Pre-processing Approach for Semi-Structured Medical Data

Aiman Haziq Ab Yazik[1], Azliza Mohd Ali[2*], Sharifa-lillah Nordin[3], Sazzli Shahlan Kasim[4]

[1,2,3] School of Computing Sciences, College of Computing, Informatics and Mathematics, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia
[4] Faculty of Medicine, Sungai Buloh Campus, Universiti Teknologi MARA, Malaysia
azliza@tmsk.uitm.edu.my

Abstract. — The exponential growth of healthcare data nowadays due to the widespread use of electronic health records (EHRs) has presented both opportunities and challenges in patient care, resource allocation, and medical research. Although advances in automated machine learning (AutoML) have streamlined many processes especially in data preprocessing, however the preprocessing of semi-structured data remains a time-consuming task, particularly when the data does not conform to standard database structures. This paper introduces a preprocessing approach designed to automate and simplify the preprocessing of semi-structured medical data. The approach specifically addresses the challenges posed by nested data within columns and involves a series of steps, including renaming and removing columns, merging separated rows, and expanding data in to structured formats. Through the application of this approach to an actual medical dataset, we demonstrate its effectiveness in automating the data preparation phase. The results indicate a successful transformation of nested data into a structured format, where each previously nested element is now represented by its own row and column, thereby facilitating future data analysis. The integration of this approach into healthcare data management systems has the potential to enhance the efficiency of data preprocessing and improve the quality of subsequent data analysis. Additionally, this preprocessing step can be further refined using other techniques to enhance the data. Future research should focus on expanding the preprocessing's capability to handle a wider variety of data types and structures, and exploring its applicability to other fields that encounter similar nested data challenges.

Keywords: Data Preprocessing, Semi-Structured Data, Electronic Health Records (HER), Automated Machine Learning (AutoML).

1 Introduction

The exponential growth of healthcare data presents both challenges and opportunities for improving patient care, optimizing resources, and advancing research. Electronic Health Records (EHRs) have played a key role in moving healthcare towards a paperless environment, reducing medical errors and administrative costs [1, 2]. However, the complexity and volume of EHR data create significant challenges in data analysis and interpretation [2].

Healthcare data analysis, particularly exploratory data analysis (EDA), is crucial for finding patterns, trends, and correlations. EHRs contain comprehensive patient information, including demographics, medical history, and lab results, providing valuable health insights [2]. Preprocessing techniques, such as Automated Machine Learning (AutoML), encounter several challenges when preprocessing data, especially with semi-structured data which is not usually structured like a traditional database [3].

Medical data also face challenges such as the curse of dimensionality, with numerous variables increasing complexity [4]. High-dimensional data, such as from microarrays and sequencing, require extensive preprocessing [4]. Semi-structured data, which does not fit traditional databases, further complicates machine learning and data analytics tasks [5].

This paper develops a preprocessing approach for semi-structured medical data, with a focus on handling nested data within columns. We employ a Python-based solution to create an automated and interactive process that streamlines data preprocessing. The paper is organized as follows: Section 2 reviews the related literature, Section 3 presents the methodology, Section 4 discusses the results, and Section 5 concludes the paper.

2 Literature Works

2.1 Medical Data

Medical data is often complex and comes from various sources, including electronic health records (EHRs), patient surveys, clinical trials, medical imaging, wearable devices, and healthcare databases [1, 2]. EHRs provide detailed patient histories, lab results, and treatments, offering insights into patient care and outcomes [2, 6]. However, processing this data is challenging due to its unstructured or semi-structured nature, making traditional analysis difficult [2, 7].

2.2 Semi-Structured Data

Semi-structured data, which doesn't fit neatly into traditional structured models, adds another layer of complexity. For instance, CSV files can have varying column numbers across rows, leading to inconsistencies [5, 8]. Spreadsheets, commonly used in data management, are designed for human interaction rather than automated processing, further complicating data integration [8]. Addressing preprocessing issues particularly in semi-structured data requires significant effort, which is crucial for developing reliable machine learning models [2, 8, 9].

