CLEANING DATA WITH FORBIDDEN ITEMSETS

A PROJECT REPORT

Submitted in the partial fulfillment of requirements to

RVR&JC COLLEGE OF ENGINEERING

For the award of the degree
B.Tech. in CSE

By

Paladugu Dakshayani(Y20CS130)
Meer Luqman Hussaini(Y20CS116)
Madala Kavya(Y20CS102)
Kanakam Gnana Sindhura Lahari(Y20CS075)

May-2024

R.V.R. & J.C. COLLEGE OF ENGINEERING (Autonomous)
(Affiliated to Acharya Nagarjuna University)
Chandramoulipuram::Chowdavaram
GUNTUR-522 019

R.V.R. & J.C. COLLEGE OF ENGINEERING
(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that this project work titled **"Cleaning Data With Forbidden Itemsets"** is the work done by Paladugu Dakshayani (Y20CS130), Meer Luqman Hussaini (Y20CS116), Kanakam Gnana Sindhura Lahari (Y20CS075) and Madala Kavya(Y20CS102) under my supervision, and submitted in partial fulfillment of the requirements for the award of the degree, B.Tech. in Computer Science & Engineering, during the Academic Year **2023-2024.**

Sri. Ch. Ratna Babu	**Dr. B. Vara Prasad Rao**	**Dr.M.Sreelatha**
Project Guide	**In-charge, Project Work**	**Prof.&Head**

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without a proper suggestion, guidance and environment. Combination of these three factors act like backbone to our Project **"Cleaning Data With Forbidden Itemsets".**

We would like to express our special thanks to our guide **Sri.Ch.Ratna Babu**, who has inspired us to select this topic and also for his valuable advice in preparing this project report.

We are very thankful to **Dr.B.Vara Prasad Rao**, Project In-charge who extended his encouragement and support to carry out this project successfully.

We are very glad to express our special thanks to **Dr. M. Sreelatha**, Professor & HOD, CSE, who encouraged and supported to carry out this report.

We are very much thankful to **Dr. K.Srinivas**, Principal of R.V.R. & J.C. College of Engineering, Guntur for providing a supportive Environment.

Finally, we would like to submit our reserved thanks to the lab staff in the Department of Computer Science and Engineering and to all our friends for their cooperation during this project work preparation.

With utmost respect and gratitude,

Paladugu Dakshayani(Y20CS130)

Meer Luqman Hussaini(Y20CS116)

Madala Kavya(Y2OCS102)

Kanakam Gnana Sindhura Lahari(Y20CS075)

ABSTRACT

Methods for cleaning dirty data typically employ additional information about the data such as user-provided constraints specifying when data is dirty, for example, domain restrictions, illegal value combinations, or logical rules. However, real-world scenarios usually only have dirty data available, without known constraints. In such settings, constraints are automatically discovered on dirty data, and discovered constraints are used to detect and repair errors. Typical repairing processes stop there. Yet, when constraint discovery algorithms are re-run on the repaired data (assumed to be clean), new constraints and thus errors are often found. The repairing process then introduces new constraint violations. A different type of repairing method is presented, which prevents introducing new constraint violations, according to a discovery algorithm. Summarily, repairs guarantee that all errors identified by constraints discovered on the dirty data are fixed, and the constraint discovery process cannot identify new constraint violations. This is done for a new kind of constraints, called forbidden itemsets (FBIs), capturing unlikely value co-occurrences. It is shown that FBIs detect errors with high precision. Evaluation on real-world data shows that the repair method obtains high-quality repairs without introducing new FBIs. Optional user interaction is readily integrated, with users deciding how much effort to invest

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.No	Abbreviation	Full Form
1	FBI	Forbidden Itemset
2	CFD	Conditional Functional Dependency
3	cCFD	Constant Conditional Functional Dependency
4	DC	Denial Constraints
5	cDC	Constant Denial Constraints
6	BART	Bayesian Additive Regression Trees
7	MBA	Market Based Analysis
8	A-FBI	Almost Forbidden Itemset
9	FP-Growth	Frequent Pattern Growth
10	ARM	Association Rule Mining
11	DQ	Data Quality
12	KDD	Knowledge Discovery in Databases

