Data Analytics

Japanese learning database

November, 2025 Victor Daube

General introduction

This project was created to help me learn Japanese. When I was told I could do whatever I wanted for my last project, I wanted to try doing something that could be useful to me. I went to Japan in 2023 and I'm planning to go back in 2026 but I'd like to be able to speak and understand a bit of Japanese. Before working on this project, I tried to learn Japanese by myself and quickly realized that there are multiple apps, websites, YouTube channels … that had lots of information but it was always missing some information and it left me frustrated and I wondered if I could gather all the information I wanted in one place and access it when I needed it.
My ultimate goal is to create a mobile app where I can access my database and this project is a great first step : designing and creating the database with the information I want in it.

Japanese writing system introduction

The Japanese language uses a hybrid writing system composed of several distinct scripts, each serving a different linguistic purpose. Understanding these scripts is essential for interpreting Japanese text and for following the data structures used in this project.

Hiragana (ひらがな)

Hiragana is a phonetic syllabary consisting of 46 basic characters, each representing a specific syllable. It is used for native Japanese words, grammatical particles, verb endings, and for providing pronunciation when kanji are not used. Hiragana is often the first script learned by Japanese children.

Katakana (カタカナ)

Katakana is another phonetic syllabary with the same set of sounds as hiragana but written with distinct angular shapes. It is primarily used for foreign loanwords, scientific terms, emphasis (similar to italics), and the names of some animals or plants.

Because hiragana and katakana share pronunciations, they are collectively referred to as **kana**.

Kanji (漢字)

Kanji are logographic characters originating from Chinese. Each kanji carries meaning and usually has multiple readings depending on context:

- **On'yomi or on reading**: Sino-Japanese readings, often used in compound words.
- **Kun'yomi or kun reading**: Native Japanese readings, often used when the kanji appears alone or in traditional vocabulary.

Many words combine several kanji, each contributing semantic information. Kanji can vary greatly in complexity: simple characters may have 1–3 strokes, whereas more intricate ones may exceed 20 strokes.

Furigana (ふりがな)

Furigana are small hiragana written above or beside kanji to indicate pronunciation. They are commonly used in dictionaries, children's books, or learning materials. In datasets, furigana allow algorithms—or readers unfamiliar with a kanji—to interpret how a word is read.

Rōmaji (ローマ字)

Rōmaji is the romanization of Japanese sounds using the Latin alphabet. It is used for teaching pronunciation, text input on keyboards, and accessibility for non-Japanese speakers. While not part of native Japanese writing, it is essential for linguistic preprocessing and data manipulation.

