Interactive Data Wrangling for Northstar

6.830 Final Project Paper
Gina Yuan, Joanna Sands, Sophie Mori
{gyuan, joannas, ksmori}@mit.edu

Abstract

We propose adding common data wrangling capabilities to Northstar, an interactive data science tool. This streamlines the data analysis process for all users, especially users who may not be familiar with more complex existing data cleaning tools. We analyzed which cleaning capabilities were worth adding and then implemented those features in Northstar. As an extension, we implemented automatic rule detection to further help analysts clean their data.

Introduction

Northstar is an interactive data science tool that saves data analysts time with data configuration, leaving more time to analyze the data. Northstar combines many data sources to unveil relations among the sources. The system works well for data that is already in a usable form, but there currently is not a way to remove unusable entries and ensure that incoming data is typed accurately. Data analysts instead need to preprocess the data, either manually or with existing tools for data cleaning and transformation like Wrangler.

An integration of existing data wrangling practices into Northstar would improve the ease of use for Northstar users. This is especially important considering Northstar's goal is to streamline the data science process and have good general usability. Taking inspiration from the Wrangler paper, we implemented four basic wrangling features: (1) delete null rows, (2) delete given values for row (3) fill null values, and (4) extract regex. We also implemented automatic rule suggestions for these wrangling features. Both contributions improve the usability of Northstar by integrating wrangling and analysis capabilities into a single system.

Design

The Northstar codebase is composed of two repositories, the frontend and the backend, that interact over HTTP endpoints. When the backend starts up, it loads specified datasets from a configuration file into memory. As the user interacts with the frontend, the frontend sends requests to the backend. One example of such a request is making a histogram for a certain

attribute. The backend then performs the computation and sends the result back to the frontend. Occasionally, the dataset may be too large to fit in memory, but we do not consider that case for simplicity.

We designed the new wrangling operations to fit into the frontend-backend model by creating a new **/wrangler** POST endpoint (Figure 1). The endpoint takes a different set of parameters for each supported wrangler operation. Based on the type of the parameters, the backend mutates the data and sends the results of the operation back to the principal who made the request. We also created a **/wranglersuggestion** POST endpoint that does a pre-analysis of the dataset and returns a list of suggested wrangler operations based on automatic rule detection.

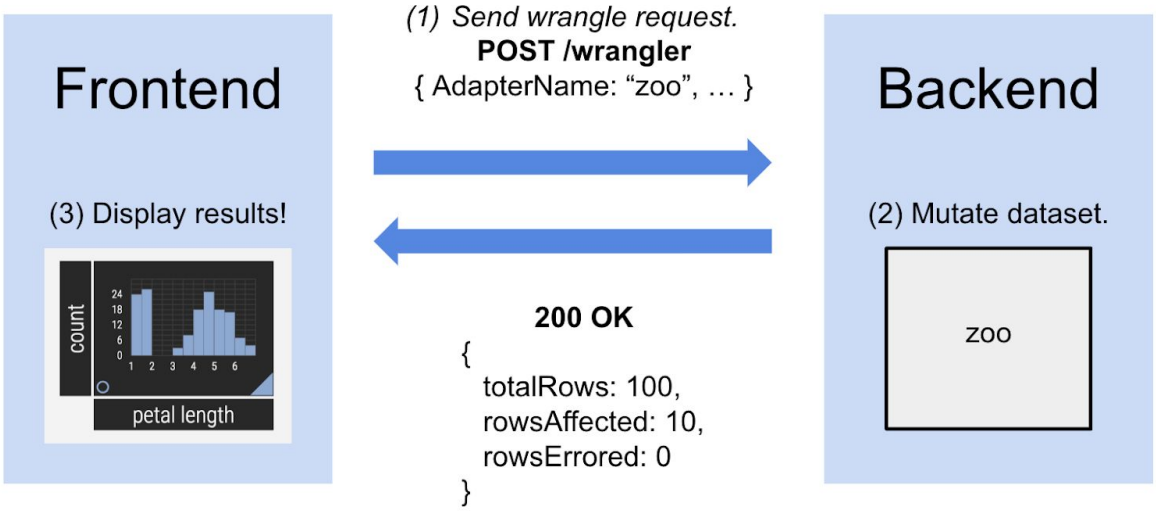

Figure 1: *The frontend interacts with the backend by making POST requests to the /wrangler endpoint with a set of parameters for each operation. The backend mutates the dataset in memory and returns a result that is displayed in the frontend.*

We built an interactive form in the frontend to streamline the process of submitting requests to the **/wrangler** endpoint (Figure 2). Before the user submits anything, the frontend automatically makes a POST request to **/wranglersuggestion** and visualizes suggested operations in the form. The user can choose to submit a custom wrangling request or one based on these suggestions. The form automatically formats the parameters and sends the POST request to the **/wrangler** endpoint. In the same window, the frontend will visualize the returned results and show how many rows were affected out of a total number of rows.

