Functional Dependency-based Null-Value Replacement

Lakehouse: Data Integration Services for Data Lakes

Ragna Solterbeck

Supervisors:

Alexander Vielhauer
Prof. Dr. Thorsten Papenbrock

24.03.2023

Abstract This term paper is about the question if functional dependencies can be used to replace Null values. This is important, because missing values can complicate an analysis of tables. For this purpose a process was developed that uses functional dependencies to search for a suitable value for a Null value within the table. Analyses have shown that this process could replace 31.841 values in the tested tables. With the help of this program values can be replaced, but it would be important to check the quality of these replacements.

1 Motivation

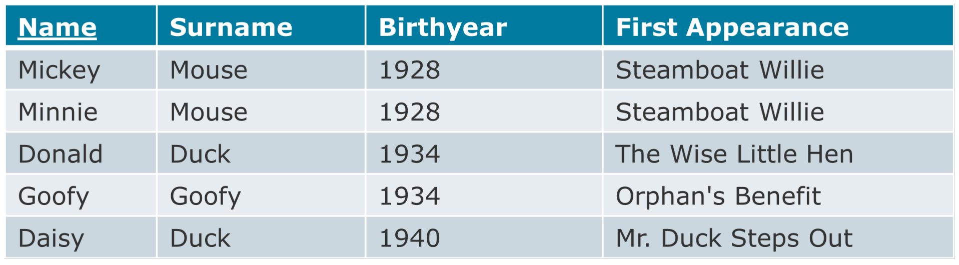
In this term paper we will discuss the question if missing values can be filled using functional dependencies. It is important to acknowledge that Null values within a table can pose significant challenges to data analysis. Incomplete data can lead to inaccurate results or wrong conclusions, therefore making it critical to find a method for handling missing values.

Functional dependencies describe a relationship between attributes in a table, where the values of one attribute determines the values of the other one. Formally, a functional dependency is defined as follows: Let R be a relation schema, and let $X \subseteq R$ and $A \in R$. A is functionally dependent on X ($X \rightarrow A$) if and only if for every pair of tuples t1 and t2 in R that have the same value for X, they must also have the same value for A.

Let $X \rightarrow A$ and $Y \rightarrow A$ be two functional dependencies. If $X \subseteq Y$ holds, then $X \rightarrow A$ is a generalization of $Y \rightarrow A$ and $Y \rightarrow A$ is a specialization of $X \rightarrow A$. With $|X| = 1$, it is called an unary functional dependencies and with $|X| = \text{n} > 1$, it is called n-ary. X is the left-hand-side (lhs) and A is the right-hand-side (rhs) of the Functional Dependency. $X \rightarrow A$ is minimal if there is no generalization $Y \rightarrow A$. If $A \in X$ holds, then $X \rightarrow A$ is trivial.

In the example table in Figure 1, $Name \rightarrow FirstAppearance$ is a minimal functional dependency, because for each value in Name there is exactly one value in First Appearance. $Surname \rightarrow FirstAppearance$, for example is no functional dependency because there are two different values in First Appearance for the value "Duck". $Name, Surname \rightarrow FirstAppearance$ on the other hand is a functional dependency, but it is not minimal because it is a specialization of $Name \rightarrow FirstAppearance$.

Figure 2 shows the search space for the table in the form of a lattice, which is a graph in which each node is connected to its direct subsets and supersets [3]. The search space includes all possible subsets of the attribute set. An edge between two nodes describes which functional dependency is involved. For example, the edge between nodes A and AB is equivalent to $A \rightarrow B$. Each node pair thus describes a possible functional dependency.



Name	Surname	Birthyear	First Appearance
Mickey	Mouse	1928	Steamboat Willie
Minnie	Mouse	1928	Steamboat Willie
Donald	Duck	1934	The Wise Little Hen
Goofy	Goofy	1934	Orphan's Benefit
Daisy	Duck	1940	Mr. Duck Steps Out

Figure 1: Example Table

The lattice shows all minimal functional dependencies (thick edges), non-minimal functional dependencies (dashed edges) and non functional dependencies (grey edges) of the given table from figure 1

