OVERVIEW

The Generic Number Plugin (GNP) takes in a column name and dataset column values that are assumed to be converted to a list of strings. The GNP first removes any leading and trailing spaces in the strings. The GNP then removes any null (None) values or any string values denoting a null value ('NA', etc.) found in the list. Then, the GNP returns a score for each value based on how well the GNP is confident that that value is a generic number. After scores have been assigned, any outlier scores (anything not within 2 standard deviations from the mean) are removed before taking and returning the final average of the scores (the final confidence score for the entire column).
Note: The Generic plugins are used for the Fallback column in the Catalog.

FUNCTION DEFINITIONS

DEPENDENCIES: re

get_confidence_score()

The *get_confidence_score()* function takes in a *string* called *col_name* and a list of *strings* called *col_vals*. *col_vals* is presumed to be a dataset column that has been converted to a list of *strings*. *col_name* is presumed to be the dataset column name. The function then creates an empty *double* list called *scores* to contain the confidence score for each value in *col_vals*. The function then modifies *col_vals* by calling on the *remove_lead_trail_space()* then the *remove_null()* function. After, the function iterates through each value in *col_vals* and calls on the *get_elem_score()* function to append each score returned from that function to the *scores* list. Afterwards, the *remove_outliers()* function is called on the *scores* list. Finally, the average (which is *double* type) of the *scores* list is returned (this is the final confidence score for the entire column) or 0.0 is returned if *scores* is empty.

Note: col_name is never used as the column name will not have an effect whether the column denotes a generic type or not. It is still passed in as the Framework is expected to pass both the column names and the column values into the plugins.

get_elem_score()

The *get_elem_score()* function takes in a *string* called *elem*. *elem* is presumed to be a value from a list of *strings*. The function then uses regular expressions in an if-statement to determine what confidence score (which is *double* type) to return. *See the NUMBER CONVERSIONS section for more information on the confidence score.*

remove_null()

The remove_null() function takes in a list of strings called *col_vals*. *col_vals* is presumed to be a dataset column that has been converted to a list of strings. The function uses list comprehension to return a list where any None values have been removed and where any strings denoting null values have been removed. The list of strings that denote values are noted as: ['NA', 'N/A', 'na', 'n/a', 'Na', 'N/a*', '']*.

remove_lead_trail_space()

The remove_lead_trail_space() function takes in a list of strings called *col_vals*. *col_vals* is presumed to be a dataset column that has been converted to a list of strings. The function uses list comprehension to return a list where all the string elements have leading and trailing space removed.

remove_outliers()

The *remove_outliers()* function takes in a list of *doubles* called *scores*. *scores* is presumed to be the list of confidence scores of each value in a dataset column. *scores* is immediately returned if the *scores* list is empty. The function first creates an empty *set* to contain outliers found. The function then calculates the average and the standard deviation of *scores*. Then the function

iterates through each score in *scores* and determines whether the score is within 2 standard deviations from the average. If the score is not found within that range, that score is added to the outlier list. Because 95% of values in any distribution (*Statistic Found Here*) fall within that 2 standard deviation range, generally only extreme outliers will be removed. Finally, list comprehension is used to remove any found outliers from the *scores* list then that modified list is returned.

NUMBER CONVERSIONS

Conversion Per Number Format

Number Format	Regex Form	Confidence Score
12345	(\d+(\.\d+)?)	100.00
45678.56789000	(\d+(\.\d+)?)	100.00
.456789	(\.\d+)	100.00
67,564	(\d{1,3},\d{3}(,\d{3})*(\.\d+)?)	100.00
89,890,890.89089898	(\d{1,3},\d{3}(,\d{3})*(\.\d+)?)	100.00
98%	(\d+(\.\d+)?%)	90.00
98.56%	(\d+(\.\d+)?%)	90.00
.53%	(\.\d+%)	90.00
34/343	\d+/\d+	40.00

Conversions Per Confidence Score

Note: These regex forms are used in the if-else statements.

Regex Form	Confidence Score
(\d+(\.\d+)?)|(\.\d+)|(\d{1,3},\d{3}(,\d{3})*(\.\d+)?)	100.0
(\d+(\.\d+)?%)|(\.\d+%)	90.0
\d+/\d+	40.0

Generally, there is one regex expression per confidence score category.
If none of the above formats are matched, a confidence score of 0.0 is returned.

The confidence scores for each format was chosen due to the specificity of the format. For example, the whole numbers both in standard and comma format *(123456 and 123,456)* and decimal format *(123.456)* generally do not denote anything except for generic numbers *(hence the 100.0 confidence scores)*. Percentages are used for more special cases and less commonly seen as ‘generic’ hence the 80.0 confidence score. Fractions have a 40.0 score because they could also denote a date *(12/2 can also be interpreted as December 2nd)*. *These confidence score pairings are subject to change either to team majority vote or mentor discretion.*

re.fullmatch() is used for these regular expressions so any string containing any sort of letter or other special character not in the group, *[./%]*, are immediately nixed and return a 0.0 score.

IMPLICATIONS FOR FUTURE

In the future, we would like to handle scientific notations for numbers (8.67×10^3). We could do additional regular expressions, however the challenge would be handling superscripts and subscripts.

In the future, we would like to create a separate Generic plugin for ID. For time purposes, the Generic ID plugin is absorbed into the Generic Number Plugin (since whole numbers can denote IDs too).

Since this plugin uses the same format as other plugins that use regular expression, we would like to have one .txt file containing all data types' regular expressions and one Python file that handles all regular expression cases instead of having separate Python files for each data type.