2.3 Data Preprocessing

Data preprocessing is a critical phase in machine learning and data analysis, where raw data is cleaned, transformed, and prepared for modeling [10, 11]. High-quality preprocessing directly impacts the performance of deep learning models, with poor data quality potentially undermining algorithm outcomes [11]. Data scientists spend a substantial portion of their time on tasks such as data cleaning, missing data imputation, and categorical data encoding [3, 11]. Each of these tasks presents its own set of challenges, especially when dealing with the complexities of medical and semi-structured data.

2.4 Automated Machine Learning (AutoML)

Recognizing the time-intensive nature of data preprocessing, Automated Machine Learning (AutoML) have been developed to streamline this process. Automated Machine Learning (AutoML) streamlines the machine learning pipeline, including data preprocessing, feature engineering, model selection, and hyperparameter optimization [12–14]. AutoML automates many of the preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling features, thereby enhancing consistency and saving time [13]. However, challenges remain, particularly with semi-structured data, where the lack of standardized handling methods can hinder AutoML's scalability and effectiveness [8, 13].

2.5 Fuzzy Matching

Fuzzy matching is a technique used to identify strings that are similar but not identical, accounting for minor errors or variations [15]. It is particularly useful in data preprocessing when dealing with inconsistent or incomplete data entries. For example, fuzzy matching can address common typing errors by comparing and scoring the similarity of strings using algorithms like Levenshtein Distance (LS), Token Sort Ratio (TSR), and Jaro Distance [16]. Among these, TSR is often more effective for longer strings, as it normalizes word order and reduces variability due to different word arrangements [16].

For instance, if the target word "human" receives a response of "huwan," TSR would be calculated as TSR = 2 * (length of shared substring) / (sum of lengths of both strings) * 100 [15]. The shared substrings are "hu" and "an," so the calculation would be 2 * 4 / 10 * 100 = 80.

3 Methodology

In this research, we develop a Python-based method to provide an interactive, data-driven service for users to perform data preprocessing efficiently and effectively. The user will interact with the command line interface to generate the cleaned dataset. The method aims to preprocess semi-structured data with nested columns. We employ pre-existing approaches to address a wide range of data cleaning and feature engineering tasks. Our primary focus is on the combination of different rows and the expansion of columns into multiple columns, followed by the data cleaning process.

The most challenging aspect of designing this method is integrating the column expansion phase, where each column may contain multiple nested variables, which can occur multiple times even for a single ID. Therefore, our proposed method must handle this task efficiently.

3.1 Column Rename

In semi-structured data, the data often lacks a clear structure and column names may not be standardized. This phase involves a straightforward process to rename columns using a for loop. The program prompts the user to specify which columns need renaming and then allows the user to provide both the original and the new column names.

3.2 Column Dropper

The next step is the column dropper, a straightforward phase where the program prompts the user to specify which columns to drop. Users can choose to drop columns individually or provide a list of columns separated by commas to drop them all at once. The program will continue to prompt the user in a loop until they are satisfied with the structure of the DataFrame.

3.3 Row Combinator

The next step is the row combinator, which is crucial for merging rows that have been separated due to conversion from CSV or Excel formats. In medical systems, data entries might be written in a way that separates related information into new rows. The row combinator collects and combines all related rows into a single entry based on a common ID, ensuring the data is consolidated correctly. For example the table 1 below show what the data example looks like before it will combine. The actual data is more complex, but it is simplified here to provide a clearer understanding in this paper.