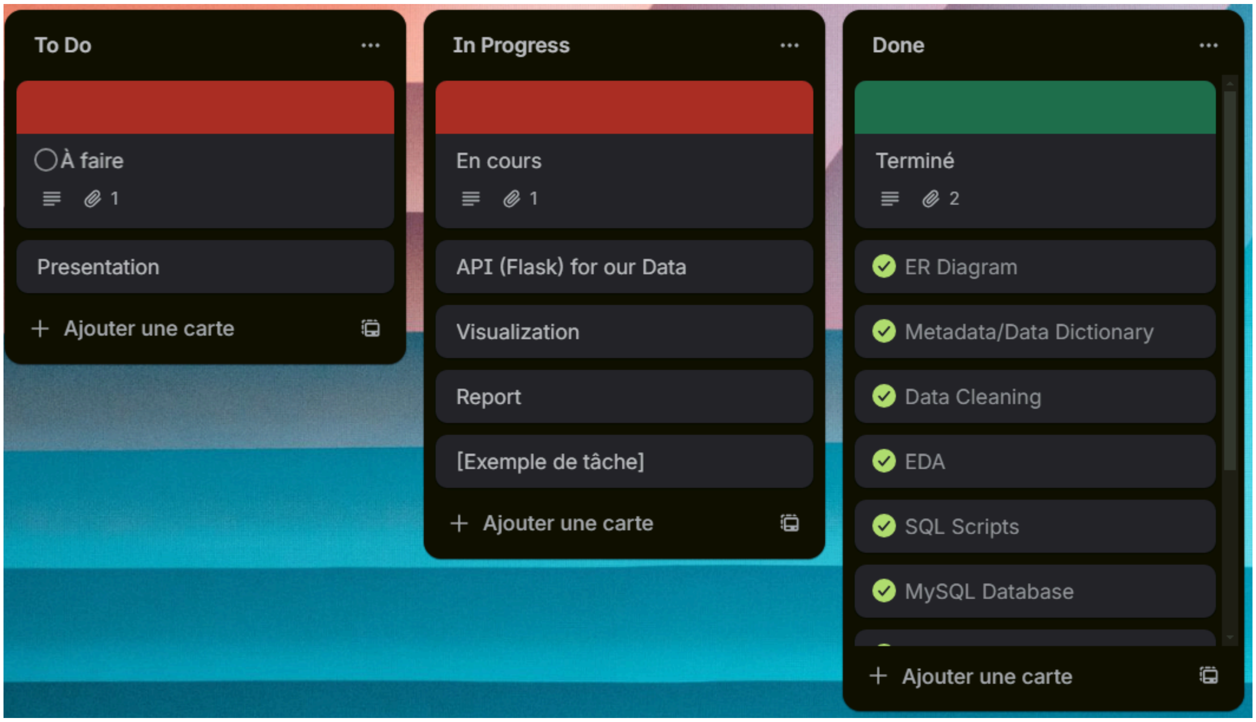
JLPT

Japanese Language Proficiency Test (or JLPT) is a test made for foreigners speaking Japanese. It's the equivalent of TOEIC/TOEFL that we have for English. It has 5 levels :

- N5 – Beginner Level (Basic vocabulary and grammar)
- N4 – Upper Beginner Level (More grammar and kanji)
- N3 – Intermediate Level (Conversational Japanese)
- N2 – Advanced Level (Business-level proficiency)
- N1 – Near-Native Level (Highly complex Japanese)

Project Planning

I used Trello for project planning. Here is a screenshot from a few days ago :



I wrote on a paper sheet daily to-do lists, I mostly used Trello to confirm that I finished a step and to see what I still needed to work on.
Here is the link to my board :
https://trello.com/invite/b/68df7c8d74a7fe35f67b7c40/ATTI5603f1f8f9b9f07ec0289e7933ac1670606450F3/final-project

Data sources and data collection

I used 3 data sources and 3 different ways to import my data :

Webscraped this website for kana (hiragana and katakana) with romaji:
https://www.sljfaq.org/afaq/romanization-table.html

Used Jisho API to retrieve data about kanji:
https://jisho.org/ & https://pypi.org/project/jisho-api/

Downloaded a csv file containing JLPT Vocabulary:
https://www.kaggle.com/datasets/robinpourtaud/jlpt-words-by-level

I didn’t use BigQuery to fetch data because I had enough but later on, I loaded a denormalized dataset to BigQuery for ML applications. Jisho API was the most complete database about kanji that I could access via API. It was also easy to use thanks to the Python library available. Kana data is widely available so I tried scraping from a few websites and used the easiest one. The JLPT Dataset has a great usability score on Kaggle and had just the amount of information I needed. It was also pretty clean already.
After importing to pandas DataFrame, here are the two DataFrames I had :

Kana (Web Scraping)		
Field	**Description**	**Type**
kana	The kana scraped (for example カ)	string
romaji	The roman pronunciation of the kana (for example ka)	string

JLPT Vocabulary (CSV File)		
Field	**Description**	**Type**
word	Word composed of several characters	string
furigana	The word’s japanese syllabic writing	string
translation	The word’s english translation	string
JLPT Level	The word’s JLPT Level	string

Then, I used the JLPT Vocabulary table to have a list of all the kanjis used and their frequency. I also counted the number of characters per word. For all of these kanjis, I queried Jisho API to build my kanji table as described below:

Kanji (API Calls)		
Field	**Description**	**Type**
kanji	The kanji	string
count	How many times was this kanji mentioned in JLPT Vocab	int
strokes	How many strokes to write this kanji	int
translation	English translation of this kanji	string
kun_readings	Kun readings of this kanji	string
on_readings	On readings of this kanji	string
radical_basis	Radical of the kanji	string
radical_meaning	Meaning of the kanji’s radical	string

Data Cleaning & Enriching

Cleaning

I removed punctuation at the start and end of my string type columns. The “starting point” of my database was the JLPT Vocabulary CSV file. It has 8129 rows and 4 columns. It had only two rows with null “furigana” values but that’s because the word itself was already all kana. For these two rows, I copied the word column in the furigana column so that would not raise issues later on.