CHAPTER 1

INTRODUCTION

1.1 Introduction to Data Cleaning

In recent years, research on detecting inconsistencies in data has focused on a constraint-based data quality approach: a set of constraints in some logical formalism is associated with a database, and the data is considered consistent or clean if and only if all constraints are satisfied. Many such formalisms exist, capturing a wide variety of inconsistencies, and systematic ways of repairing the detected inconsistencies are in place. The constraints are either supplied by experts, or automatically discovered from the data. Once in place, they are treated as a gold standard, both complete and correct, and are subsequently used for repairing the data such that the repair satisfies the constraints. Suppose that the used constraints were discovered on the dirty data, then this data is now changed by repairing. It is likely that running the constraint discovery algorithm again on the repair would result in the discovery of different constraints and errors. In other words, using the logic from before, our repair is not so clean after all.

To remedy this situation, one may consider using an iterative approach in which the repairing and discovery steps are repeated until no further constraints can be found. First of all, there is no guarantee that this process terminates since existing repair algorithms do not take into account the discovery process. Repair modifications may result in more constraints (and thus errors) to be discovered than from which one started. Second, constraint discovery is a time consuming task (typically exponential in the number of attributes), resulting in a slow data cleaning process.

A repairing algorithm is proposed that is closely tied to constraint discovery. In particular, the main aim is to find repairs that are clean with respect to a dynamic notion of data quality in which a database is regarded to be clean if a constraint discovery algorithm does not detect any violated constraints on that data.

1.1.1 Introduction to Forbidden Itemsets in Data Cleaning

Forbidden itemsets, in data cleaning, refer to sets of attributes or features that are prohibited from co-occurring in a dataset due to certain constraints or rules. These constraints can include privacy regulations, business policies, or domain-specific requirements. Detection and handling of forbidden itemsets are crucial steps in ensuring data quality and integrity, as they help mitigate the risk of erroneous analyses or decisions based on prohibited data patterns.

Forbidden itemsets play a critical role in identifying and removing inconsistencies, anomalies, or violations of business rules within datasets. By identifying attribute combinations that are not allowed according to predefined rules, forbidden itemsets contribute to maintaining compliance with legal and regulatory standards, preserving data privacy and confidentiality, and preventing adverse effects on decision-making processes and business operations. In real-world scenarios, forbidden itemsets are commonly encountered in various domains, including healthcare, finance, retail, and telecommunications, where strict regulations or business requirements govern the permissible combinations of attributes.

Understanding the characteristics and types of forbidden itemsets, along with their practical applications in data cleaning processes, is essential for data analysts and decision-makers to ensure the accuracy, reliability, and integrity of their data-driven insights. Therefore, exploring methods for detecting, analyzing, and addressing forbidden itemsets is crucial for effectively managing and leveraging datasets for informed decision-making and business success.

In addition to their role in ensuring data quality and compliance, forbidden itemsets also serve as indicators of potential data anomalies or inconsistencies that may require further investigation. By identifying patterns of attribute combinations that violate predefined constraints, forbidden itemsets can help uncover underlying issues such as data entry errors, system failures, or fraudulent activities.

1.1.2 Introduction to types of forbidden Itemsets

Forbidden itemsets can be categorized into two main types: single and composite. Understanding these types is essential for effectively identifying and managing forbidden itemsets in data cleaning processes.

Single Forbidden Itemsets:

- Definition: Single forbidden itemsets consist of pairs of attributes or features that are prohibited from co-occurring in a dataset.
- Characteristics: These itemsets typically represent straightforward constraints or rules that specify certain attribute combinations as invalid.
- Example: In a dataset containing patient records, a single forbidden itemset might consist of the combination of a patient's age and a diagnosis code that violates privacy regulations.
- Application: Single forbidden itemsets are commonly used to enforce simple data governance policies and ensure compliance with regulatory requirements.