Table 1. Example of the Data

MRN	**Inpatient**	**Outpatient**	**Deceased Date**	**Echo Results**
0001	Admit Date: 2022-08-28	Clinic: Clinic ABC	-	Date Report : 2022-09-08
	Discharge Date: 2023-08-29	Date Clinic : 2020-03-09		Echo Window : Suboptimal
	Main Diagnosis: 2 Vessel Disease	Diagnosis:		
	Admit Date: 2023-01-03			
	Discharge Date: 2023-01-04			
0002	Admit Date: 2023-01-20	Clinic: Clinuc ABC	-	Date Report : 2023-02-08
	Discharge Date: 2023-01-22	Date Clinic : 2019-04-11		Echo Window : Suboptimal
	Main Diagnosis : Unstable Angina			

Figure 1 shows the flow of the row combinator phase. In this phase, the content separated by rows is combined into a single row. The program first asks the user which row represents the unique ID, then begins looping through each row in the DataFrame. For the first row, all data is taken and stored in a dictionary, and the unique ID is captured. For subsequent rows, the saved unique ID acts as an identifier to determine if the row should be combined. If no matching ID is found, or the ID is the same, the data is appended to the same dictionary. Otherwise, the dictionary is appended to the Staging DataFrame, a new unique ID is saved, and the current dictionary, which has already been appended to the Staging DataFrame, is emptied.

```
Create Raw_DataFrame from the previous modified raw dataset.
Create an empty DataFrame called Staging_DataFrame.
Create another empty dictionary called Row_Dictionary.
Create a null variable called Current_Row_ID

Take Unique_ID_Column by prompting the user for the unique id column

for row in Raw_DataFrame:
  if is_first_row(row):
    Append all row values into Row_Dictionary
    Set Current_Row_ID as row[Unique_ID_Column]
  else:
    if row[Unique_ID_Column] is the same as Current_Row_ID or row[Unique_ID_Column] is empty:
      Append all row values to Row_Dictionary
    else:
      Append Row_Dictionary into Staging_DataFrame
      Empty the Row_Dictionary
      Append new row value to Row_Dictionary
      Set Current_Row_ID as row[Unique_ID_Column]

return Staging_DataFrame
```

Fig. 1. Row Combinator

3.4 Column-Row Expansion

Next is the Column-Row Expansion Phase, where the combined data will be distributed into their respective columns and rows. This phase is extensive because the data must be accurately placed into the correct rows and columns. For instance, if there are six occurrences of a variable like "Admit Date," the program should create six new rows to accommodate each admit date, as illustrated in Figure 2 below.

In this phase, the combined data is loaded, and a new empty Staging DataFrame is created. The Staging DataFrame will then include both the new column names and the old columns after prompting the user to specify which columns need to be expanded.

The program also asks the user for the variables within the specified columns that need to be expanded, creating new columns in the Staging DataFrame accordingly. All user inputs, such as the intended columns and their variables, are stored in a dictionary.

For variables containing spaces, the program will use regular expressions and fuzzy matching based on Token Sort Ratio (TSR) to locate them. Spaces in these variables will be replaced with underscores. Other values are also identified using regular expressions and fuzzy matching with a threshold of 80% to correct any errors, such as misspellings. This has to be done to enhance the matching strategy using regular expression in the next step.

Next, the program processes each row of the DataFrame in a loop. Initially, the system creates a dictionary combining old and new column names. All data, except columns with nested elements, are appended to this dictionary. Regular expressions are generated from user input to extract nested data from each cell. Each row's result will be a variable containing an array of these nested values. The loop then appends the extracted data into the appropriate rows and columns of the Staging DataFrame. This process continues until all rows are processed, after which the program returns the Staging DataFrame as the result.

a. Initialization
- Input_DataFrame as the raw data.
- each_row_new_column, each_row_data, regex, toappend_regex, staged_cleaned_data, and column_need_expansion as empty dictionaries.
- generated_regex, row_value, and variables_to_regex as empty variables.
- cleaned_data as an empty list.

b. User Input
- The user is prompted to specify the column and its variables to be expanded. This input is stored in column_need_expansion.

c. Replacing Spaces in Variable Names
- For each variable in the targeted column, spaces are replaced with underscores using regular expressions to standardize the variable names.

d. Row-wise Processing
- Each row in Input_DataFrame is processed as follows:
    - Initialize each_row_new_column and each_row_data for the current row.
    - Create new columns in each_row_new_column for each variable in the columns specified for expansion.
    - Add all original columns to each_row_data.
    - Remove columns targeted for expansion from each_row_data.
    - Update each_row_data with each_row_new_column.

