CEA

The CEA (Cell Entity Annotation) task focuses on annotating each target cell with a corresponding entity from Wikidata Knowledge Graph. To accomplish this, the MediaWiki API can be utilized, as it provides a robust and powerful tool for interacting with various functionalities, including the retrieval of information about entities and their properties from the Wikidata knowledge graph.

This API serves as a valuable resource for accessing and retrieving information, ultimately enhancing the quality and depth of annotations within the CEA task.

Querying the Wikidata API

To retrieve the entity associated with a specific value using the MediaWiki API, the following steps are followed:

1. Construct the API request URL, specifying the action as "wbgetentities" (which stands for "Wikibase get entities").
2. Include the value of the cell as a parameter in the URL, typically using the "search" parameter. For example, if we want to obtain the entity for the value "Kelso Township", we would include "title='Kelso Township'" in the URL.
3. Send a GET request to the constructed URL using the Python requests package.
4. The API will respond with a JSON object containing the information about various potential possible entities associated with the provided value.
5. Parse the JSON response, select an entity, and extract the desired information, such as the entity ID, the full url in Wikidata, descriptions, or other properties.

By following these steps, the MediaWiki API facilitates the retrieval of the entity corresponding to a specific value.

Evaluation

$$Precision = \frac{correct\_annotations\#}{submitted\_annotations\#}$$

$$Recall = \frac{correct\_annotations\#}{ground\_truth\_annotations\#}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

First Experiment:

In the initial experiment, we injected all the cells requiring annotation, specified in the cea_gt CSV file, directly into the Wikidata API. Then, we selected the first entity from the candidates (possible entities) returned by the API. The table below presents the resulting score metrics obtained from this experiment:

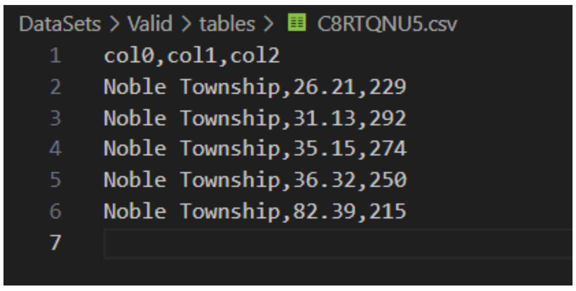
Precision	Recall	F1-score
0.701	0.663	0.681

The exact number of correct, incorrect and empty annotation are described in the table below.

Correction Annotations	Incorrect Annotations	Empty Annotations
2815	1200	232

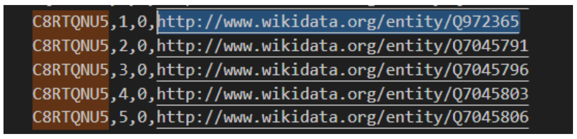
Second Experiment

To improve the results, we employed a strategy that leveraged the values from the same row to aid in identifying the most appropriate entity from the potential entities returned by the API. These values were utilized as values for properties associated with the entity. For instance, consider the string "Noble Township" from the first row in the "C8RTQNU5" table. The target entity for this string is Q972365, and it is accompanied by two different row values: 26.21 for the second column and 229 for the third column, as shown in Figure 1. Upon exploring the corresponding Wikidata page for this entity, it was discovered that these row values serve as values for two properties, namely "area" and "elevation above sea level" respectively as shown in Figure 2.



