Wikidata Tables

Dataset and Files for CEA Task

The dataset contains:

- The validation set (DataSets/Valid/gt/cea_gt.csv, DataSets/Valid/tables).

cea_gt CSV file

The cea_gt.csv file stores essential information regarding cell values, their corresponding positions, and associated annotations. The dataset comprises four columns, namely: table name, row index, column index, and annotation. The row index column indicates the position of the cell value in terms of rows, while the column index represents its position in terms of columns. The annotation column provides the wikidata URL for the entity associated with the cell value at the specific table for the position (i, j) where i denotes the row index column value and j denotes the value presented in the column index.

Below is a preview of the first five rows of the cea_gt CSV file. As a note, the row index begins from 1, while the column index starts from 0:

IUPOCN5C	1	0	http://www.wikidata.org/entity/Q6386554
IUPOCN5C	2	0	http://www.wikidata.org/entity/Q6500028
BQC7DZZR	1	0	http://www.wikidata.org/entity/Q4975221
BQC7DZZR	2	0	http://www.wikidata.org/entity/Q7996163
BQC7DZZR	3	0	http://www.wikidata.org/entity/Q6858435

Additionally, the dataset analysis reveals the following findings:

- The total number of annotated values within the cea_gt dataset is: `4247`.
- Among these annotated values, there are a total of `3601` unique annotations.

Tables

The dataset consists of tables containing entities sourced from Wikidata. Each table is stored in a separate CSV file. The total number of tables is `500.` The Table below describes an example from the provided tables.

col0	col1	col2
Bryngwyn	250.5	95.9
Castell Dinas Bran	321.4	96.2

Mynydd Mawr	160.0	96.0
Skid Hill	186.0	98.0
Sokol	668.0	100.0

Functional Data Analysis

Columns Count

To gain a deeper understanding of the content within the tables, we initiated our analysis by examining the number of columns in each table. This was achieved by utilizing a count plot, which allows us to visualize the distribution of tables based on the number of columns they contain. Figure 1 provides an illustration of the findings, highlighting that a significant proportion of the tables (about 64%) consist of 2 columns.