Enriching data

I created a function based on my kana table that would translate kana to romaji. Then, I used this function to create a romaji column in my JLPT Vocabulary DataFrame. Thanks to the same function, I also created a kun_romaji and on_romaji column in my Kanji DataFrame.
Because I couldn’t find this information easily, I created and filled myself a stroke_count column in my Kana DataFrame. This column and the strokes column in my Kanji DataFrame allowed me to calculate stroke_count and store it in a new column of my JLPT Vocabulary DataFrame.
I also added a column num_characters to my JLPT Vocabulary DataFrame that simply counted how many characters were in this word.

Creating a denormalized dataset

Based on the DataFrames I had, I created a denormalized dataset so I could import it easily to BigQuery. First, I added a character column to my JLPT Vocabulary DataFrame containing only one character of a word and all the data associated with the word. Then, I used two left joins on my Kanji and Kana DataFrames so the information about the character itself would appear on each line.

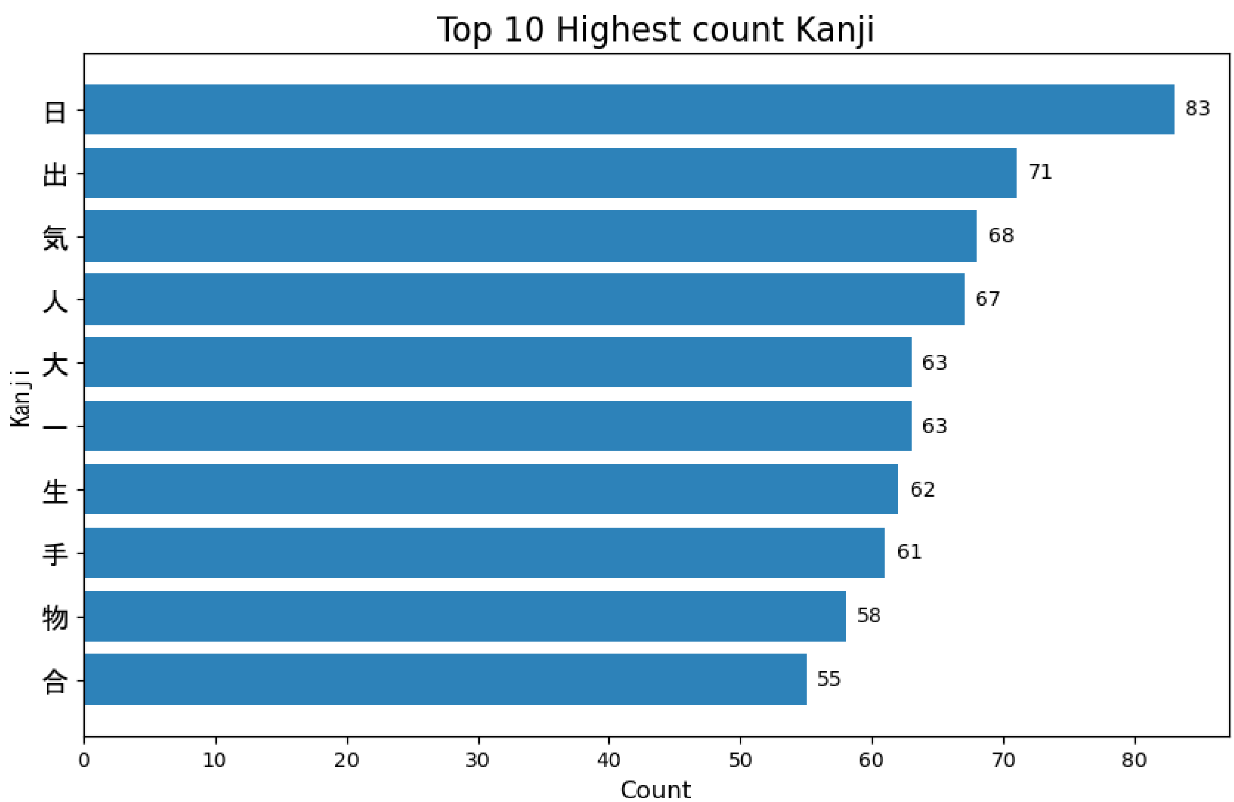
EDA & Visualization

EDA

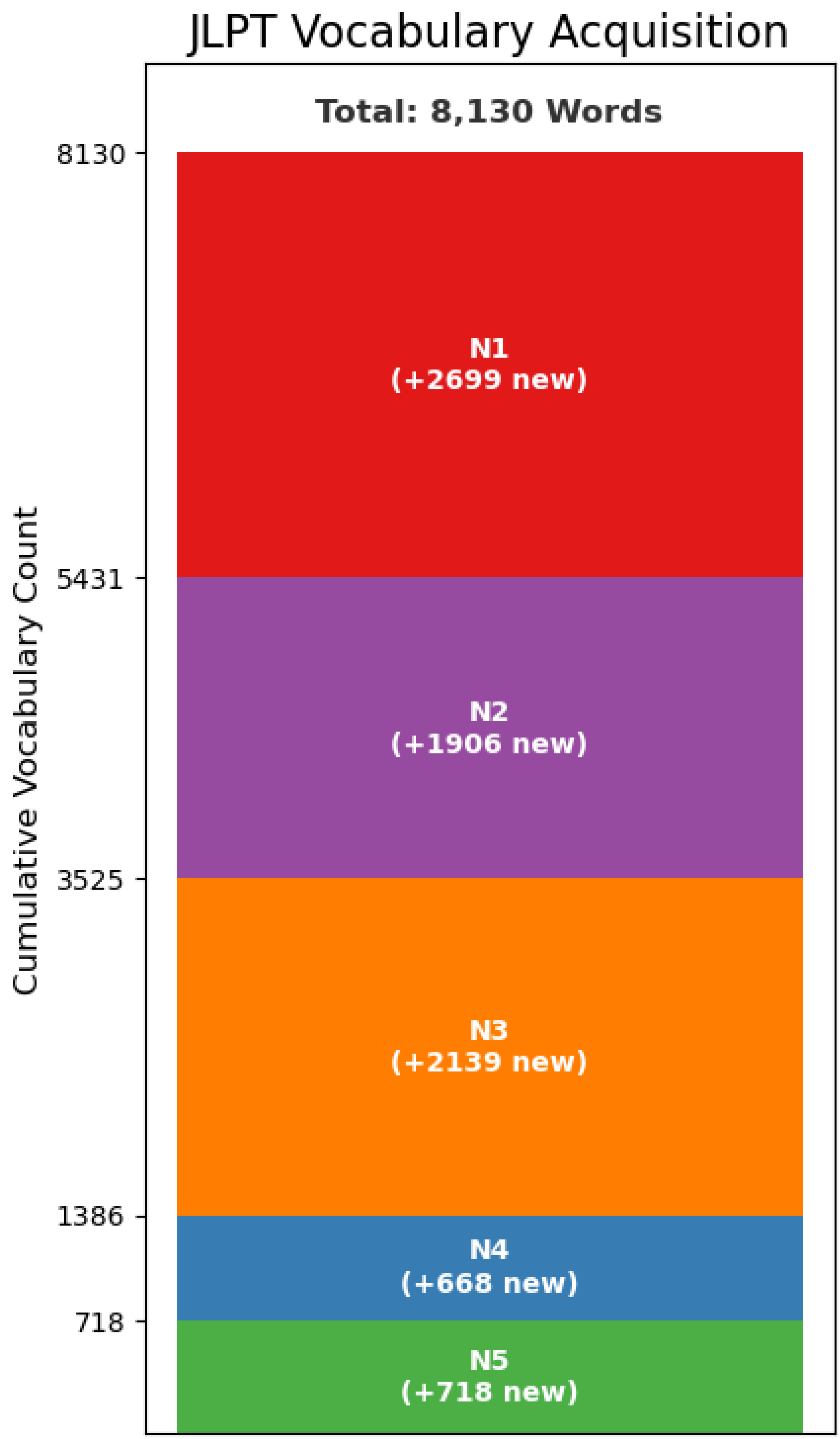
I had 8129 words, 1974 kanji and 218 kana at my disposal. My EDA is detailed in the Visualization and SQL sections.

