Computer-Assisted Language Comparison in Practice

Tutorials on Computational Approaches to the History and Diversity of Languages

Leveraging JavaScript, jQuery, and ChatGPT for Data Extraction from Web Tables

Alessandro Mantelli and Michele Pulini
Department of Asian and North African Studies
Ca' Foscari University of Venice

This case study explores the combined use of JavaScript, jQuery, and the AI-driven tool ChatGPT to efficiently extract data from HTML tables, specifically focusing on a website containing Middle Chinese and Old Chinese readings of characters. The study provides a step-by-step guide for accessing the website, utilizing browser developer tools, implementing JavaScript and jQuery code, and leveraging ChatGPT to refine the extraction process. By employing this methodology, the extraction of Chinese characters and their corresponding readings from an HTML table was automated, saving time and effort. The resulting data was then imported into a Google Sheets document for further analysis. This case study highlights the potential of AI-driven tools to enhance web development tasks and streamline data extraction processes, demonstrating their value for both technical and non-technical users.

1 Introduction

Accessing and extracting data from websites is an essential skill for researchers and professionals alike (Bozzon et al. 2018). However, this task can be time-consuming and challenging, especially when the data is presented in a format that does not readily support downloading or exporting. This article presents a case study in which JavaScript, jQuery, and ChatGPT were combined to address a data extraction problem, demonstrating the potential for AI-driven solutions to streamline and enhance the development process. We provide a detailed walkthrough of each step, from accessing the website and using browser developer tools to implementing JavaScript and jQuery code and leveraging ChatGPT's AI capabilities. The primary aim of this case study is to provide a comprehensive understanding of the techniques used, as well as to encourage further exploration and application of these methods in various contexts.

Computer-Assisted Language Comparison in Practice
Volume 6, Number 5
https://calc.hypotheses.org/6079/

Published on 22/05/2023

2 The Problem to Solve

The problem encountered was the need to to fetch data from a website (http://dighl.github.io/tls.html) containing a repository of Middle Chinese and Old Chinese readings of characters. The website itself served as a tool for rather informal private use, created by Johann-Mattis List several years ago, providing access to parts of the data presented in the Thesaurus Linguae Sericae, a website initiated by Christoph Harbsmeier, that has been under development for many years and can currently be found online at http://tkb.zinbun.kyoto-u.ac.jp/, but its fate is unclear, since it seems that development has been inactive since 2020. The website by List displays the data in an HTML table format but does not offer an option to download or export the information. The data is organized in tabular format, with each row representing a different character, and various columns containing information such as pronunciation, tone, and rhyme class. While this layout is helpful for browsing and comparing characters, it poses a challenge when attempting to extract specific data for further analysis or storage (Yates and Stent 2018). Manually copying each character and its corresponding reading is laborious, so that a strategy to automate the data extraction process was needed.

The task at hand involved fetching the Chinese characters from the first column of the table and their corresponding Old Chinese (OCBS) readings from the eleventh column (Baxter and Sagart 2014). The goal was to create a Google Sheets document with two columns, one for the characters and another one for their Middle Chinese and the Old Chinese readings. However, the lack of a built-in download or export feature on the website necessitated a more creative approach.

The approach taken involved several steps, starting with accessing the website and utilizing browser developer tools to identify the HTML elements containing the desired data. JavaScript and jQuery were then employed to extract the information. ChatGPT was used to write the code that allows the extraction process. The final output was a list of characters and readings that could be easily copied into a Google Sheets document, saving considerable time and effort.

3 The Developer Tools (DevTools)

Developer tools are integrated into modern web browsers, allowing users to inspect and modify HTML, CSS, and JavaScript elements on a web page. These tools are invaluable for developers and researchers, enabling them to identify elements within a web page's structure and manipulate them to achieve their desired outcomes (Google n.d.). In this case, Google Chrome's DevTools was used to identify the dynamically generated HTML table containing the Chinese characters and readings.