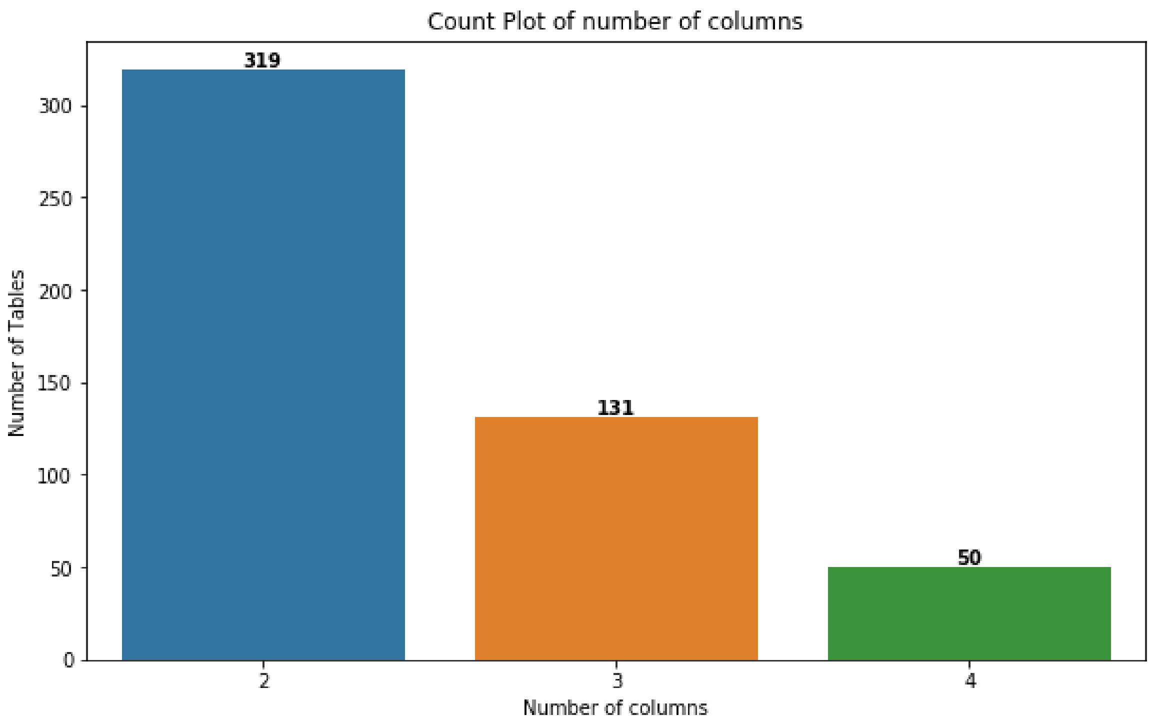


Figure 1: Count Plot of number of columns

Types of Columns

Understanding the types of columns allows for proper data validation and verification. It helps ensure that the data injected into the Wikidata API requests are consistent with the expected data types. Figure 2 shows the total number of columns with each specific type. According to the plot, there are only 3 types which are object, float, and integer with a majority (about 64%) of columns having the string values.

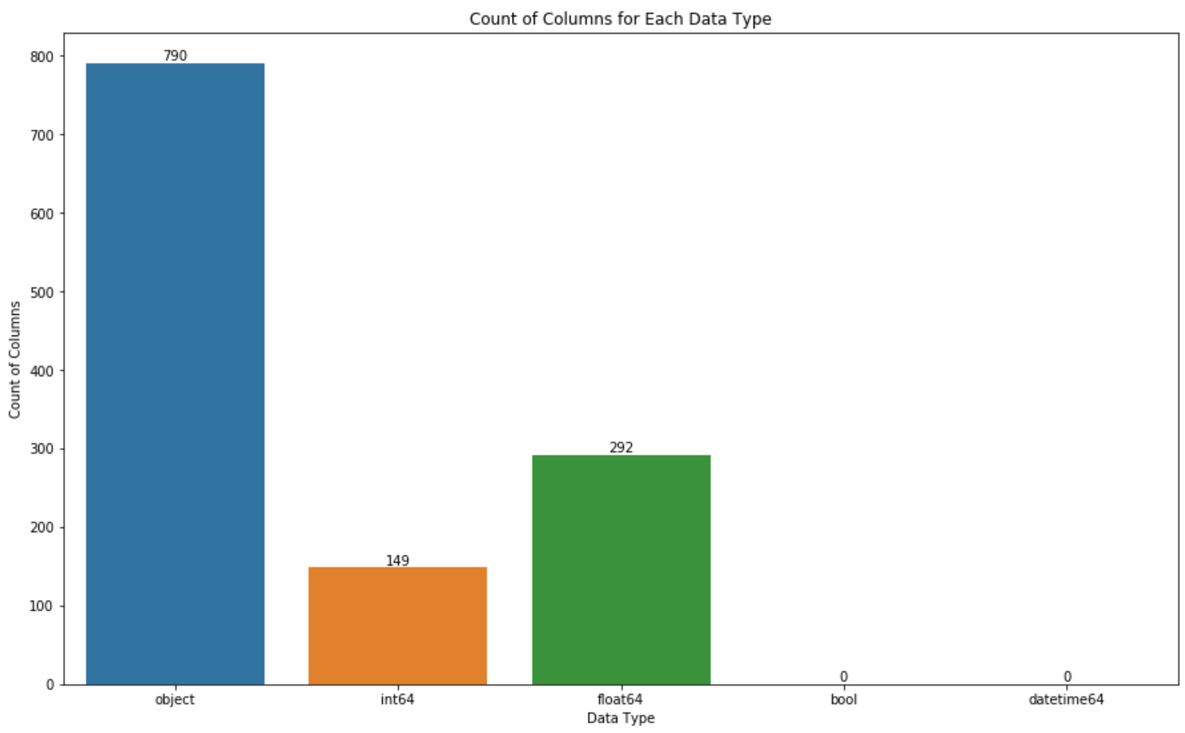


Figure 2: Count of columns for each data type

Missing Values

Detecting missing values is crucial in data analysis and decision-making processes. The total number of missing values in the Wikidata tables is equal to 751. Figure 3 shows that there about 43.6% of total tables have at least one missing value. With a maximum of missing values equal to 10 for 4 tables.