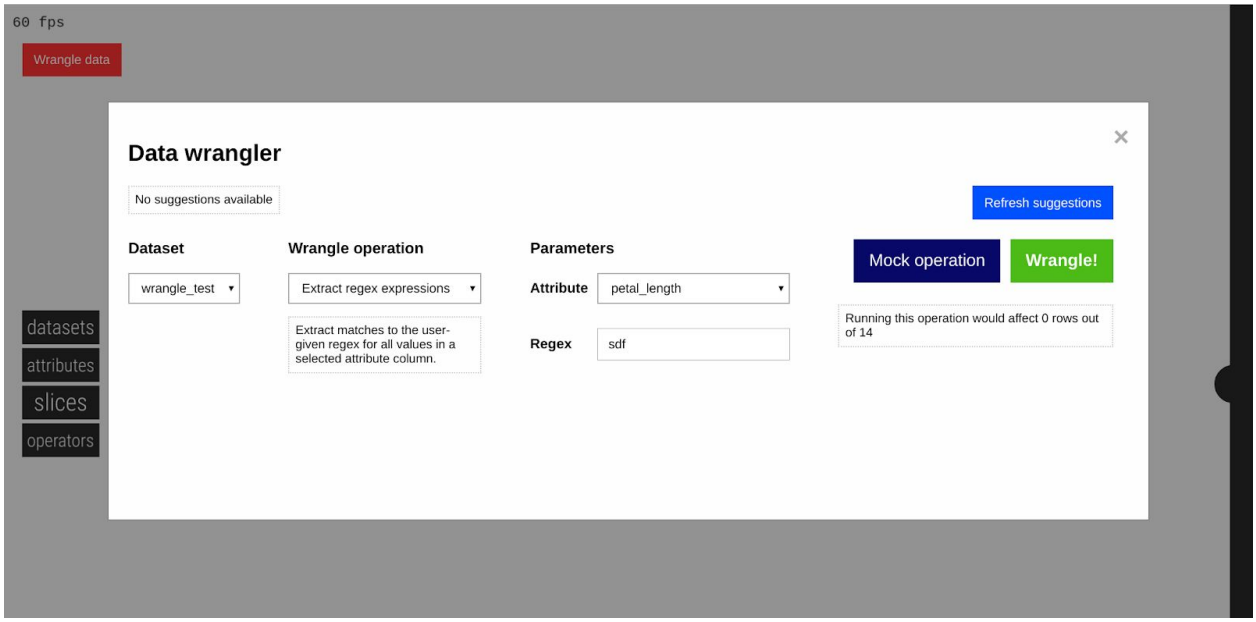

Figure 2: *The UX flow for completing a wrangler operation. From left to right, the user selects a dataset, a wrangle operation, and the parameters for that operation. After the request successfully completes, the number of affected rows is displayed.*

DeleteNullRows
{ AdapterName: str,
AttributeNames: str[] }

FillNullValues
{ AdapterName: str,
AttributeName: str,
AttributeValue: *obj }

ExtractRegex
{ AdapterName: str,
AttributeName: str,
Regex: str }

DeleteGivenValue
{ AdapterName: str,
AttributeName: str,
AttributeValue: obj }

Figure 3: *Summary of wrangling operations and the parameters they take. * indicates optional.*

Features

Wrangling Operations

(1) Delete null rows
This operation deletes any rows with null values in the given attributes.

Null values may appear in rows to show that the corresponding attribute does not apply to that particular row, or that no data was collected for the attribute. However, the presence of null values in Northstar can cause anomalies in the histogram generation because the binning functions are unable to determine the correct bin for a null value (Figure 4b). After deleting rows with null values from the dataset (Figure 4a), the histogram is able to correctly render bins (Figure 4c).