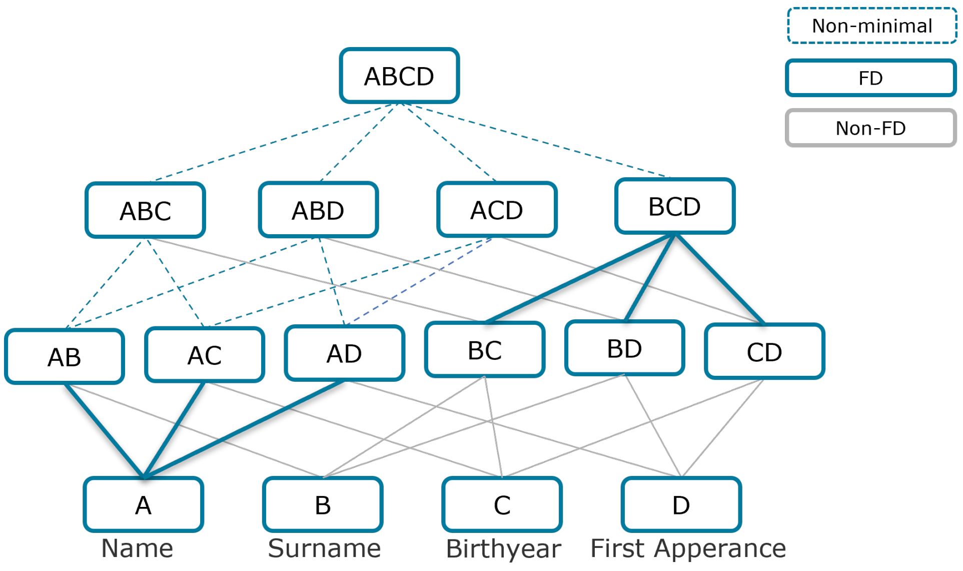


Figure 2: Lattice and functional dependencies (FD) of the table and the different types of lines

2 Related Work

"RENUVER: A Missing Value Imputation Algorithm based on Relaxed Functional Dependencies" by B. Breve, L. Caruccio, V. Deufemia and G. Polese [2]
In this paper the algorithm "RENUVER" is developed. It tries to fill Null values using Relaxed Functional Dependencies. The essential difference between normal and Relaxed Functional Dependencies is that with normal dependencies there is only a dependency if the rhs values match exactly. With Relaxed Functional Dependencies, however, an

approximate match is sufficient. And this is where it differs from this work, in that it only considers normal functional dependencies.

"Conditional Functional Dependencies for Data Cleaning" by P. Bohannon, W. Fan and F. Geerts, X. Jia and A. Kementsietsidis [1]
This paper develops a method that uses conditional functional dependencies for data cleaning. The difference with normal functional dependencies is that conditional dependencies allow constraints to be placed on data based on conditions.

"Enriching Data Imputation under Similarity Rule Constraints" by S. Song , Y. Sun, A. Zhang, L. Chen and J. Wang [6] In this paper, four algorithms are developed to handle missing values using differential dependencies. Similar to relaxed functional dependencies, differential dependencies place more emphasis on approximate matching. Here, similarity rules are defined beforehand, which are then used to determine the dependencies.

The difference between the present work and the three papers mentioned above is that in this case we have concentrated only on the exact correspondence.

"Pandas" and "Pandas User Guide" by pydata.org [5]
The Python library Pandas offers many different functions for working with, analyzing and manipulating table data. The use of Pandas in this work allowed for fast and accurate manipulation of tables with and without Null values.

3 Big Picture

The program consists of a total of three components: "Main", "Null Replacer" and "FD Finder". Figure 3 shows a rough overview of the components, what they exchange with each other, how the dataflow looks like and how the final output is structured.

"Main" calls the "Null Replacer", which in turn calls the "FD Finder". In the "FD Finder" the functional dependencies are determined and verified and returned to the "Null Replacer". Here the Null values are replaced with the help of the found dependencies. These modified tables are passed to "Main" where the final output is generated. This contains an ID of the table, the replacement rate, the initial number of Null values and the number of Null values after execution of the program. The Replacemnt Rate corresponds to the number of replaced Null values in percent. Figure 4 shows what the output of the program looks like in Python. For example, the table with ID 4 had 345 Null values before and 180 after the program was executed, which corresponds to a replacement rate of 47.83%. In the case of table 6, no dependencies could be found, so the text "There are no Functional Dependencies" was generated as output. For tables in which no Null values could be found (e.g. table 0), the text "There are no Null values" was generated.