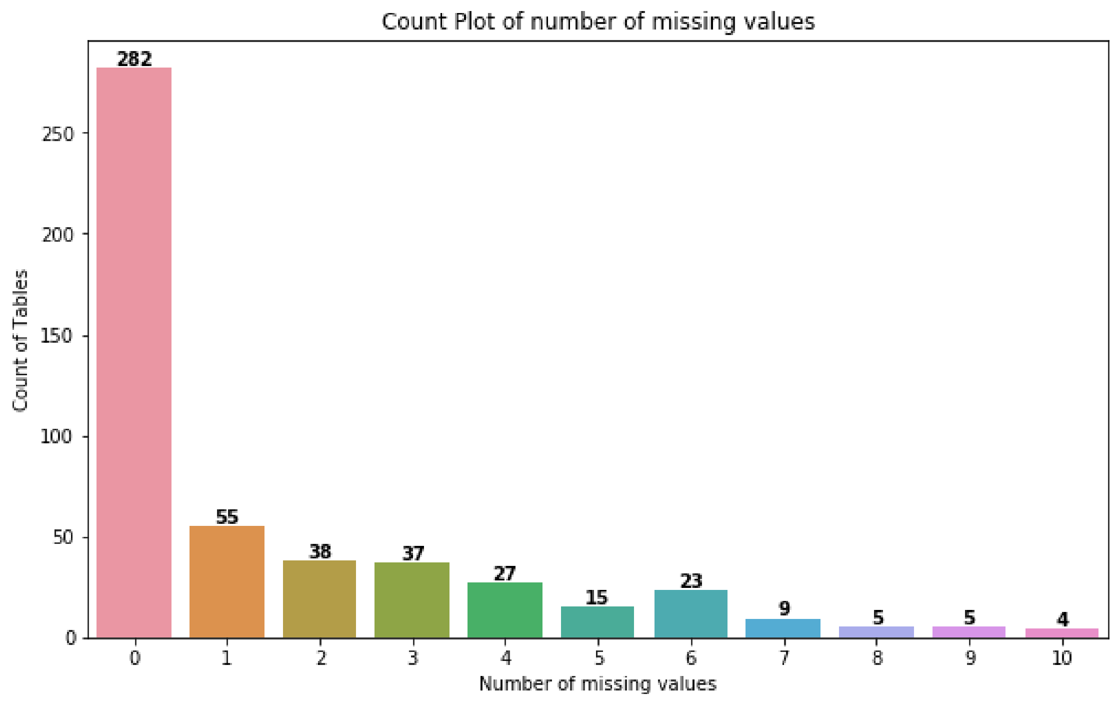


Figure 3: Count of tables for every number of missing values

In order to ensure accurate data analysis, it is essential to identify missing values within the dataset and determine appropriate handling strategies such as imputation or exclusion. In this context, our focus is to investigate whether any annotations are provided for NaN (Not a Number) values, which can help determine if imputation is required. To do this, we randomly select a table named `0WPFA2SX` with some NaN values and are represented with the following (row index, column index) pairs:

- (3,1)
- (4,1)
- (6,1)
- (7,1)
- (2,3)
- (5,3)

col0	col1	col2	col3
Design for an altar crown	paper	King Baudouin Foundation	Heritage Fund
Design for a Virgin Mary	pencil	Cuypershuis	NaN
Design for an altar	NaN	King Baudouin Foundation	Heritage Fund
Design for an altar	NaN	Collection Van Herck	Alfons Van

			Herck
Design for a Female Saint	pencil	Cuypershuis	NaN
Design for a door ornament with putti	NaN	Collection Van Herck	Alfons Van Herck
Design for a door ornament with putti	NaN	Collection Van Herck	Alfons Van Herck

We then checked the cea_targets CSV file if it contains any annotation for one of those values. The rows, having a table name equal to `0WPFA2SX`, are presented below.