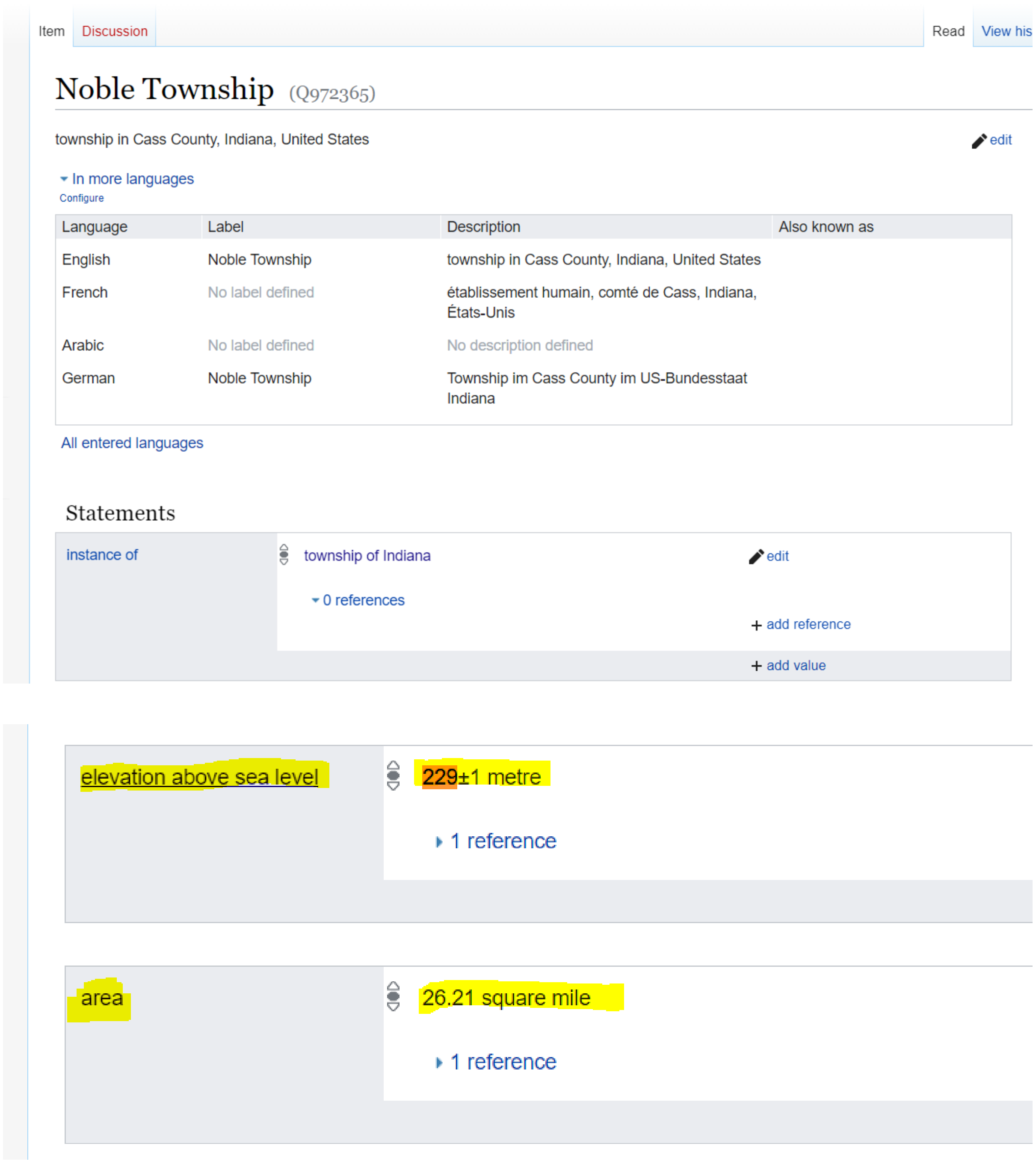
DataSets > Valid > tables > C8RTQNU5.csv

```
col0,col1,col2
Noble Township,26.21,229
Noble Township,31.13,292
Noble Township,35.15,274
Noble Township,36.32,250
Noble Township,82.39,215
```



```
C8RTQNU5,1,0,http://www.wikidata.org/entity/Q972365
C8RTQNU5,2,0,http://www.wikidata.org/entity/Q7045791
C8RTQNU5,3,0,http://www.wikidata.org/entity/Q7045796
C8RTQNU5,4,0,http://www.wikidata.org/entity/Q7045803
C8RTQNU5,5,0,http://www.wikidata.org/entity/Q7045806
```

Figure 1: Samples in C8RTQNU5.csv file and their corresponding CEA targets.