e. Generating Regular Expressions
- For each column in column_need_expansion, regular expressions are generated for the variables. These expressions identify single or multiple inputs of each variable and are stored in regex.

f. Extracting Variables Using Regular Expressions

- The generated regular expressions are applied to extract variables from the targeted column text. The results are stored in staged_cleaned_data.

g. Determining Maximum Array Length
- The maximum array length for each variable in staged_cleaned_data is determined to facilitate the expansion of rows.

h. Expanding Rows
- Rows are expanded based on the extracted data until the maximum array length is reached. The expanded rows are stored in cleaned_data.

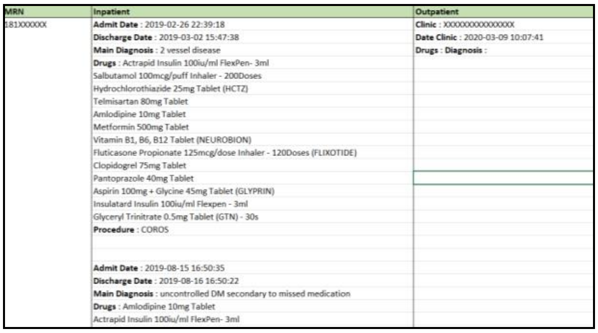
Fig. 2. Column-Row Expansion Pseudocode

4 Evaluation and Results

Given that our dataset is semi-structured and nested, we will explore a variety of feasible scenarios. We use a secondary CSV dataset provided by a medical institution, which has already been anonymized. Our aim is to handle nested and uncleaned semi-structured data.

4.1 Dataset Overview

Figure 3 below shows an example of a column from the cardiology data CSV. In this data, we can see multiple admittance dates under a single ID, each on a different row, and this pattern repeats until the next ID is encountered.



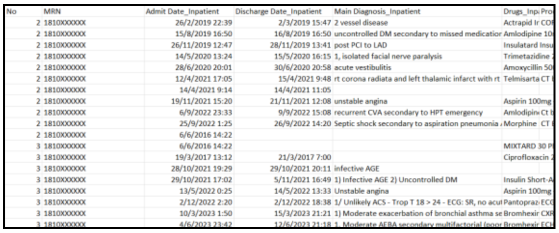
MRN	Inpatient	Outpatient
181XXXXXX	**Admit Date** : 2019-02-26 22:39:18	**Clinic** : XXXXXXXXXXXXXXX
	Discharge Date : 2019-03-02 15:47:38	**Date Clinic** : 2020-03-09 10:07:41
	Main Diagnosis : 2 vessel disease	**Drugs** : **Diagnosis** :
	Drugs : Actrapid Insulin 100iu/ml FlexPen- 3ml	
	Salbutamol 100mcg/puff Inhaler - 200Doses	
	Hydrochlorothiazide 25mg Tablet (HCTZ)	
	Telmisartan 80mg Tablet	
	Amlodipine 10mg Tablet	
	Metformin 500mg Tablet	
	Vitamin B1, B6, B12 Tablet (NEUROBION)	
	Fluticasone Propionate 125mcg/dose Inhaler - 120Doses (FLIXOTIDE)	
	Clopidogrel 75mg Tablet	
	Pantoprazole 40mg Tablet	
	Aspirin 100mg + Glycine 45mg Tablet (GLYPRIN)	
	Insulatard Insulin 100iu/ml Flexpen - 3ml	
	Glyceryl Trinitrate 0.5mg Tablet (GTN) - 30s	
	Procedure : COROS	
	Admit Date : 2019-08-15 16:50:35	
	Discharge Date : 2019-08-16 16:50:22	
	Main Diagnosis : uncontrolled DM secondary to missed medication	
	Drugs : Amlodipine 10mg Tablet	
	Actrapid Insulin 100iu/ml FlexPen- 3ml	

Fig. 3. Dataset Overview

For example, the columns include "Inpatient" and "Outpatient," which can repeat multiple times until the next ID is found. This dataset can be separated by each Admit Date, Discharge Date, Main Diagnosis, Drugs, and Procedure for the Inpatient column. From this example, the new columns would be Admit Date_Inpatient, Discharge Date_Inpatient, Main Diagnosis_Inpatient, Drugs_Inpatient, and Procedure_Inpatient.