0WPFA2SX	1	0	http://www.wikidata.org/entity/Q59097177
0WPFA2SX	1	1	http://www.wikidata.org/entity/Q11472
0WPFA2SX	1	2	http://www.wikidata.org/entity/Q2780100
0WPFA2SX	1	3	http://www.wikidata.org/entity/Q47458848
0WPFA2SX	2	0	http://www.wikidata.org/entity/Q55693869
0WPFA2SX	2	1	http://www.wikidata.org/entity/Q14674
0WPFA2SX	2	2	http://www.wikidata.org/entity/Q15874141
0WPFA2SX	3	0	http://www.wikidata.org/entity/Q59098216
0WPFA2SX	3	2	http://www.wikidata.org/entity/Q2780100
0WPFA2SX	3	3	http://www.wikidata.org/entity/Q47458848
0WPFA2SX	4	0	http://www.wikidata.org/entity/Q59097490
0WPFA2SX	4	2	http://www.wikidata.org/entity/Q58072385
0WPFA2SX	4	3	http://www.wikidata.org/entity/Q58083967
0WPFA2SX	5	0	http://www.wikidata.org/entity/Q55928528
0WPFA2SX	5	1	http://www.wikidata.org/entity/Q14674
0WPFA2SX	5	2	http://www.wikidata.org/entity/Q15874141
0WPFA2SX	6	0	http://www.wikidata.org/entity/Q59097539
0WPFA2SX	6	2	http://www.wikidata.org/entity/Q58072385
0WPFA2SX	6	3	http://www.wikidata.org/entity/Q58083967
0WPFA2SX	7	0	http://www.wikidata.org/entity/Q59097540
0WPFA2SX	7	2	http://www.wikidata.org/entity/Q58072385
0WPFA2SX	7	3	http://www.wikidata.org/entity/Q58083967

Based on the analysis conducted, it is observed that there are no annotations provided for the NaN values within the selected table, "0WPFA2SX". Consequently, it is determined that no imputation or further actions need to be taken to address these missing values.

Languages Used in Cells

As part of the FDA (Functional Data Analysis) process, querying the Wikidata API for specific cell values requires specifying the language of those values. Therefore, it is essential to examine the languages used in different cells. To accomplish this, we first utilized the “langid” package. It is a Python library that offers language identification functionality, enabling us to determine the language of a given text or document. Figure 4 shows the distribution of languages beyond the cells to be annotated using CEA task.

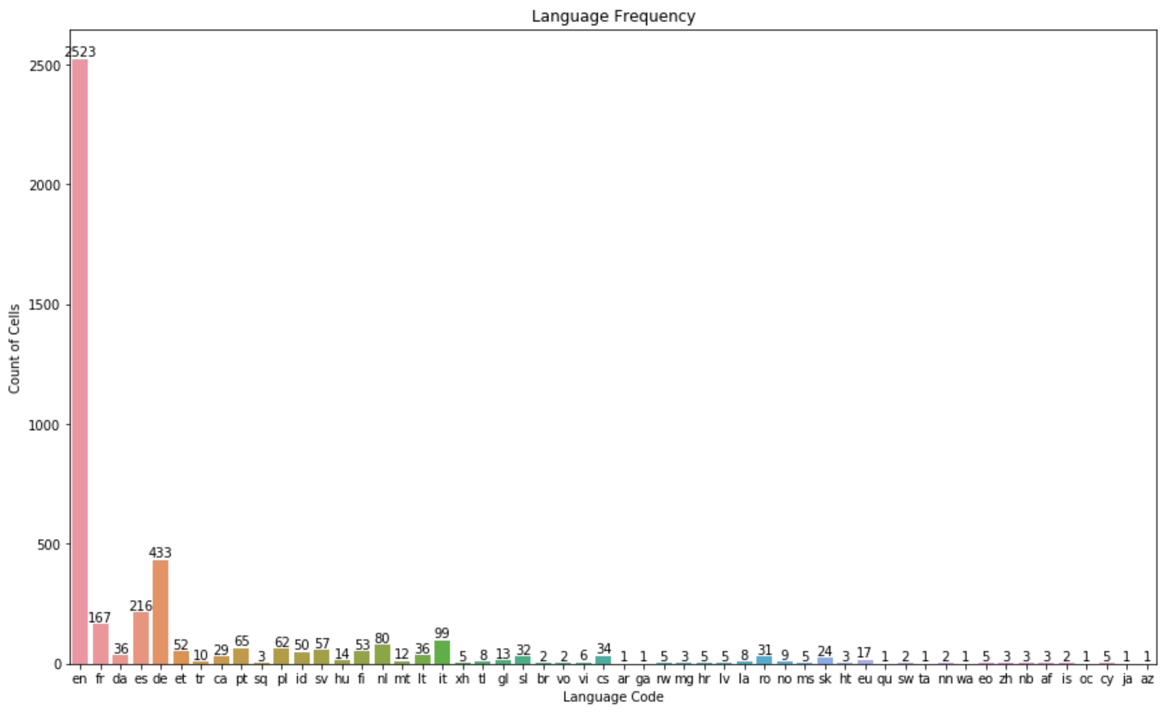
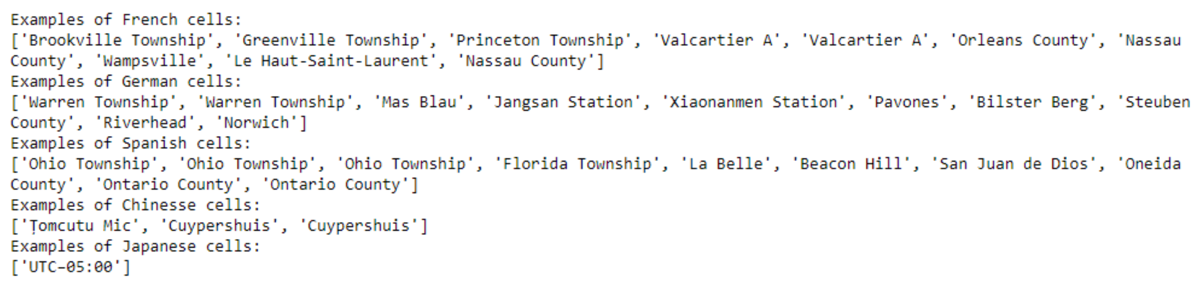