Visualization

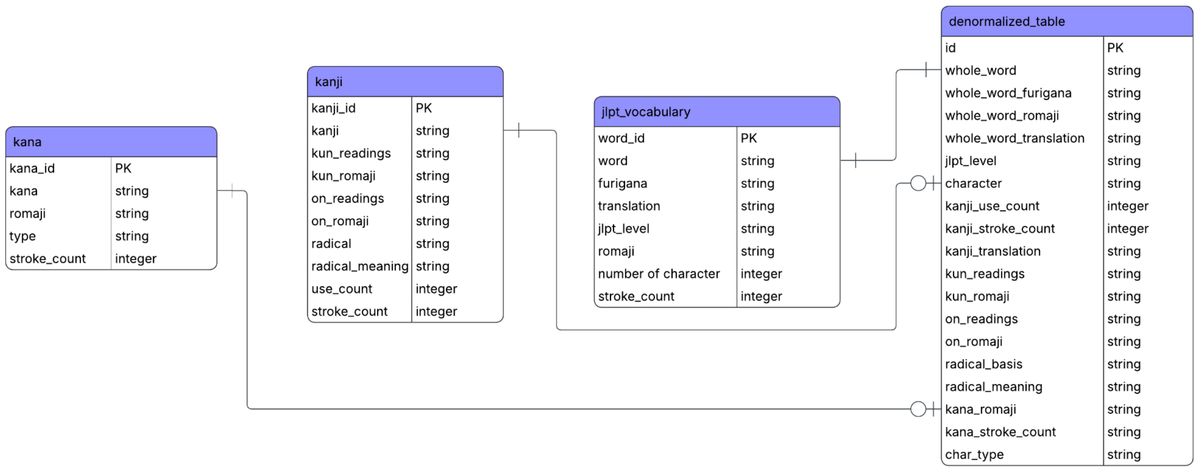
The first visualization I chose to plot shows the 10 most used kanjis in the JLPT Vocabulary dataset. It's a great way to focus on learning common words first.



The second plot shows how difficult it is to go from one JLPT Level to another. It shows how many words you have to learn and how much more it represents compared to the last level.



ER Diagram



This ER Diagram shows the final result of my SQL Tables (but also my pandas DataFrame). As you can see, the denormalized table is linked to all three of my tables. It can contain 0 to 1 kanji or kana and contains 1 word.

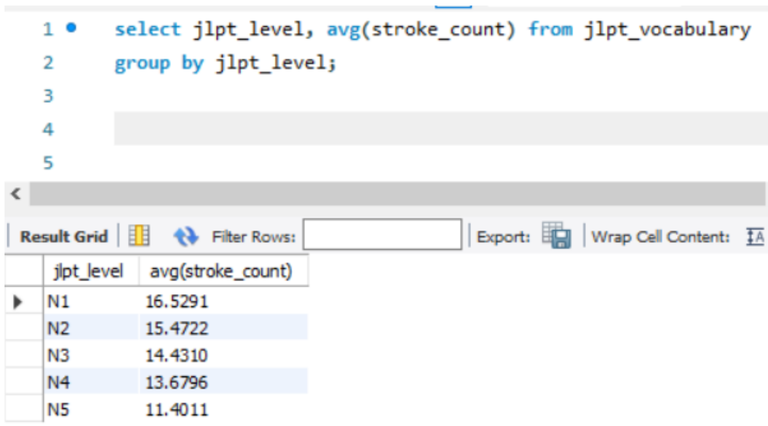
MySQL Database & BigQuery

SQL

I imported my tables to MySQL with sqlalchemy and pymysql libraries. Then, I built 5 queries :

1. Average stroke count of words per JLPT Level
2. Average number of character per JLPT Level
3. Number of words per JLPT Level
4. Number of kanji sharing the same radical and radical meaning
5. Number of times a kanji was used and its translation, ordered from most used to less used

The screenshot below shows the result for the first query for example :