Noble Township (Q972365)

township in Cass County, Indiana, United States

Language	Label	Description	Also known as
English	Noble Township	township in Cass County, Indiana, United States	
French	No label defined	établissement humain, comté de Cass, Indiana, États-Unis	
Arabic	No label defined	No description defined	
German	Noble Township	Township im Cass County im US-Bundesstaat Indiana	

Statements

instance of	township of Indiana
elevation above sea level	229±1 metre
area	26.21 square mile

Figure 2: Wikidata page for the entity Q972365

In this experiment, we adopted an iterative approach to handle each potential entity obtained from the API. For each possible entity, we retrieved all associated properties along with their corresponding values. A crucial step in this process involved checking if the cell values from the same row were included in the retrieved properties values.

If, during the iteration process, any of the possible entities were found to have the cell values from the same row included in their properties, that entity was selected as the appropriate annotation. In case of none of the possible entities has the values from the same row in their proerties, we proceed with selecting the first possible entity.

To retrieve the properties along with their values, we used sparql. SPARQL (SPARQL Protocol and RDF Query Language) is a query language specifically designed for querying RDF (Resource Description Framework) data. It provides a standardized way to retrieve and manipulate data stored in RDF format, including the vast knowledge graph maintained by Wikidata.

Skipping existing NaNs value in the same row.

When we don't skip the NaN value in the process of verifying if the value exists in the properties/values json retrieved by SPARQL, we can get some wrong annotation. In fact, the condition returns always "False" and therefore the possible entity is skipped and assumed as wrong annotation. Therefore, we need to add a condition that captures and skip NaN values.

In addition, it was observed that when querying the Wikidata API without specifying the "limit" parameter, a maximum of seven possible entities were returned. This limitation posed a challenge when certain cell values had more than seven potential entities associated with them. As a result, there were instances where the target entity was not present within the first seven retrieved entities.

We chose to proceed the second experiment with a limit equal to 50 (maximum).

The table below presents the resulting score metrics obtained from the second experiment without skipping NaNs:

Precision	Recall	F1-score
0.843	0.797	0.819

The exact number of correct, incorrect and empty annotation are described in the table below.

Correction Annotations	Incorrect Annotations	Empty Annotations
3384	631	232

The table below presents the resulting score metrics obtained from the second experiment without skipping NaNs:

Precision	Recall	F1-score
0.866	0.819	0.842

The exact number of correct, incorrect and empty annotation are described in the table below.

Correction Annotations	Incorrect Annotations	Empty Annotations
3478	537	232

Third Experiment

In the third experiment, we took care of the misspelling errors and the special characters which are presented in 232 cell value (returned NaN during the last experiment). The total number of strings without any special character and with misspelling errors is 161. We tried to apply the correction first and then handling the special character if the API still cannot return any possible entity for the input string. We excluded the strings without special character and we applied the correction to the rest. The results are described in the table below.

Total Number of misspelled strings without special characters	Successful having the correct entity in the possible entities returned by the API
161	82
	50.93%

For the strings with special characters, we started with handling the full stops "." presented in the end of the string. We applied the correction at first, then proceeded with the deletion for the strings that still has empty returned result from the API. The results are presentend in table below.

Total Number of strings with full stop in the end	Successful having the correct entity in the possible entities returned by the API after correction	Getting the correct entity in the possible entities returned by the API after correction + full stop deletion
37	1	15
	2.7%	40.54%

For the strings with a full stop “.” in the middle, we only apply correction + deletion of the full stop in the end if there is. The table below presents the results:

Total Number of strings with full stop in the middle and if there is a full stop in the end	Successful having the correct entity in the possible entities returned by the API after correction + deletion of the full stop in the end if there is
18	5
	27.78%

We handle other strings with other special characters as specified in the FDA and Processing part. The table presents the results for them.

Total Number of strings with a special character different than a full stop “.”.	Successful having the correct entity in the possible entities returned by the API after correction.	Successful having the correct entity in the possible entities returned by the API after correction + special character handling
28	12	17
	40.625%	56.25%

The table below describes the

The table below presents the resulting score metrics obtained from the third experiment:

Precision	Recall	F1-score
0.853	0.837	0.845

The exact number of correct, incorrect, and empty annotations are described in the table below.

Correction Annotations	Incorrect Annotations	Empty Annotations
3556	612	79