Figure 4: Language Frequency for cells to annotate using CEA task (Langid python package)

The total number of cells with non-english content is 1724 corresponding to 40.6% of total cells. Figure 5 provides a descriptive overview by presenting examples of different languages detected within the dataset. Unfortunately, the detection seems to be inaccurate for some languages. The inaccuracies observed in language detection can arise due to the complex nature of the data, especially when dealing with property names, people names, or location-specific terms.



```
Examples of French cells:
['Brookville Township', 'Greenville Township', 'Princeton Township', 'Valcartier A', 'Valcartier A', 'Orleans County', 'Nassau
County', 'Wampsville', 'Le Haut-Saint-Laurent', 'Nassau County']
Examples of German cells:
['Warren Township', 'Warren Township', 'Mas Blau', 'Jangsan Station', 'Xiaonanmen Station', 'Pavones', 'Bilster Berg', 'Steuben
County', 'Riverhead', 'Norwich']
Examples of Spanish cells:
['Ohio Township', 'Ohio Township', 'Florida Township', 'La Belle', 'Beacon Hill', 'San Juan de Dios', 'Oneida
County', 'Ontario County', 'Ontario County']
Examples of Chinesse cells:
['Țomcutu Mic', 'Cuypershuis', 'Cuypershuis']
Examples of Japanese cells:
['UTC-05:00']
```

Figure 5: Examples of values in detected languages by Langid package

In order to improve the accuracy of language detection, further exploration of alternative packages and solutions was conducted. This search led to the discovery of the Detect Language API (**https://detectlanguage.com/**), a sophisticated and reliable language identification API. The API offers fast and accurate detection for 164 languages, supporting both short and long texts, as well as batch requests. It is important to note that the free plan of the API allows for a maximum of 1000 requests per day and a maximum usage limit of 1MB per day.

To leverage the capabilities of the Detect Language API effectively, the total number of cells with was split into five batches, with each batch containing up to 950 cells. The API calls were applied to each batch daily, with the results stored in a JSON file to avoid redundant detections and for later use. By utilizing this approach, a total of 1371 cells were identified as having non-English content, accounting for approximately 32.28% of the total cell values that were provided with annotations.

The distribution of language frequency is shown in Figure 6.