Accessing the DevTools in Google Chrome is straightforward; one can either right-click on the web page and select "Inspect" or use the keyboard shortcut "Ctrl+Shift+I"

(or “Cmd+Shift+I” on macOS). Once the DevTools panel is open, it offers several features to help users navigate and interact with the web page’s structure. The “Elements” tab displays the HTML and CSS code, allowing users to select specific elements and view or edit their properties. In this case, the “Elements” tab was used to locate the HTML table containing the data.

JavaScript (https://developer.mozilla.org/en-US/docs/Learn/JavaScript), is a powerful and versatile programming language that empowers developers to create dynamic and interactive web pages by manipulating Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) elements in real-time. As a client-side language, JavaScript operates within the user’s web browser, making it an essential tool for extracting data from HTML after a web page has been rendered (Mozilla Developer Network 2021). This functionality is beneficial to both technical and non-technical users, as it allows web pages to respond quickly to user interactions and provide a seamless browsing experience.

JavaScript code can be executed within a web browser’s built-in developer tools by accessing the “Console” tab. This feature is useful for both debugging purposes, as it can display JavaScript-related errors, and for injecting new code into the HTML page. JavaScript’s potency is especially evident in modern web development, as seen in the Single Page Application (SPA) architecture. In SPA, JavaScript is responsible for creating the entire HTML page, while the server-side language’s primary function is to return raw data from the database. This approach enhances the user experience by minimizing page reloads and providing a more fluid interaction with the web application. To extract the necessary information in this case study, we employed a JavaScript code snippet that was inserted directly into the “Console” area of Google Chrome’s Developer Tools. This method enables users to interact with and modify web content in real-time.

In the context of this case study, JavaScript was employed in the form of jQuery (https://jquery.com/), a popular library that simplifies the process of traversing HTML tables and extracting desired data programmatically. For individuals without a strong background in information technology, jQuery can be seen as a valuable resource for streamlining the manipulation of web content, making it more accessible to a broader audience. jQuery is designed to simplify various tasks, including HTML document traversal, manipulation, and event handling (jQuery Foundation, n.d.). Its concise and expressive syntax facilitates the creation of complex scripts and simplifies common tasks. In this case study, jQuery was utilized to streamline the data extraction process further. jQuery’s primary strength is its ability to simplify DOM manipulation tasks by providing a powerful set of functions for selecting elements, traversing the DOM tree, and modifying element properties or content. By leveraging jQuery, it was possible to rewrite the JavaScript function with a more concise syntax and improved readability. For instance, the more flexible jQuery selector $('.classname') replaced the

getElementsByTagName() vanilla JavaScript method, which allowed targeting of the table element directly. Additionally, jQuery's each() function could loop through the table rows and extract Chinese characters and OCBS readings from the corresponding cells . Although modern JavaScript offers the document.querySelector statement, similar to jQuery's $, jQuery simplifies HTML element traversal to a greater extent. The use of jQuery also enabled the usage of the text() method to retrieve the text content of the targeted cells, ensuring that only the desired data was extracted and stored in the results array. Thus, the use of jQuery in this case study not only simplified the code but also enhanced its overall efficiency and maintainability. However, even with jQuery, composing the code requires a significant time investment. After several tests using jQuery to identify the pertinent HTML elements containing the desired information, we opted to leverage ChatGPT to generate the code.

In this case study, ChatGPT (https://openai.com/blog/chatgpt) played a crucial role in enhancing the efficiency of the data extraction process. Assistance from ChatGPT was needed to optimize the JavaScript and jQuery code and provide insights on how to further improve the data extraction procedure (Brown et al. 2020). ChatGPT's AI-driven capabilities enabled it to analyze the code, offering suggestions for optimization and clarification where necessary. By leveraging ChatGPT's assistance, it was possible to refine JavaScript and jQuery functions, ensuring that they were both efficient and maintainable. We utilized ChatGPT via the conventional web-based chat system, delineating the objective of the project and our requirements. The inquiry was formulated as follows:

> *Write jQuery code that retrieves the text content of each td element in the first row of the "datatable" table class, as well as the text content of each td element in the eleventh row of the same table. The resulting output should be in a format compatible with Excel and follow the structure: "Character 1, Row 1: [content], Character 1, Row 11: [content], Character 2, Row 1: [content], Character 2, Row 11: [content], …" The HTML markup of the "datatable" table class is provided below.*

```
<table class="datatable">
 <tbody>
  <tr>
    <th>HANZI</th>
    <td>夏</td>
    <td>目</td>
    <td>探</td>
  </tr>
```

```
  <tr>
    <th>PINYIN</th>
    <td>xià</td>
    <td>mù</td>
    <td>tàn/tān</td>
  </tr>
  <tr>
    <th>BAXTER</th>
    <td>haeH</td>
    <td>mjuwk</td>
    <td>thom</td>
  </tr>
  <tr>
    <th>SHENGMU</th>
    <td>匣</td>
    <td>明</td>
    <td>透</td>
  </tr>
  <tr>
    <th>KAIHE</th>
    <td>開</td>
    <td>合</td>
    <td>開</td>
  </tr>
  <tr>
    <th>DENG</th>
    <td>二</td>
    <td>三</td>
    <td>一</td>
  </tr>
  <tr>
    <th>YUNBU</th>
    <td>魚</td>
    <td>覺1</td>
    <td>侵3</td>
  </tr>
  <tr>
    <th>SHENGDIAO</th>
    <td>去</td>
    <td>入</td>
    <td>平</td>
  </tr>
  <tr>
```

```
      <th>FANQIE</th>
      <td>胡駕</td>
      <td>莫六</td>
      <td>他含</td>
    </tr>
    <tr>
      <th>TLS</th>
      <td><a target="_blank" href="http://tls.uni-
hd.de/procSearch/procSearchPhon.lasso?-MaxRecords=1&amp;amp;-
SkipRecords=10961">10961</a></td>
      <td><a target="_blank" href="http://tls.uni-
hd.de/procSearch/procSearchPhon.lasso?-MaxRecords=1&amp;amp;-
SkipRecords=1186">1186</a></td><td><a target="_blank" href="http://tls.uni-
hd.de/procSearch/procSearchPhon.lasso?-MaxRecords=1&amp;amp;-
SkipRecords=14190">14190</a></td>
    </tr>
    <tr>
      <th>OCBS</th>
      <td>*[g]ˤraʔ</td>
      <td>*C.m(r)[u]k</td>
      <td>*ɬˤ[ə]m</td>
    </tr>
    <tr>
      <th>Pan Wuyun</th>
      <td>graas</td>
      <td>mug</td>
      <td>kh-luum</td>
    </tr>
    <tr>
      <th>Gloss (OCBS)</th>
      <td>summer</td>
      <td>eye</td>
      <td>put the hand into and test</td>
    </tr>
  </tbody>
</table>
```

The code generated by ChatGPT was the following:

```
// Define an empty array to store the results
var result = [];
```

```
// Find the first and eleventh rows of the table
var $firstRow = $('.datatable tr:first');
var $eleventhRow = $('.datatable tr:eq(10)');

// Loop through each cell in the first row
$firstRow.find('td').each(function(index, element) {

// Get the text content of the cell and store it in a variable
var firstChar = $(element).text();

// Get the corresponding cell in the eleventh row
var $eleventhCell = $eleventhRow.find('td:eq(' + index + ')');

// Get the text content of the eleventh cell and store it in a variable
var eleventhChar = $eleventhCell.text();

// Add the characters to the result array
result.push(firstChar + ', ' + eleventhChar);
});

// Output the result as a comma-separated string that can be pasted into Excel
console.log(result.join('\n'));
```

This code snippet performs the following actions:

- Creates an empty array called ‘result’ to store the extracted data.
- Selects the first and eleventh rows of the table using jQuery’s $('.datatable tr:first') and $('.datatable tr:eq(10)') methods, respectively.
- Loops through each cell in the first row using the .find() and .each() methods.
- Retrieves the text content of the first row’s cell and the corresponding cell in the eleventh row.
- Appends the extracted characters to the result array.
- Outputs the result array as a comma-separated string, which can be copied and pasted into a csv file.