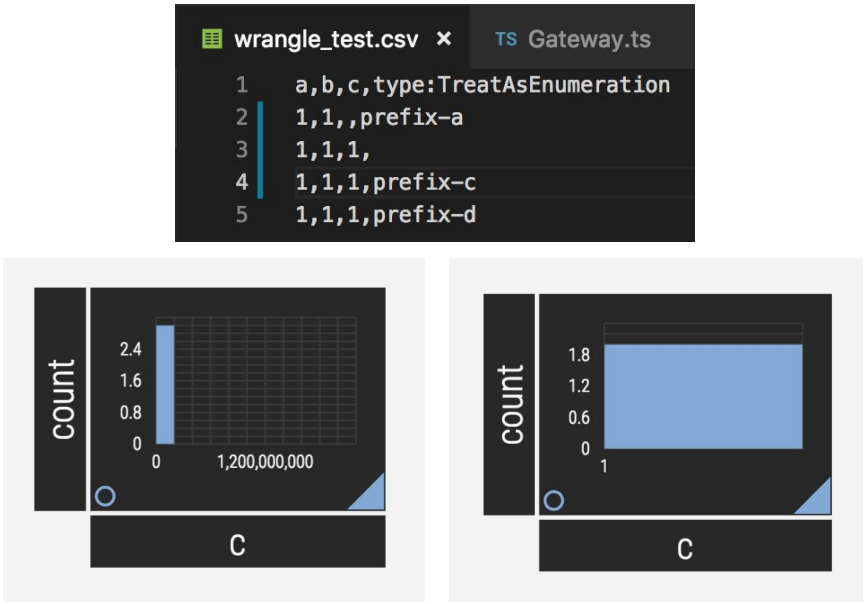

Figure 4: *(4a, top) A minimal test dataset with null values in the first two rows. (4b, bottom left) Pre-wrangle histogram that fails to handle null values, generating anomalous bins from 0 to 1.2 billion. (4c, bottom right) Post-wrangle histogram that correctly renders a single bin for value 1.*

Additionally, another reason to delete rows with null values is in order to analyze on complete data. For example, if we wish to analyze a certain statistic for young children, it would not be helpful to include and analyze the rows where the age was not recorded.

Our system allows users to specify exactly which attributes they wish to check when deleting on null values, and filters the data only on the specified attributes.

(2) Fill null values

This operation fills in null values in that attribute either with the given value, or if no value is given, the last non-null value seen in that attribute.

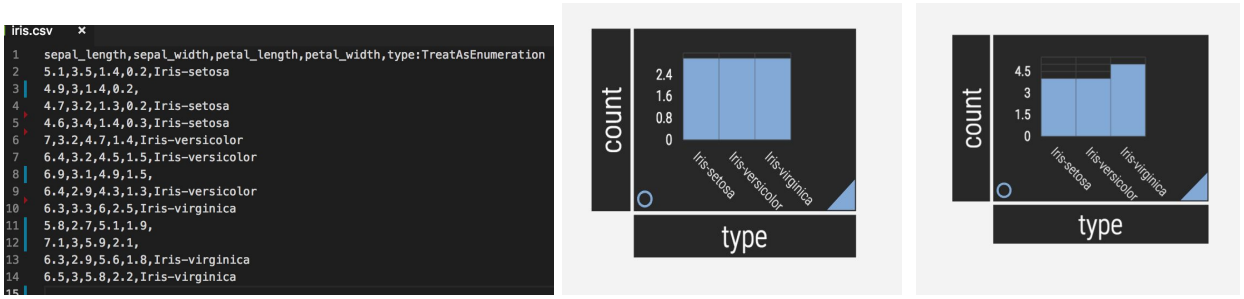

Figure 5. *Null values for attribute "type" were filled based on the most recent valid value. This results in the count of each bar increasing.*

This operation most immediately applies to values where a default value should have been applied, but the null value was set instead. For example, a data enterer may forget to include the country when inserting the address of a new student to a dataset because in most cases, the country is the United States. The students who are not from the United States will have their countries entered correctly because their case is the non-default case. In this case, the operation should be used to fill in null values with the United States.

In some datasets, there may be enough information in the dataset to automatically fill in the null values. In the Wrangler paper, there is an example of a csv composed of what appears like multiple smaller datasets (Figure 6). In this case, the header of one of the smaller datasets can actually be used as the value for that attribute for each of the null subsequent rows. This is what we mean by filling in null values with the last non-null value.