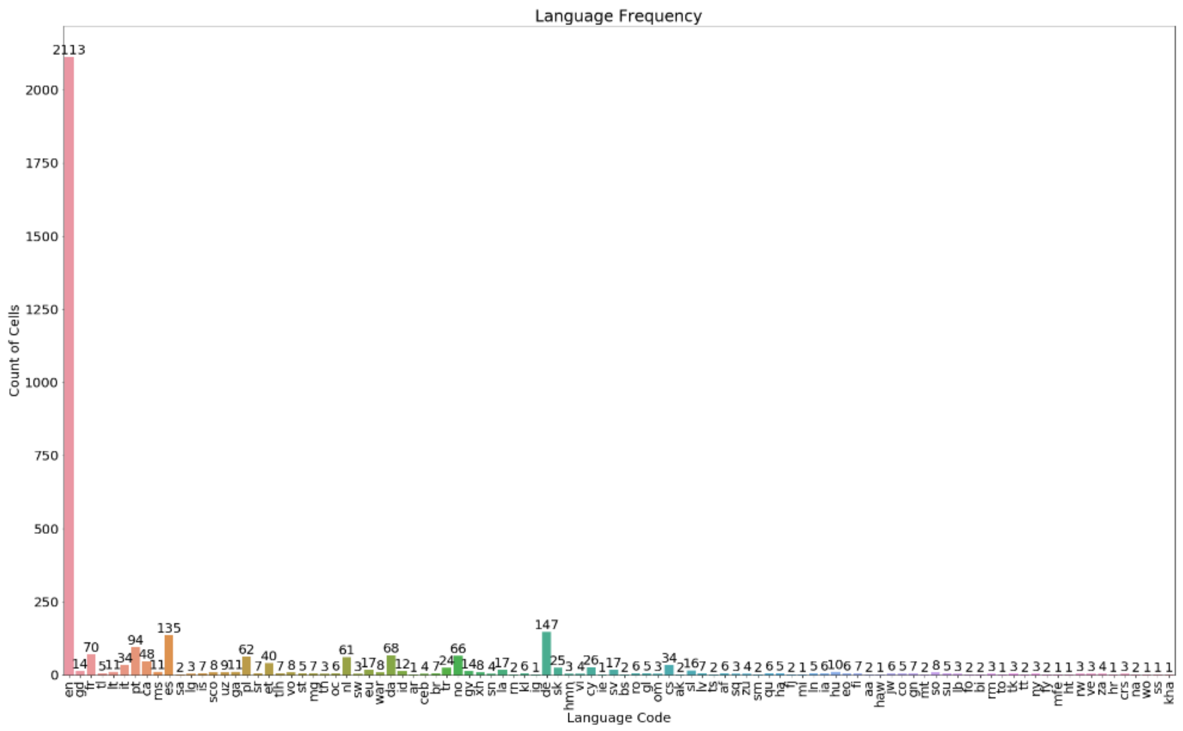
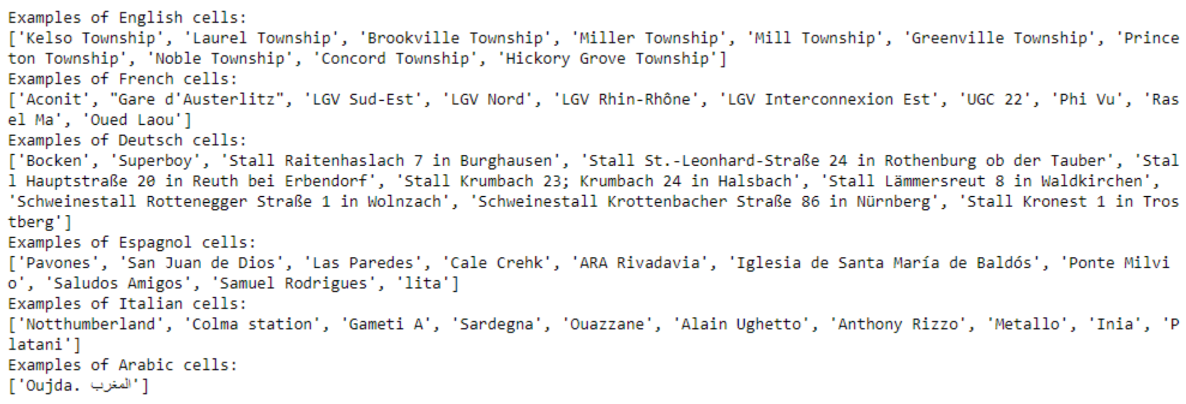


Figure 6: Language Frequency for cells to annotate using CEA task (Detect Language API)