Composite Forbidden Itemsets:

- Definition: Composite forbidden itemsets comprise multiple attributes or features that are prohibited from occurring together in a dataset.
- Characteristics: These itemsets represent more complex constraints or rules that involve combinations of three or more attributes.
- Example: In a retail dataset, a composite forbidden itemset could involve a combination of product attributes such as category, brand, and price range that are incompatible or restricted by business policies.
- Application: Composite forbidden itemsets are often used to enforce intricate business rules, product compatibility constraints, or operational requirements within organizations.

Understanding the distinctions between single and composite forbidden itemsets is crucial for implementing effective data cleaning strategies. While single itemsets address straightforward constraints, composite itemsets handle more complex scenarios.

1.1.3 Challenges in handling Forbidden Itemsets

Detecting and managing forbidden itemsets in data cleaning processes present several challenges that organizations must overcome to ensure the integrity and reliability of their datasets.One significant challenge is scalability. As datasets grow in size and complexity, the computational resources required to detect and handle forbidden itemsets also increase. Traditional algorithms for itemset mining may struggle to cope with large volumes of data, leading to performance bottlenecks and longer processing times. Addressing scalability issues often requires the development of efficient algorithms and optimization techniques capable of handling big data environments effectively.

Another challenge is the presence of conflicting constraints. In real-world scenarios, datasets may be subject to multiple constraints or rules that overlap or contradict each other. For example, a dataset may contain both privacy regulations prohibiting the sharing of certain attributes and business rules requiring the use of those same attributes for analysis. Resolving conflicting constraints requires careful consideration and prioritization to ensure that data cleaning processes adhere to the most critical requirements while minimizing the risk of violating others.

Furthermore, handling large datasets poses its own set of challenges. As datasets grow in size, the complexity of detecting and managing forbidden itemsets increases exponentially. Processing large datasets requires efficient algorithms and distributed computing techniques capable of handling the volume, velocity, and variety of data sources. Additionally, organizations must consider the storage and computational costs associated with processing and analyzing large datasets, as well as the potential impact on performance and scalability.

Overall, addressing these challenges requires a combination of innovative algorithms, scalable infrastructure, and domain-specific expertise. By overcoming these obstacles, organizations can effectively detect and manage forbidden itemsets in their datasets, ensuring data quality, integrity, and compliance with regulatory requirements.

1.2 Problem Definition

Given:

Transactional data across different industries (like retail, healthcare, finance etc.) often contains inconsistencies and errors, one specific kind being the occurrence of "forbidden itemsets". These are specific combinations of items that, due to various constraints like legal restrictions, store policies or health risks, should not appear together within a single transaction.

Task:

The task is to design an algorithm that can effectively identify these forbidden itemsets within the transactional data, and subsequently correct the data. The algorithm must be flexible and adaptable to handle evolving definitions of forbidden itemsets and must accommodate large volumes of data.

Approach:

The algorithm should first define what constitutes a forbidden itemset within the context of the specific industry, potentially requiring input from domain experts. Then, it should efficiently scan the data and identify instances where these forbidden itemsets occur. Once identified, these instances need to be corrected in a manner that preserves the integrity of the rest of the data. The algorithm should consider efficiency, scalability, and robustness during its design and implementation.

Evaluation:

The ultimate goal is to have a robust and scalable data cleaning algorithm that effectively improves the quality and reliability of the transactional data, facilitating more accurate and meaningful analyses and insights. The outcome of such a data cleaning process is expected to improve the quality and reliability of the data, facilitating more accurate and meaningful analyses and insights.The approach was evaluated on a number of datasets. The results showed that the approach was able to detect errors with a high accuracy.