4.2 Processed Dataset Overview

Figure 4 shows the results after the semi-structured data has been processed through the preprocessing pipeline. The previously uncleaned data has been organized into their respective rows and columns. In this figure, we can see that the MRN serves as the unique ID for each patient in the dataset. This data is now ready for the next pre-processing step, which involves handling the imputation of missing data and additional data normalization.



No	MRN	Admit Date_Inpatient	Discharge Date_Inpatient	Main Diagnosis_Inpatient	Drugs_Inp	Pro
2	1810XXXXXX	26/2/2019 22:39	2/3/2019 15:47	2 vessel disease	Actrapid Ir	COF
2	1810XXXXXX	15/8/2019 16:50	16/8/2019 16:50	uncontrolled DM secondary to missed medicatior	Amlodipine 10	
2	1810XXXXXX	26/11/2019 12:47	28/11/2019 13:41	post PCI to LAD	Insulatard Insu	
2	1810XXXXXX	14/5/2020 13:24	15/5/2020 16:15	1, isolated facial nerve paralysis	Trimetazidine 2	
2	1810XXXXXX	28/6/2020 20:01	30/6/2020 20:58	acute vestibulitis	Amoxycillin 50	
2	1810XXXXXX	12/4/2021 17:05	15/4/2021 9:48	rt corona radiata and left thalamic infarct with rt	Telmisarta	CT I
2	1810XXXXXX	14/4/2021 9:14	14/4/2021 11:05			
2	1810XXXXXX	19/11/2021 15:20	21/11/2021 12:08	unstable angina	Aspirin 100mg	
2	1810XXXXXX	6/9/2022 23:33	9/9/2022 15:08	recurrent CVA secondary to HPT emergency	Amlodipin	Ct b
2	1810XXXXXX	25/9/2022 1:25	26/9/2022 14:20	Septic shock secondary to aspiration pneumonia	Morphine	CT I
2	1810XXXXXX	6/6/2016 14:22				
3	1810XXXXXX	6/6/2016 14:22			MIXTARD 30 P	
3	1810XXXXXX	19/3/2017 13:12	21/3/2017 7:00		Ciprofloxacin 2	
3	1810XXXXXX	28/10/2021 19:29	29/10/2021 20:11	infective AGE		
3	1810XXXXXX	29/10/2021 17:02	5/11/2021 16:49	1) Infective AGE 2) Uncontrolled DM	Insulin Short-A	
3	1810XXXXXX	13/5/2022 0:25	14/5/2022 13:33	Unstable angina	Aspirin 100mg	
3	1810XXXXXX	2/12/2022 2:20	2/12/2022 18:38	1/ Unlikely ACS - Trop T 18 > 24 - ECG: SR, no acut	Pantopraz	ECG
3	1810XXXXXX	10/3/2023 1:50	15/3/2023 21:21	1) Moderate exacerbation of bronchial asthma se	Bromhexir	CXR
3	1810XXXXXX	4/6/2023 23:42	12/6/2023 21:18	1. Moderate AEBA secondary multifactorial (poor	Bromhexir	ECH

Fig. 4. After Preprocessing

5 Conclusions

In conclusion, our study commenced with meticulous initial data preprocessing steps, including renaming columns, dropping unnecessary columns, combining rows into a single row, and expanding rows based on nested data, before implementing automated procedures. During the Column-Row Expansion phase, we effectively managed data preprocessing using regular expressions and fuzzy matching of TSR to separate the nested value in the cell. Moving forward, we aim to introduce additional automation in subsequent phases, including data cleaning, scaling, feature selection, and feature extraction, as well as handling a broader range of data types and structures, to enhance the data's suitability for analysis and machine learning models. Additionally, if the semi-structured data exhibits repeated structures across rows within a column, this approach can be applied to other fields beyond medical data, effectively transforming semi-structured data into a structured format.