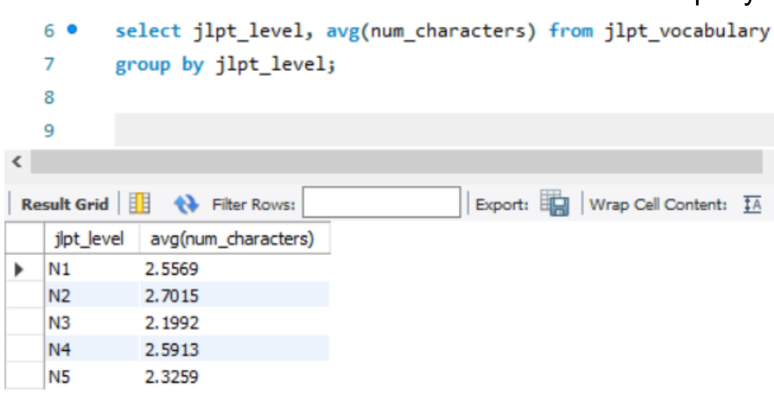


```
select jlpt_level, avg(stroke_count) from jlpt_vocabulary
group by jlpt_level;
```


jlpt_level	avg(stroke_count)
N1	16.5291
N2	15.4722
N3	14.4310
N4	13.6796
N5	11.4011

Thanks to this query, we can conclude that there is a correlation between JLPT Level and stroke count of words. The more strokes you need to write a word, the harder it will be JLPT-wise.

The screenshot below show the result of the second query :



```
select jlpt_level, avg(num_characters) from jlpt_vocabulary
group by jlpt_level;
```


jlpt_level	avg(num_characters)
N1	2.5569
N2	2.7015
N3	2.1992
N4	2.5913
N5	2.3259

There is no correlation between the number of characters in a word and the complexity of the word. It makes sense because lots of simple words are written in katakana and therefore have more character than their kanji counterparts.

The third screenshot shows the fourth query :

```
select radical_basis, radical_meaning, count(*) as kanji_count from kanji
group by radical_basis, radical_meaning
order by kanji_count desc
limit 10;
```

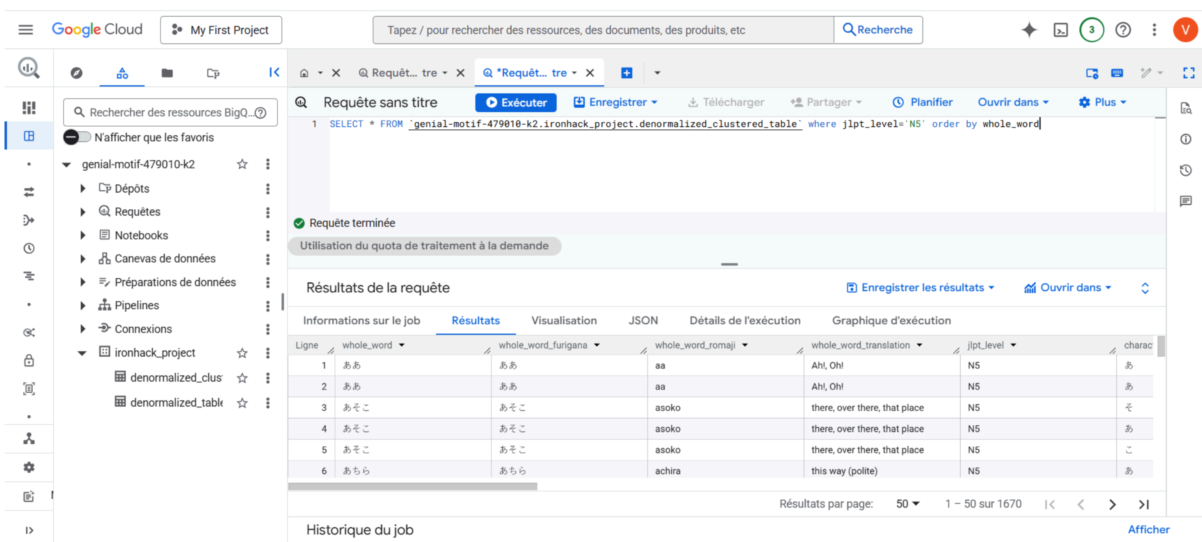
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

radical_basis	radical_meaning	kanji_count
水	water	114
人	man	91
手	hand	87
木	tree	80
口	mouth	75
糸	silk	65
心	heart	63
言	speech	60
辵	walk	49
土	earth	41

This result is another lead of what kanji to learn first because they are the 10 most common radicals to other kanji. If you're used to seeing, writing and understanding these kanji, it will help you learn new ones that share the same radical because of their resemblance.