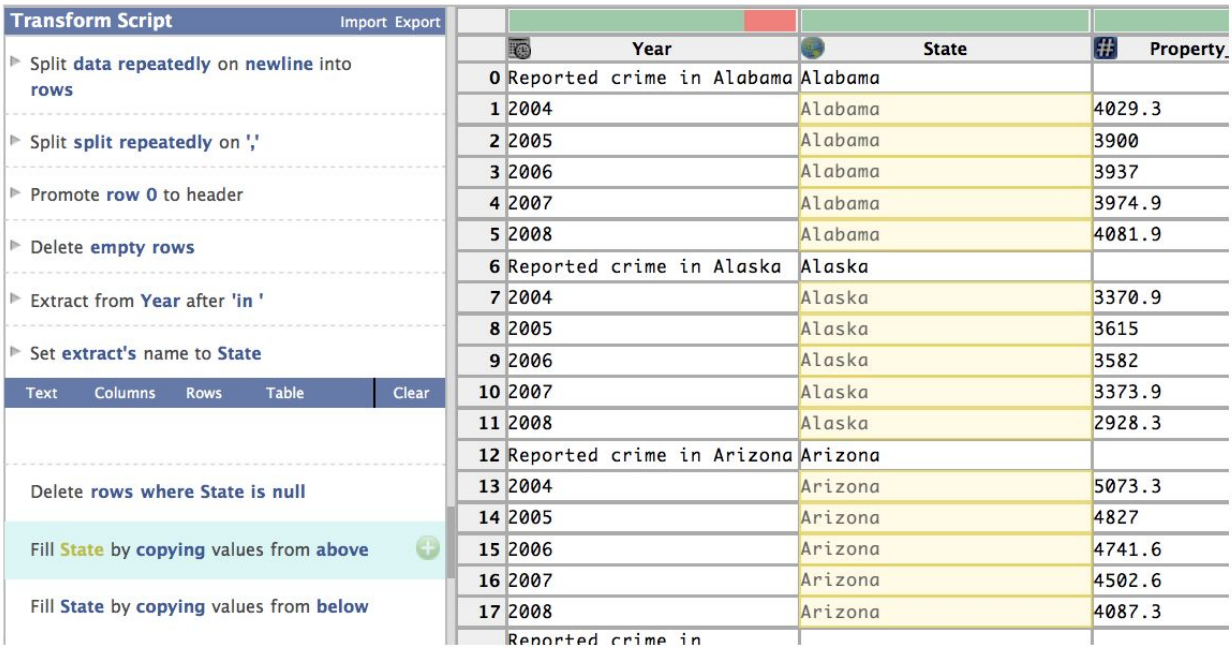

Figure 6: *Example from Wrangler of the motivation behind automatically filling in null values.*

(3) Extract regex

This operation replaces a string in an attribute with the string extracted by the given regex.

Text extraction is useful for when long strings are stored in a value, and the analyst would like to extract a smaller but more significantly meaningful substring for analysis. Again referring to the example in the Wrangler paper (Figure 6), the subheaders “Reported crime in Arizona” is much more meaningfully analyzed as the single substring of the state “Arizona”. This simple feature gives users the capability to discover powerful insights from uncleaned strings.

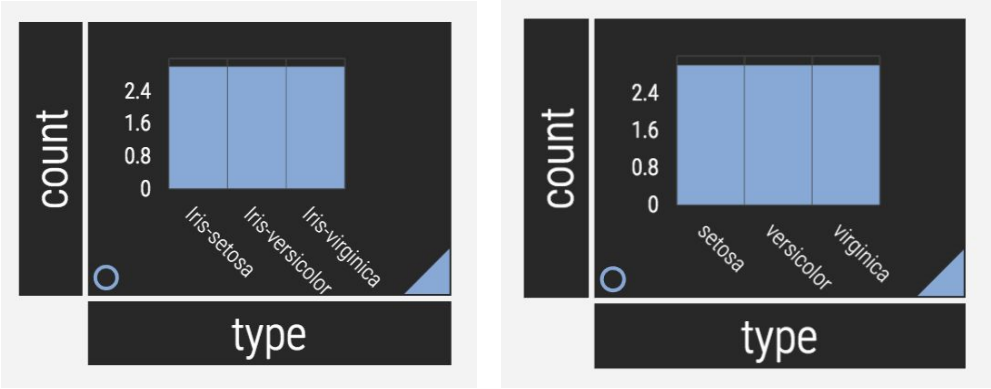

Figure 7: *Result of the operation on the iris dataset in Figure 5. The operation extracted the iris species from the type attribute by matching with regex "iris\-(.*)", removing the redundancies across the values.*

Mocking Operations

For any operation, we give the user to "mock" the operation before running it on the dataset. The mock gives the user the same information as executing the operation would do (how many rows would be affected), but it does not change the dataset. This allows users to perform a sanity check before any operation.