Acknowledgments. The authors would like to express gratitude to the School of Computing Sciences, College of Computing, Informatics and Mathematics, Hospital Al Sultan Abdullah, and Faculty of Medicine, Universiti Teknologi MARA for all the given support.

Disclosure of Interests. The authors declare that there are no competing interests.

References

1. Sáinz-Pardo Díaz J, López García Á (2022) A Python library to check the level of anonymity of a dataset. Sci Data 9:. https://doi.org/10.1038/s41597-022-01894-2

2. Fathima Shah W (2021) Data Preprocessing in Healthcare: A Vital Step towards Informed Decision-Making Article in. International Journal of Science and Research. https://doi.org/10.21275/SR231226164816

3. Bilal M, Ali G, Iqbal MW, et al (2022) Auto-Prep: Efficient and Automated Data Preprocessing Pipeline. IEEE Access 10:107764–107784. https://doi.org/10.1109/ACCESS.2022.3198662

4. Lee CH, Yoon HJ (2017) Medical big data: Promise and challenges. Kidney Res Clin Pract 36:3–11. https://doi.org/10.23876/j.krcp.2017.36.1.3

5. van Engelen JE, Hoos HH (2019) A Survey on Semi-Supervised Learning. Mach Learn. https://doi.org/10.1007/s10994-019-05855-6

6. Sheikhalishahi S, Miotto R, Dudley JT, et al (2019) Natural Language Processing of Clinical Notes on Chronic Diseases: Systematic Review. JMIR Med Inform. https://doi.org/10.2196/12239

7. Ladas N, Borchert F, Franz S, et al (2023) Programming Techniques for Improving Rule Readability for Rule-Based Information Extraction Natural Language Processing Pipelines of Unstructured and Semi-Structured Medical Texts. Health Informatics J. https://doi.org/10.1177/14604582231164696

8. Bonfitto S, Cappelletti L, Trovato F, et al (2021) Semi-automatic Column Type Inference for CSV Table Understanding. In: Bures T, Dondi R, Gamper J, et al (eds) SOFSEM 2021: Theory And Practice Of Computer Science. Springer International Publishing Ag, Gewerbestrasse 11, Cham, Ch-6330, Switzerland, pp 535–549

9. Sun W, Cai Z, Li Y, et al (2018) Data Processing and Text Mining Technologies on Electronic Medical Records: A Review. J Healthc Eng. https://doi.org/10.1155/2018/4302425

10. Fan C, Chen M, Wang X, et al (2021) A Review on Data Preprocessing Techniques Toward Efficient and Reliable Knowledge Discovery From Building Operational Data. Front Energy Res 9

11. Mumuni A, Mumuni F (2024) Automated data processing and feature engineering for deep learning and big data applications: A survey. Journal of Information and Intelligence. https://doi.org/10.1016/j.jiixd.2024.01.002

12. Li PR, Chen Z, Chu X, Rong K (2023) DiffPrep: Differentiable Data Preprocessing Pipeline Search for Learning Over Tabular Data. Proceedings of the Acm on Management of Data. https://doi.org/10.1145/3589328

13. Truong A, Walters A, Goodsitt J, et al (2019) Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools. https://doi.org/10.1109/ICTAI.2019.00209

14. Pečnik L, Fister I (2021) NiaAML: AutoML Framework Based on Stochastic Population-Based Nature-Inspired Algorithms. The Journal of Open Source Software. https://doi.org/10.21105/joss.02949

15. Bosker HR Using fuzzy string matching for automated assessment of listener transcripts in speech intelligibility studies. https://doi.org/10.3758/s13428-021-01542-4/Published

16. Kleshch K, Shablii V (2023) Comparison of fuzzy search algorithms based on Damerau-Levenshtein automata on large data. Technology audit and production reserves 4:27–32. https://doi.org/10.15587/2706-5448.2023.286382