BigQuery

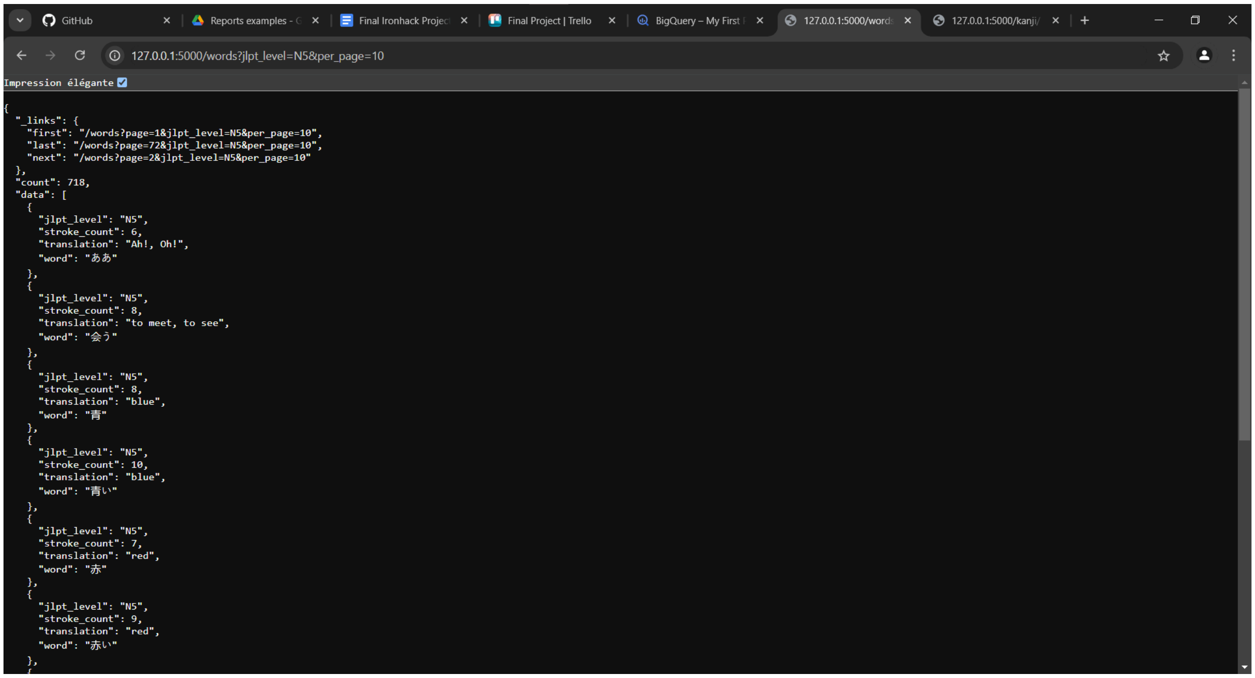
I did not really use BigQuery apart from importing my denormalized dataset to it but I thought it would be a great idea for ML Applications.