1.3 Significance of the Work

- Cleaning dirty data often relies on additional information like user-provided constraints.
- Real-world scenarios frequently lack known constraints, relying on automatic discovery from dirty data.
- Discovered constraints help in error detection and repair during data cleaning.
- However, rerunning constraint discovery on repaired data can reveal new errors, leading to new constraint violations.
- A novel repair method is proposed to prevent introducing new constraint violations, guided by a discovery algorithm.
- This approach ensures that errors identified by constraints on dirty data are fixed without introducing new violations.
- The method focuses on forbidden itemsets (FBIs) to capture unlikely value co-occurrences, effectively detecting errors with high precision.
- Evaluation on real-world data demonstrates the effectiveness of the repair method in achieving high-quality repairs without introducing new FBIs.
- Optional user interaction allows users to decide the level of effort to invest in the cleaning process.
- The method emphasizes the importance of maintaining data integrity throughout the cleaning process.
- By preventing the introduction of new constraint violations, it ensures that the data remains compliant with discovered constraints.
- The use of forbidden itemsets (FBIs) adds an additional layer of sophistication to error detection, capturing subtle patterns of unlikely value co-occurrences.
- The evaluation of real-world data underscores the practical applicability and effectiveness of the proposed repair method.
- The integration of optional user interaction provides flexibility and adaptability to accommodate varying levels of user involvement in the data cleaning process.
- The approach contributes to advancing the field of data cleaning by addressing the challenge of constraint violation propagation during the repair process.

CHAPTER 2
LITERATURE SURVEY

2.1 Review of the Project

[6] X. Chu, I.F.Ilyas and P. Papotti Denial constraints (DCs) are a type of data quality rule that express conditions that cannot hold over a given database. They were introduced to generalize other types of constraints like tuple-based and equality generating dependencies. The essence of a denial constraint is to deny the existence of certain scenarios in the data. For instance, a denial constraint could state that there cannot be two rows in a database that have the same value in the 'Email' column (to ensure unique email addresses).In the context of data cleaning, the discovery of denial constraints can be quite useful.

These constraints can help to identify inconsistencies or errors in the data. If a denial constraint is violated, this indicates that there is an error that needs to be addressed.The process of discovering denial constraints typically involves analyzing the dataset and applying algorithms or techniques to identify conditions that do not hold over the data. Once identified, these constraints can be used to guide the data cleaning process.However, discovering denial constraints can be computationally expensive, especially for large datasets with many attributes.

In addition, defining the right denial constraints often requires significant domain knowledge.Once the denial constraints are identified and defined, they can be used to scan the database for violations, which can then be corrected. This can improve the quality and reliability of the data.Despite their usefulness, as of my training cut-off in September 2021, the discovery of denial constraints does not directly address the concept of forbidden itemsets.

Forbidden item sets are a kind of constraint, but they specifically refer to sets of items that cannot occur together in a single transaction, which is a different context than the one denial constraints usually apply to Further research and development would be required to adapt denial constraint discovery techniques for the task of identifying and

cleaning forbidden itemsets.In addition to the challenges of discovering denial constraints and their relevance to forbidden itemsets, it's essential to consider the interpretability and manageability of these constraints in practical data cleaning scenarios. Denial constraints can sometimes be complex and difficult to interpret, especially when dealing with large datasets or intricate data structures. Ensuring that denial constraints are understandable and manageable by data practitioners is crucial for their effective implementation in data cleaning processes.Therefore, research efforts should also focus on developing user-friendly interfaces and tools that facilitate the creation, interpretation, and management of denial constraints, enabling data professionals to leverage them effectively for improving data quality and integrity.

Advantages:

- **Error Detection:** Denial constraints can be very effective for detecting errors in data. If a denial constraint is violated, it signifies that there's an inconsistency in the data.
- **Generalization:** Denial constraints can generalize other types of constraints like tuple-based and equality generating dependencies, making them a versatile tool for expressing a wide variety of conditions that should not hold in the data.
- **Data Quality:** By enforcing denial constraints and resolving violations, you can improve the quality of your data. This can lead to more accurate and reliable analyses.