4 Final Activities and Analysis

Upon executing the ChatGPT-generated code within the Developer Tools, a formatted list was produced, resembling the following structure:

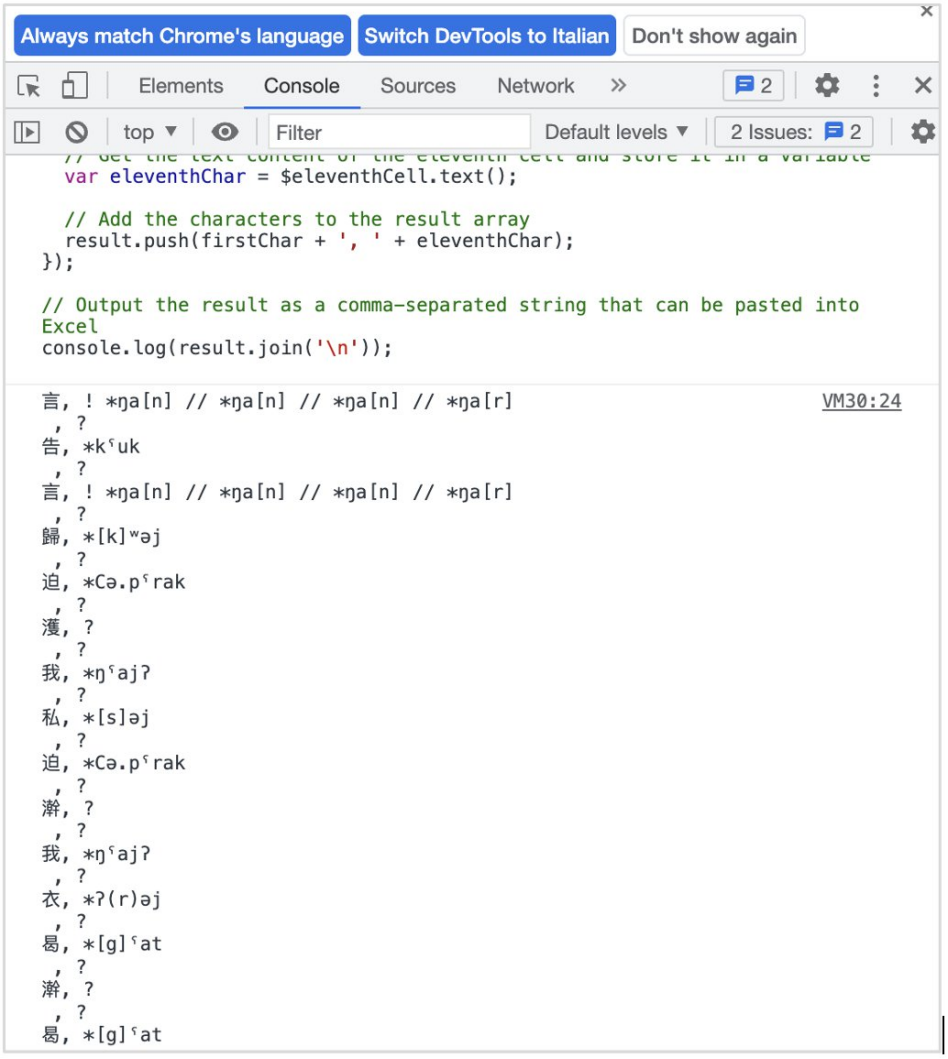

Figure 1: Screenshot of the debugger in Chrome.