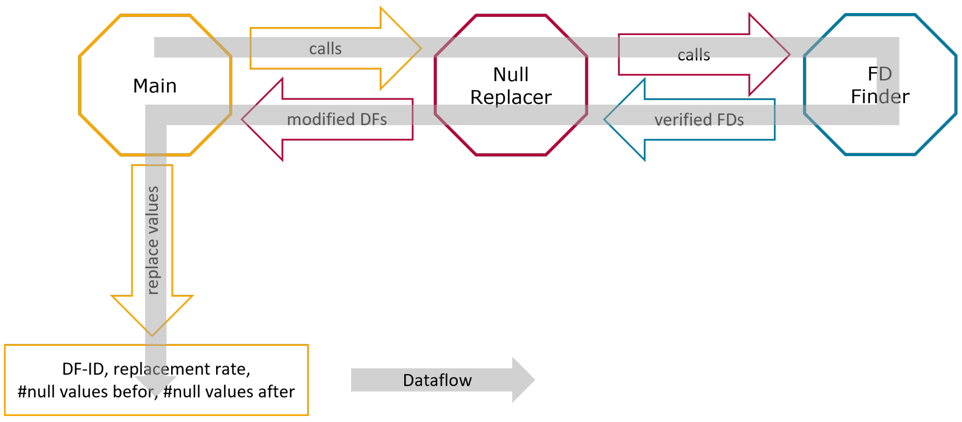
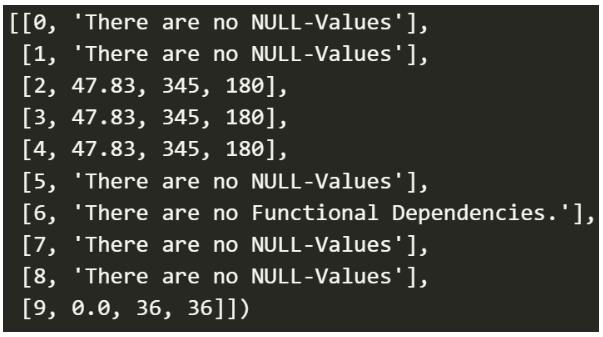


Figure 3: Architecture Overview and the internal dataflow



```
[[0, 'There are no NULL-Values'],
 [1, 'There are no NULL-Values'],
 [2, 47.83, 345, 180],
 [3, 47.83, 345, 180],
 [4, 47.83, 345, 180],
 [5, 'There are no NULL-Values'],
 [6, 'There are no Functional Dependencies.'],
 [7, 'There are no NULL-Values'],
 [8, 'There are no NULL-Values'],
 [9, 0.0, 36, 36]])
```

Figure 4: Output from the "Main" component

3.1 Main

The "Main" component starts the program and provides the final output. It consists of the parts "dataset_inerator" and "main". Figure 5 shows how the two parts interact and how the dataflow in "Main" looks like. The "dataset_iterator" starts the process for a list of tables. To do this, "main" is called for each table in the list. "main" can also be passed a list of different types of Null values. If "main" is now called for a table, the table is first searched and all values from the Null list are replaced by the standard pandas Null value "NaN". Afterwards it is checked whether the table contains Null values. If this is not the case, the program returns "There are no Null values" to the "dataset_iterator". Otherwise, "main" calls the "Replacer" component for the table and returns the result provided by the "Replacer" to the "dataset_iterator". In the "dataset_iterator" the returned results are stored in a list and output after all tables have run through the program.

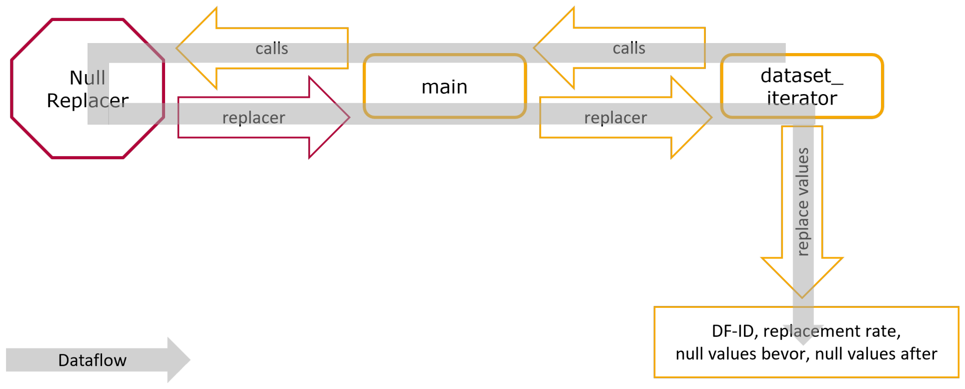


Figure 5: Architecture of the Component "Main"

3.2 Null-Replacer

The "Null Replacer" takes a table and replaces Null values using the found functional dependencies. It also consists of two parts, "replacer" and "replacer_null". Figure 6 shows the structure of the component and the dataflow.