To validate the accuracy of the language detection results, a closer examination was conducted for certain languages, including English, French, German, Spanish, Italian, and Arabic. The analysis revealed that the API demonstrated highly accurate predictions, as depicted in Figure 7.



```
Examples of English cells:
['Kelso Township', 'Laurel Township', 'Brookville Township', 'Miller Township', 'Mill Township', 'Greenville Township', 'Prince
ton Township', 'Noble Township', 'Concord Township', 'Hickory Grove Township']
Examples of French cells:
['Aconit', "Gare d'Austerlitz", 'LGV Sud-Est', 'LGV Nord', 'LGV Rhin-Rhône', 'LGV Interconnexion Est', 'UGC 22', 'Phi Vu', 'Ras
el Ma', 'Oued Laou']
Examples of Deutsch cells:
['Bocken', 'Superboy', 'Stall Raitenhaslach 7 in Burghausen', 'Stall St.-Leonhard-Straße 24 in Rothenburg ob der Tauber', 'Stal
l Hauptstraße 20 in Reuth bei Erbendorf', 'Stall Krumbach 23; Krumbach 24 in Halsbach', 'Stall Lämmersreut 8 in Waldkirchen',
'Schweinestall Rottenegger Straße 1 in Wolnzach', 'Schweinestall Krottenbacher Straße 86 in Nürnberg', 'Stall Kronest 1 in Tros
tberg']
Examples of Espagnol cells:
['Pavones', 'San Juan de Dios', 'Las Paredes', 'Cale Crehk', 'ARA Rivadavia', 'Iglesia de Santa María de Baldós', 'Ponte Milvi
o', 'Saludos Amigos', 'Samuel Rodrigues', 'lita']
Examples of Italian cells:
['Notthumberland', 'Colma station', 'Gameti A', 'Sardegna', 'Ouazzane', 'Alain Ughetto', 'Anthony Rizzo', 'Metallo', 'Inia', 'P
latani']
Examples of Arabic cells:
['Oujda. المغرب']
```

Figure 7: Examples of values in detected languages by Detect Language API

Based on these findings, the utilization of the Detect Language API proves to be a superior choice compared to the Langid Python package. The API's robust capabilities and reliable performance enhance the accuracy and reliability of language detection, providing a more effective solution for detecting non-English content within the dataset.

Misspelling Errors

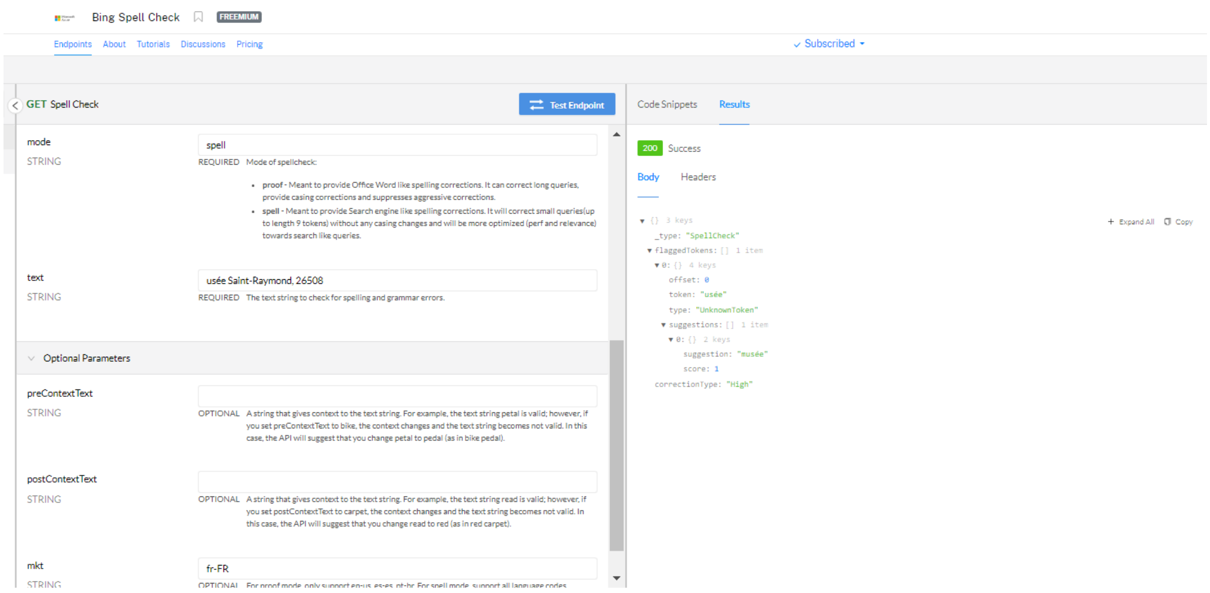
certain strings within the dataset contain misspellings, leading to empty results when querying the Wikidata API. To retrieve the correct entity using the API, it is crucial to correct these misspellings beforehand. Here are some examples of misspelled strings identified within the dataset:

- `City of Porsmouth`: should be "City of Portsmouth".
- `Rutnand`: should be "Rutland"
- `Nort Linconshire`: should "North Lincolnshire"

To solve the misspelling issue, we can use Bing Spell Check API. It is a powerful tool that offers advanced spelling correction capabilities, particularly when it comes to resolving errors related to names. It provides a spell property specifically designed to deliver search engine-like spelling corrections. This feature is particularly adept at correcting small queries, limited to a maximum length of 9 tokens, without altering the casing of the words.