Subsequently, this list was transferred to a text editor, which is a software application commonly employed by developers for editing plain text files without the inclusion of complex formatting found in programs like Microsoft Word. Utilizing a text editor in this manner ensures the preservation of the original text and facilitates the generation of a CSV file, or comma-separated values file. This type of file is easily interpreted and converted into a table by spreadsheet programs such as Microsoft Excel or Google Sheets, as shown in Figure 2.

In the current case study, the objective was to import the CSV file into a Google Sheets document. Google Sheets is a web-based spreadsheet application that allows users to create, edit, and collaborate on spreadsheets in real-time. This platform was chosen for its accessibility and compatibility with the CSV format, as well as its potential for collaboration and sharing as it is an on-line tool.

Upon successful importation of the CSV file into Google Sheets, the data was transformed into a structured table, enabling further analysis and interpretation of the extracted information. This final step marked the completion of the data extraction

process, bridging the gap between the initial JavaScript code execution and the ultimate presentation of the data in an accessible and user-friendly format.

	A	B
1	Character	OC
2	言	! *ŋa[n] // *ŋa[n] //
3	告	*kˤuk
4	言	! *ŋa[n] // *ŋa[n] //
5	歸	*[k]ʷəj
6	迫	*Cə.pˤrak
7	濩	*Gʷˤak / *Gˤa-s /
8	我	*ŋˤaj?
9	私	*[s]əj
10	迫	*Cə.pˤrak
11	澣	*Gˤon?
12	我	*ŋˤaj?
13	衣	*?(r)əj
14	曷	*[g]ˤat
15	澣	*Gˤon?
16	曷	*[g]ˤat

Figure 2: Spreadsheet result of the workflow.

5 Conclusions

This case study has demonstrated the efficacy of combining JavaScript, jQuery, and AI-driven tools such as ChatGPT to tackle intricate data extraction challenges. By harnessing these technologies, an efficient and maintainable solution was developed that automated the process of retrieving Chinese characters and their corresponding OCBS readings from an HTML table, thereby saving considerable time and effort (Bozzon et al. 2018).

The article offered a thorough walkthrough of each stage in the process, from accessing the website and utilizing browser developer tools to implementing JavaScript and jQuery code and seeking guidance from ChatGPT. The objective was to provide a

comprehensive understanding of the techniques employed and encourage further exploration and application of these methods in various contexts.

This case study underscores the potential for AI-driven solutions to augment web development tasks and streamline data extraction processes. By incorporating ChatGPT into the workflow, not only developers but also scholars with a humanities orientation can benefit from the AI's capacity to write code, supply context, and deliver valuable insights. As AI technologies continue to progress and become increasingly accessible, the possibilities for their application in web development and data extraction are virtually boundless.

In conclusion, the combination of JavaScript, jQuery, and ChatGPT has proven to be an effective strategy for addressing data extraction challenges in a web-based context. This case study serves as an exemplar of how developers and researchers can harness the power of these technologies to devise innovative and efficient solutions for extracting and analyzing data from web pages.

References

Baxter, William H., and Laurent Sagart. 2014. Old Chinese: A New Reconstruction. Oxford: Oxford University Press.

Bozzon, Alessandro, Marco Brambilla, and Stefano Ceri. (2018). "Web Information Retrieval." In Search Computing, 3-19. Springer, Cham.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. (2020. "Language models are few-shot learners." In Advances in Neural Information Processing Systems 33, 1877-1901.

Digital Humanities Lab. "TLS: A Digital Humanities Project." Accessed May 19, 2023. http://dighl.github.io/tls.html.

jQuery. "jQuery: The Write Less, Do More, JavaScript Library." Accessed May 19, 2023. https://jquery.com/.

Mozilla. "What is JavaScript?". MDN Web Docs. Accessed May 19, 2023. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

OpenAI. "ChatGPT: Language Models as Virtual Assistants." OpenAI Blog. Accessed May 19, 2023. https://openai.com/blog/chatgpt.

Yates, Alexander, and Amanda Stent. (2018). "Web Data Extraction, Applications, and Techniques: A Survey." Knowledge and Information Systems 53(2): 495-551.