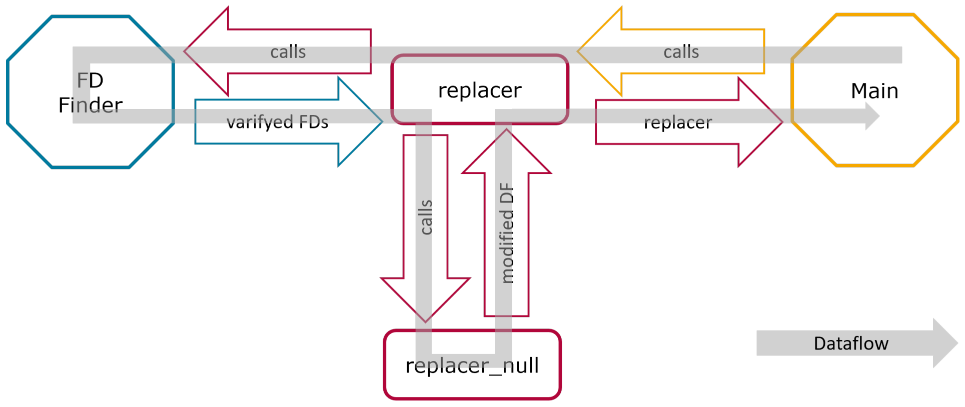


Figure 6: Architecture of the Component "Null-Replacer"

The "replacer" first calls the "Find FD" component and determines the number of Null values in the table. If the list of dependencies passed by the "FD Finder" is empty, the program is not executed further and "There are no Functional Dependencies" is returned to the "Main" component. Otherwise, the table is passed to "replacer_null" together with the verified dependencies from the "FD Finder".

"replacer_null" goes through the process shown in Figure 7 for replacing a Null value. First it looks for a Null value in rhs, then it looks for the corresponding value X in the lhs attribute. The table is grouped by X and all corresponding rhs values in the group are written to a list. The Null value is replaced by a, the first non-Null value in the list. The Null value can simply be replaced by a, since there can only be values in the list that are either Null or a, otherwise it would not be a functional dependency after all.

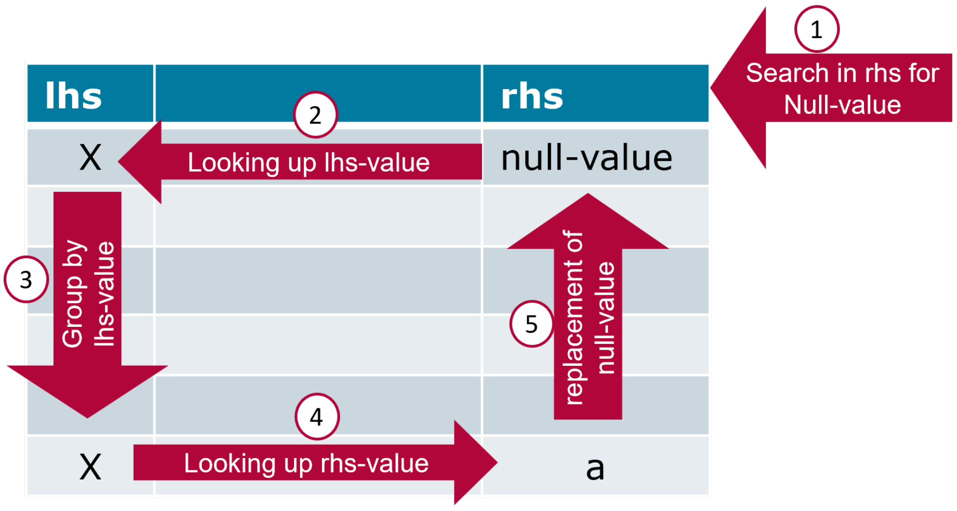


Figure 7: Null-value replacement process

The ”null_replacer” passes the modified table back to the ”replacer”. Here the number of Null values is counted again and the replacement rate is determined. The table is returned to the ”Main” component together with the replacement rate, the number of Null values before and the number after.