One of the key strengths of the Bing Spell Check API is its ability to handle errors commonly encountered with names. Names can be notoriously challenging when it comes to spelling, with numerous variations and unique spellings. The API's spell property excels at providing accurate corrections for these types of errors, enabling developers to enhance the overall user experience by reducing the frustration caused by misspelled names.

However, this API offers only 1000 requests per month for the Free package. Thus, we need to use it effectively with strings that have 100% issues with misspellings and need to be corrected. The Figure 8 shows an example of usage for the API.



mode
STRING
spell
REQUIRED Mode of spellcheck:

- **proof** - Meant to provide Office Word like spelling corrections. It can correct long queries, provide casing corrections and suppresses aggressive corrections.
- **spell** - Meant to provide Search engine like spelling corrections. It will correct small queries(up to length 9 tokens) without any casing changes and will be more optimized (perf and relevance) towards search like queries.

text
STRING
usée Saint-Raymond, 26508
REQUIRED The text string to check for spelling and grammar errors.

preContextText
STRING
OPTIONAL A string that gives context to the text string. For example, the text string petal is valid; however, if you set preContextText to bike, the context changes and the text string becomes not valid. In this case, the API will suggest that you change petal to pedal (as in bike pedal).

postContextText
STRING
OPTIONAL A string that gives context to the text string. For example, the text string read is valid; however, if you set postContextText to carpet, the context changes and the text string becomes not valid. In this case, the API will suggest that you change read to red (as in red carpet).

mkt
STRING
fr-FR
OPTIONAL For proof mode, only support en-us, es-es, pt-br. For spell mode, support all language codes

200 Success

```
▼ {} 3 keys
  _type: "SpellCheck"
 ▼ flaggedTokens: [] 1 item
  ▼ 0: {} 4 keys
     offset: 0
     token: "usée"
     type: "UnknownToken"
   ▼ suggestions: [] 1 item
    ▼ 0: {} 2 keys
       suggestion: "musée"
       score: 1
  correctionType: "High"
```

Figure 8: Example of usage for the API

Special Characters

Some cells within the dataset contain special characters that can potentially deceive the Wikidata API, making it incapable of retrieving accurate annotations. Several examples of such special characters are presented in the table below:

Special Character	Examples of Strings	Count	Position
.	[awhaka Creek., Caigieburn., Guardia., L. S. Kee, ,…]	55	[[end], [end], [1, 4], …]
{	[Frank{]	1	[[end]]
-	[Saint-Raymoqd, 26399, Musée Saint-Raymoqd, …]	11	[[11], [11], …]
}	[Bron}e pendant in the form#of a paired couple]	1	[[26], [15]]
#	[Bron}e pendant in the form#of a paired couple, Planta da Villa#Pirajussara Bairro de Pinheiros]	2	[[7]]
:	[Pendant: cescent]	1	[[7]]
,	['Lima, Ohio, metrppolitan area', …]	15	[[24, 52], [24, 52], [13], [19], [19], [18], …]
!	[Musée Saint-Raymond,!26399, …]	4	[[20], [29], [end],

			[6]]
(	[Terracotta onochoe (jug), …]	3	[[19], [13], [20]]
)	[Terracotta onochoe (jug), …]	3	[[end], [end], [end]]
—	[Gulfport–Biloxi metropolitan"area, …]	2	[[19], [8]]
“	[Gulfport–Biloxi metropolitan"area, …]	2	[[28], [10]]

After analyzing the various strings with special characters and their impact on retrieving entities from the Wikidata API, it is evident that a pre-processing step is necessary to address this issue. In this situation, we identified that:

- A full stop ‘.’ in the end of the string should be deleted.
- We apply the correction first. If no possible entity is retrieved then we proceed with handling the special character. For !, #, ", we replace them with space since they usually seperate two different words in a string. For “}”, “{”: replace with empty char “”. Other special characters are kept since they are usually a crucial part of the string.
- Delete additional spaces in start and end of a string.