Zipcode Plugin Documentation

Overview

The Zipcode Plugin helps preprocess and analyze a dataset column to calculate a confidence score for how likely each value in the column is a valid US zip code. The plugin has three main parameters for calculating the score: the column name, the value format, and an external API checker.

Function Definitions

get_confidence_score()

The get_confidence_score() function takes in the column name as a string and a sample of the dataset column as a list of strings.

The function first checks if the column name contains the substring "zip" or "postal". A boolean variable “colname_check” acts as a checker that is set to True if the column name matches the condition.

The next step is to preprocess the data: the list of values is sent to functions that remove all the null values and leading/trailing spaces from the data.

The function then initializes a list of scores to keep track of all the confidence score values for the column. Each value is passed through the format and API checker(conditional) function and the scores list is appended accordingly. Finally, the function returns the average of all the scores, which is the final confidence score of the entire column.

remove_null()

This function removes all the null and missing values from the data. Values like NA, null, and Nan are removed to avoid skewness in the results.

remove_spaces()

This function removes all leading and trailing spaces (if any) from the values. This ensures that the data is properly formatted and consistent, making it easier to work with and reducing the chances of errors in the code.

get_format()

This function checks the format of the value passed to it. Valid US zip codes are five-digit (55555) or 5+4 digit codes (55555-5555). This function acts as a boolean checker that sets the value of format_check to True if the condition is satisfied.

get_api_value()

This function checks the validity of a value passed to it through an external API. Opencage is a geolocating service that provides geocoding of addresses. It uses a range of data sources to

provide precise results. This API has an attribute “_type” that stores the type of the value being passed. For our function, we check if the result of “_type” is a postcode or not. A boolean api_checker is set to true if the final result of the API is true.

Computational Reasoning:

The column name has the most priority, followed by the format of the value. Unless there is any ambiguity in the results of column name and format, the value is not checked through an API. Based on the final values of the three checkers: colname_check, format_check and api_check. The get_sconfidence_score() assigns the following confidence scores:

Condition	Confidence Score
Column Name - True Format- True	100

If the column name matches and the format is correct, it is certain that the value is a zipcode, hence it is given a 100% confidence score

Column Name - True Format - False	80

If the column name matches but the format is incorrect, the score is set to 80. This is a rare scenario and could be possible in case of some error in data or missing values not detected by the remove_null() function.

Column Name- False Format - True	Checks through API - 100 if True - 0 if False
Column Name - False Format - False	Checks through API - 100 if True - 0 if False

If the column name does not match there is a case of ambiguity regardless of the result of the format. In this case, the values are passed through the API to find precise results. If the results of the API match, we know for certain that the value is a valid zip code and would return a 100% confidence score.

For version 1 of the plugin, only the API was used to calculate the final confidence score. Although the results were precise, two other parameters were added due to the following reasons:

- Performing tests by systematically trying every possible value through the API can significantly impact the efficiency in terms of time. This approach leads to longer

processing times and increased computational demands which can reduce the overall performance. Therefore, alternative testing strategies were considered to help optimize the API's efficiency while ensuring comprehensive test coverage.

- Implementing alternative strategies also helped decrease the number of requests made to the API and reduced API usage costs.

This approach resulted in improved testing efficiency and cost saving for the organization.
In the event of higher demand, a paid plan can be considered to meet increased user requirements.
OpenCahe Pricing details: https://opencagedata.com/pricing