3.3 FD Finder

The ”FD-Finder” finds and verifies the functional dependencies. It consists of the three parts ”find_FDs”, ”fastFD” and ”verify_FDs”. The cooperation as well as the data flow within the component is shown in Figure 8.

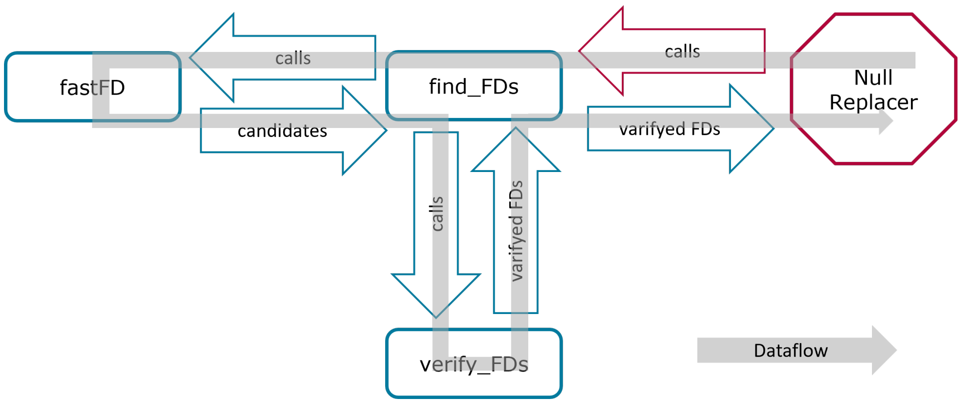


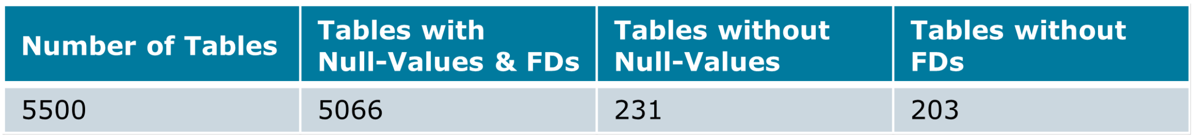
Figure 8: Architecture of the Component ”FD Finder”

In ”find_FDs” first the columns are determined in which Null values are located. Then it calls ”fastFD” and passes the corresponding attributes to it. In ”fastFD” the candidates for the functional dependencies are determined and returned to "find_FDs". The fastFD algorithm was not developed as part of this term paper but is based on a paper by C. Wyss, C. Giannella, and E. Robertson [7]. However, for the purposes of this work it was adapted in such a way that for the right side of the functional dependencies only those attributes are considered which contain Null values.

Afterwards "find_FDs" calls "verified_FDs" and passes the candidates to it. For verification, all rows are ignored in which either the left or right side contain Null values. Then, for each candidate, the table is grouped by the values in the lhs column and checked to see if the values for the rhs column are unique. If so, the candidate is a functional dependency and is stored in a list. Then, for each of the verified dependencies, it is checked if it is a specialization of another verified dependency. If no, it is written to the list of verified minimal Functional Dependencies. This list returns "verified_FDs" to "find_FDs". Which in turn returns it to the Null Replacer component.

4 Evaluation

For the evaluation, 5500 tables were considered and the returned values were analyzed. The tables were taken from the gittables-dataset [4]. Figure 9 shows the distribution of the tables into the three output types tables with Null values and functional dependencies, tables without Null values and tables without functional dependencies. The used Tables can be found in the GitHub repository: FDs and Null-Values



Number of Tables	Tables with Null-Values & FDs	Tables without Null-Values	Tables without FDs
5500	5066	231	203

Figure 9: Distribution of the tested tables in the three output types

Of the 5066 tables with Null values and functional dependencies, Null values could be replaced in 3550 of them. The program managed to replace a total of 34,841 Null values, which corresponds to an overall replacement rate of 3.01% and an average number of replaced values per table of 6.29 Values (see Figure 10).

For the following analyses, only those of the 5066 tables with a replacement rate > 0 were considered. This applies to a total of 3550 of the tested tables.

