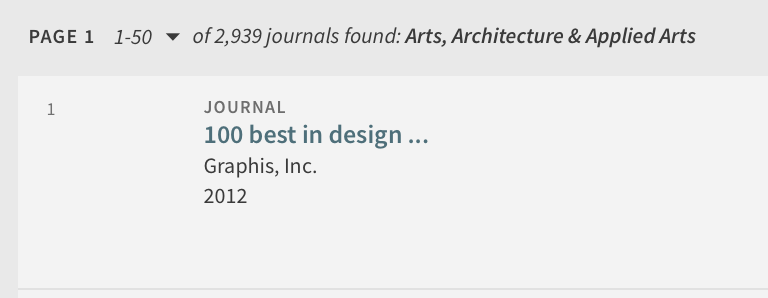
Highlighted text can be scrapped by the bottom xpath

Afterwards, you can click any of the highlighted elements if you wish to unselect a certain type of element and narrow down what you wish to scrap. Once you have selected the desired elements you can copy the xpath and use it with the RSelenium library to scrape the text. If you are a beginner with HTML tags, we would still suggest attempting to manually find the xpath or tag through “inspect element” as there are instances where SelectorGadget will not extract the desired information.

### Page switching

When scraping the journal title of a single journal category, we had to code create page switching code that would scrape all the journal entries per page and then click the next page button. To do so, we implement a “click next page” button in the code for the scraper that would identify where to click to switch to the next page. This approach caused a problem when switching sub-categories. Once the last page was reached, the code would break when attempting to switch to the next page.

To prevent this, code was implemented that calculated how many pages were available for the specific category. We took the total number of journals by extracting the text element that contained it and filtered only the digits and then divided by the number of journals displayed in each page; 50-journals.



This allows the web crawler to know how many pages to switch through before switching to the next category.

### Incomplete scrapping

Unfortunately, every sub-section of the categories only displays up to 2000 journal entries (equivalent to 40 pages) due to website limitations outside of our control. Many sub categories had more than 2000 journals thus our web crawler was not able to get every single journal under the UC Davis Library site. Approaching the project through API web scraping next time might yield better results.

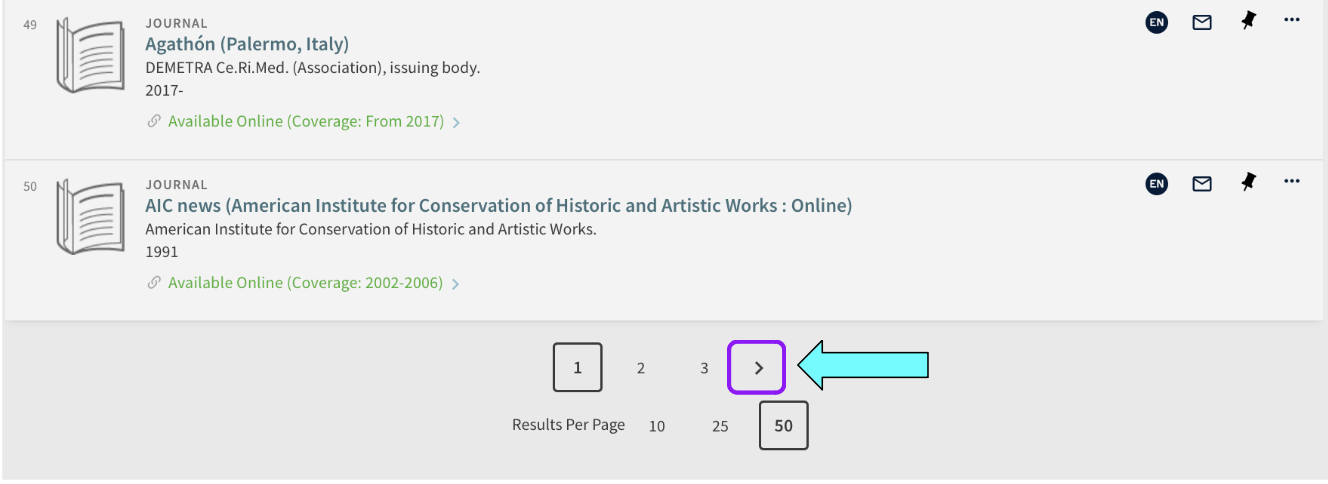
## Future directions

### Unique subject column

Towards the end of the project, we found out that the UC Davis library has an API to access all of the journals, making most of our research project inconsequential. However, as undergraduate research students we learned a lot of useful data scraping techniques applicable for other projects.

### Improving the page switching mechanism

Currently, to switch to the next page, a string that identifies the next page number (e.g. “pagebutton2”) is updated every time we switch pages. For example, before switching to page 3, we update the identifier string to “pagebutton3.” However, this code is unnecessary if we were to select the “next” button instead. By implementing this, the code can be greatly shortened and become clearer to understand.



*Bottom of the page when looking at journal categories*

### 

### Missing letter from the ISSNs

An assumption was made when filtering the ISSNs front he ID full detailed column which led to some missing characters. The assumption made was that all ISSNs are 8 digits; this is not entirely correct. While some ISSNs are 8 digits (1234-5678), some are 7 digits followed by an “X” (1234-567X). Our code was instructed to only extract digits, thus many of our ISSNs look like this: 1234-567\_ instead of 1234-567X.

In future iterations of this project, the ISSN filtering portion of the code needs to be modified so it extracts the full 8-character string mix of numbers and letters.

### Using an API instead of a web crawler

As it was mentioned before, an API is available front he UC Davis library to easily extract the journal ISSNs needed. We would have known earlier if we had asked the librarian towards the beginning of the project instead of so late.

## Training outcomes

* Web scraping skills (e.g. Selenium)
* Running code on the Data Sci Server
* Logging
* Using remote servers

All in all, however, the process was a great learning experience as we were able to be hands-on in the data collection/cleaning process and see just how important and inconsistent data can be. This helps us better grasp how important the data collection process is but also how challenging it can be to plan far into the future so that all the data is used for the right purposes.

## Appendix

[Original Project Scope](https://docs.google.com/document/d/1VyWPQVGJC_a4FJScfHklJQRFe9LeQBoqCMZ2CMURcWg/edit#heading=h.moeul1hbw9ja)

[Git Hub](https://github.com/datalab-dev/2024_startup_caief)

[ReadMe](https://github.com/datalab-dev/2024_startup_caief/blob/7d4f7b7e9405eccd214945fb019a1af73f718387/README.md)

[Drive](https://drive.google.com/drive/u/1/folders/1RQklI6wSTTjmrLQ41Aj8NSv8AtPhfwy5)