Another effect is that after a user executes a certain operation, if they then mock the identical operation, they should see that 0 rows would be affected now and know that their executed operation had been successful.

Automatic Detection

The goal in implementing automatic rule detection was to move the burden of deciding exactly how to clean a dataset from the human analyst to a machine. Our system currently analyzes the data in sample batches upon page load and collects metadata on the values for each attribute. From there, it applies a heuristic to this data in order to decide whether or not a specific wrangling operation may be helpful in cleaning this data. The system then sends back a list of attributes for which each wrangling operation would be useful.

One example of this process is calculating how many null values exist for a given attribute. For simple detection, we created a heuristic which says that within a column which has a small proportion of null values, those null values are probably the result of errors and should be removed from the data set. If there are a high proportion of null values, we suggest that the user fill in those values instead. We set thresholds for deciding whether the proportion is high or low.

For each attribute, the automatic detection code finds how many null values there are, checks that against the thresholds, and adds the attribute to a delete null suggestion list or a fill null suggestion list accordingly. We return all these suggestions to the user.

```
deletionSuggestions: sepal_width petal_length petal_width type sepal_length
```

Figure 8: *After adding partially empty lines to the dataset in Figure 5, the system used a heuristic that counted null values per attribute to identify columns for potential deletion.*

Future Work

We would like to expand the system to be able to handle datasets that do not fit in memory. Since “mutating the dataset” currently involves mutating an object in memory representing the tuples, we would need to instead write to persistent state while considering performance. This is important because most data analysis these days are performed on datasets on the order of GBs or TBs large.

A natural extension would be to add more wrangling features and more complex wrangling features. We have already implemented the four basic wrangling features described in the Wrangler paper. In the future, it might be interesting to allow operations that, for example, extract text into a new column, detect and delete outliers, or map and filter data based on specific user parameters.

Another area of future work is improving the frontend, especially for a system where usability is a key metric of success. We can improve the system by including the ability for users to see which wrangling operations have already been applied, as well as to revert applied operations.

Finally, we would like to improve the heuristics we use to automatically suggest operations to take. The heuristics we use are primitive, and may become even more complex with the more complex wrangling features described above. In addition, calculating heuristics currently requires analyzing the entire dataset, which is not a scalable process. We might instead want to only sample a fixed number of values. Since rule suggestions are essentially pattern matching, another possibility would be to use machine learning to automatically detect useful operations.

Conclusion

In this paper, we integrated Northstar with common data wrangling features including deleting rows with null or certain values, filling in null values, and extracting text from regex. We found that these wrangling operations not only aligned with common use cases a data analyst might face, but also resolved bugs in Northstar where null values were improperly handled. We also implemented automatic rule suggestion based a set of heuristics from pre-processing the dataset. In the future, we hope that tools like Northstar and Wrangler can be fully integrated to streamline the data analysis process into a single interactive tool.

Acknowledgment

We would like to thank Emanuel Zgraggen for his help in getting us started in the Northstar codebase and for reviewing our plan to help us find the most efficient way to approach our problem. We would also like to thank Professor Tim Kraska and Raul Castro Fernandez for teaching this databases course.

References

Northstar Paper: http://www.vldb.org/pvldb/vol11/p2150-kraska.pdf
Data Wrangler: http://vis.stanford.edu/wrangler/
Stanford's demo app: http://vis.stanford.edu/wrangler/app/
http://vis.stanford.edu/files/2011-Wrangler-CHI.pdf
Vizdom paper:
https://pdfs.semanticscholar.org/4c81/5e21c909211d7c047a2437938b24217e0a22.pdf?_ga=2.204332880.1750662818.1539375394-418437015.1539043586
More Vizdom: http://tupleware.cs.brown.edu/vizdom/
http://www.vizdom.eu/
IBM rule discovery paper:
https://pdfs.semanticscholar.org/a9ac/6a858ad547f1d6452991958fd3d94b769d88.pdf