4.1 Analysis Replacemnt Rate

Figure 11 shows the distribution of replacement rate values. For example, about 3050 tables have a replacement rate between 0 and 10%. Figure 12 shows the same distri-

Null-Values before	Replaced Null-Values	Mean replaced Null-Values	Replacement Rate Overall
1,058,530	31,841	6.29 Values	3.01 %

Figure 10: Evaluation results for replaced Null values

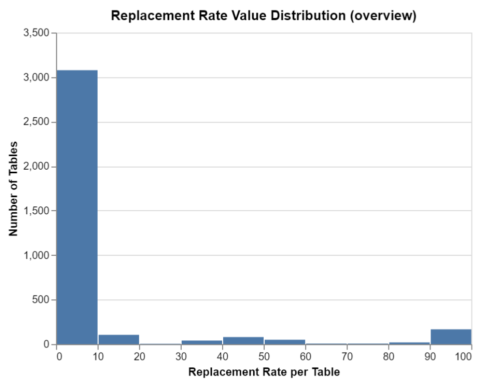


Figure 11: Distribution of the Replacement Rate Values

bution, but split into two sub-plots, the left plot shows the distribution for all rates $< 10\%$ and the right plot shows the distribution for rates $> 10\%$. As expected from the total replacement rate of 3.01%, the replacement rates of 2706, i.e. 76% of the analyzed tables, are in the range up to 3.01%. For 166 tables, between 90 and 100% of the values could be replaced.

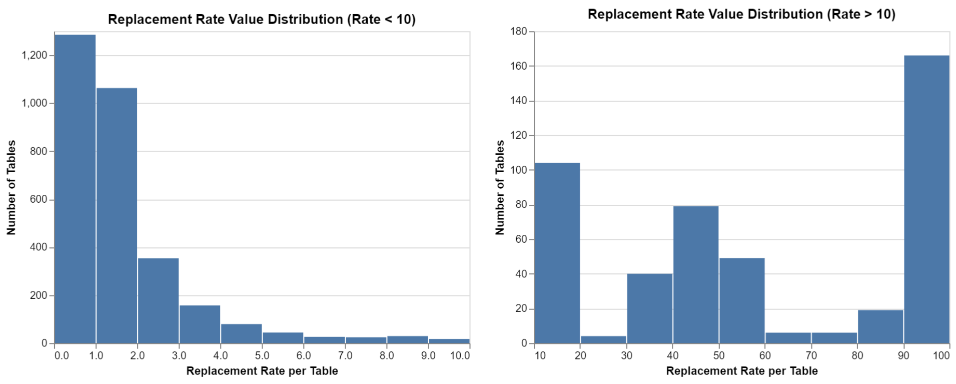
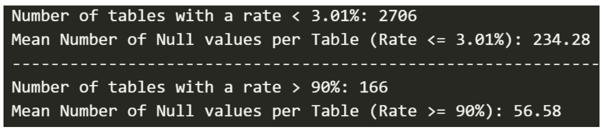


Figure 12: Distribution of the Replacement Rate Values
Left: Rates $< 10\%$ - Right: Rates $> 10\%$

However, this may be mainly related to the size or number of Null values of these tables. If these tables have on average only 56.58 Null values, those with a replacement rate $< 3.01\%$ per table have 234.28 Null values (see Figure 13)



```
Number of tables with a rate < 3.01%: 2706
Mean Number of Null values per Table (Rate <= 3.01%): 234.28
------------------------------------------------------------
Number of tables with a rate > 90%: 166
Mean Number of Null values per Table (Rate >= 90%): 56.58
```

Figure 13: More analyse Result for Replacement Rates

4.2 Analysis Replaced Values

The number of replaced Null values per table is shown in Figure 14, again most tables, about 3450, are found in the front range (0 to 50 replaced values).

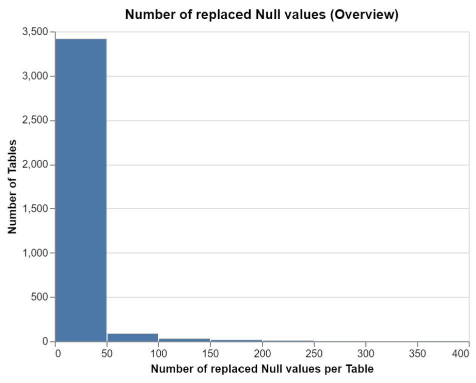


Figure 14: Distribution of the Replaced Values

A look at the left image of Figure 15 shows that also here the number of replaced values per table is mainly around the value of the average number of replaced Null values per table. Due to the fact that the tested tables have on average 206.55 Null values each,