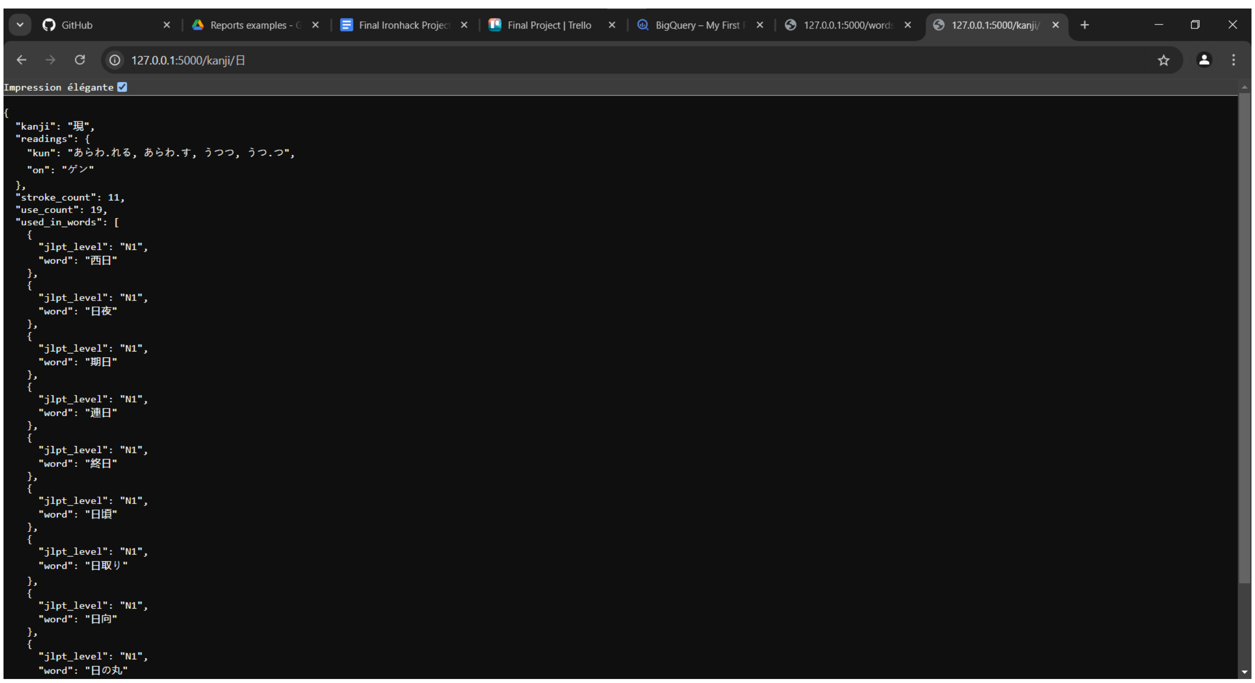
API

Resource	Endpoint	Method	Query Parameters	Description
Words	/words	GET	page, per_page, jlpt_level	Returns a paginated list of words, filterable by JLPT level.
Words	/words/<word>	GET	None	Returns a single word's details.
Kanji	/kanji	GET	page, per_page, min_strokes	Returns a paginated list of Kanji, filterable by minimum stroke count.
Kanji	/kanji/<char>	GET	None	Returns a single Kanji's details, nested with a list of words it appears in.

The screenshot below shows the result of the GET Method used with a filter on jlpt_level (N5) and words per page (10):



The screenshot below shows the result of the GET Method used for a specific kanji



GDPR

This project did not trigger any significant GDPR (General Data Protection Regulation) concerns because it exclusively involved Japanese vocabulary and characters. The GDPR is specifically designed to protect the personal data of individuals within the European Union (EU) and European Economic Area (EEA). Data such as vocabulary lists and character sets, even if language-specific, are considered non-personal data as they do not relate to an identified or identifiable natural person. Since the project contained no names, addresses, identification numbers, location data, or any other information that could be used to directly or indirectly identify an EU resident, it fell outside the scope and regulatory requirements of the GDPR.

Machine Learning Applications

My denormalized BigQuery dataset is well structured for ML because it acts as a ready-made feature store. By denormalizing, I did some feature engineering, aggregating complex, low-level data (like individual Kanji stroke counts or use frequencies) into powerful, quantitative features at the word level. This wide, clean table can be directly used to train models. For instance, you could use these features to build a classification model that predicts the precise JLPT Level of a new, unseen Japanese word, or a regression model to predict the relative learning difficulty score for vocabulary. The table's design eliminates the need for complex joins or pre-processing during model training, streamlining the entire MLOps pipeline.

Conclusion

This project was a great first step to creating a Japanese language learning app. I was able to gather, clean and enrich data from different sources and learned how to expose my data with an API thanks to Flask. This gave me a few leads on which words and kanji I should focus on first to have some solid basics when I go back to Japan. This could also be used for an AI/ML Project later on.