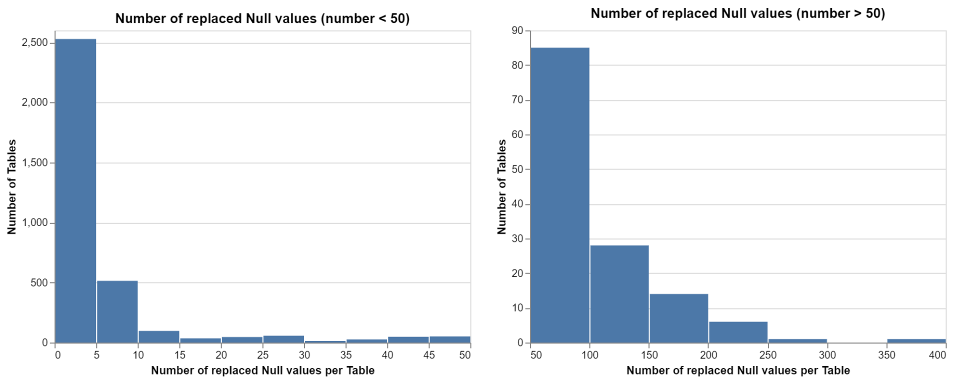
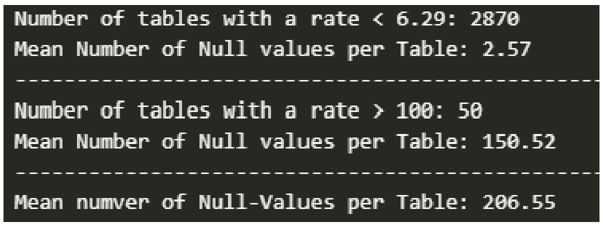


Figure 15: Distribution of the Replaced Values
Left: replaced Values < 50 - Right: replaced Values > 50

the result from the right image of Figure 15 is also understandable, here we see that there are only about 50 tables in which more than 100 values could be replaced (see Figure 16).



```
Number of tables with a rate < 6.29: 2870
Mean Number of Null values per Table: 2.57
---------------------------------------------
Number of tables with a rate > 100: 50
Mean Number of Null values per Table: 150.52
---------------------------------------------
Mean numver of Null-Values per Table: 206.55
```

Figure 16: More analyse Results for Replaced Values

All analyses performed in this section can be traced at this point in the GitHub repository: FDs and Null-Values

5 Summary

A conceivable next step at this point could be to assess the quality of the functional dependencies provided by the FD-Finder. This is especially important for the Null columns that have more than one left side.

In addition, it would also be conceivable to modify the code so that a table runs through the process several times. This would mean that with the help of already replaced values new functional dependencies can be found and thus further values can be replaced. However, the implementation of this idea only makes sense if it is possible to check how good the replaced values are. Since otherwise an error can pull itself through ever further.

Nevertheless, the results of the analysis have shown that it is possible to replace Null values with the help of functional dependencies. Of the more than one million Null values found, the program was able to replace a total of 31,841 values, which corresponds to a total replacement rate of 3.01% and an average of 6.29 replaced values per table.

The GitHub repository for this term paper can be found here: FDs and Null-Values

References

[1] P. Bohannon, W. Fan, and F. Geerts. Conditional functional dependencies for data cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 746–755, 2007.

[2] B. Breve, L. Caruccio, V. Deufemia, and G. Polese. Renuver: A missing value imputation algorithm based on relaxed functional dependencies. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 52—-64, 2022.

[3] P. Crawley and R. P. Dilworth. *Algebraic Theory of Lattices*. Prentice-Hall, Englewood Cliffs, 1 edition, 1973.

[4] Hulsebos. Gittables: A large.scale corpus of relational tables, 2023. URL `https://gittables.githup.io`.

[5] pydata.or. Pandas user guide, 2023. URL `https://pandas.pydata.org/docs/user_guide/index.html`.

[6] S. Song, Y. Sun, A. Zhang, L. Chen, and J. Wang. Enriching data imputation under similarity rule constraints. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2020.

[7] C. Wyss, C. Giannella, and E. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *Proceedings of the International Conference of Data Warehousing and Knowledge Discovery (DaWaK)*, pages 101–110, 2001.