Disadvantages:

- **Defining Constraints:** The definition of denial constraints can be a challenging task that often requires domain expertise. It can be difficult to specify what conditions should not hold in the data, especially in complex datasets.
- **Computational Expense:** Checking for violations of denial constraints can be computationally expensive, particularly for large datasets with many attributes. This could be a limiting factor in some contexts.
- **Resolution of Violations:** While detecting violations of denial constraints can be straightforward, resolving these violations can be more challenging. It may not always be clear what the best course of action is when a violation is detected.

- **Specificity for Forbidden Itemsets:** While denial constraints can be used to express a wide variety of conditions, they do not directly address the concept of forbidden itemsets.

[11] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, Conditional Functional Dependencies (CFDs) are a type of data integrity constraint that extend traditional Functional Dependencies (FDs) by adding a condition or predicate that needs to be satisfied. This allows for a more fine-grained specification of data dependencies, making them particularly useful for data cleaning. Here's a step-by-step process of how data cleaning can be done using Conditional Functional Dependencies:

1. **Defining Conditional Functional Dependencies:** The first step involves defining the CFDs for the specific database or dataset. CFDs are specified as a pair of a predicate and a traditional FD. The predicate defines the conditions under which the FD holds. For example, a CFD might be that if a customer is from the U.S., their phone number should follow a specific format.
2. **Constraint Checking:** Once the CFDs are defined, they are applied to the data. This involves scanning through the database or dataset and checking if the CFDs are violated.
3. **Violation Detection:** If a CFD is violated, it signifies that there's an inconsistency in the data. These violations are logged and marked for further action. For example, if a U.S. customer's phone number doesn't follow the expected format, it's marked as a violation.
4. **Resolution:** The next step is to resolve the violations. Depending on the nature of the violation, resolution could involve correcting the data to adhere to the expected format or structure defined by the CFD.
5. **Validation:** Once the violations are resolved, the data should be checked again to validate that all issues have been addressed and that no new issues were introduced during the resolution step.
6. **Iteration:** Data cleaning is an iterative process. It's possible that the resolution of some violations might lead to new ones, so steps 2-5 might need to be repeated until no more violations are found.

By using CFDs in data cleaning, the process can capture inconsistencies that traditional FDs might miss.However, defining appropriate CFDs can be complex and requires domain knowledge, and the process can be computationally expensive with large datasets.In conclusion, while Conditional Functional Dependencies (CFDs) offer significant advantages in capturing fine-grained data inconsistencies and enhancing error detection in data cleaning processes, their effectiveness is balanced by challenges such as the complexity of defining constraints, computational expense, limited scope in capturing diverse data structures, manual effort required for crafting meaningful CFDs, and the potential for dependency conflicts. Despite these challenges, leveraging CFDs in data cleaning can lead to improved data quality and reliability, facilitating more accurate and insightful data analyses.

Advantages:

- **Fine-grained Constraints:** CFDs extend traditional functional dependencies by adding predicates. This allows for more specific and fine-grained data constraints, enabling the capture of more detailed and context-specific data dependencies.
- **Enhanced Error Detection:** By defining specific conditions under which certain dependencies should hold, CFDs can be highly effective in detecting inconsistencies and errors that traditional Functional Dependencies might miss.
- **Data Quality:** By enforcing CFDs and resolving violations, you can improve the quality and reliability of your data, leading to more accurate subsequent analyses.

Disadvantages:

- **Defining Constraints:** Defining CFDs can be complex and requires a considerable amount of domain knowledge. Identifying the conditions and dependencies that should hold in your data can be challenging, especially in intricate or domain-specific datasets.
- **Computational Expense:** Checking for violations of CFDs can be computationally intensive, particularly for large datasets with numerous attributes and complex conditions. This could be a limiting factor in certain contexts.
- **Limited Scope:** While CFDs are capable of capturing a certain level of complexity in data dependencies, they might not fully capture all complexities, particularly in highly diverse or non-relational data structures. For instance,

nested or hierarchical relationships might not be adequately represented through CFDs.

- **Manual Effort:** Crafting accurate and meaningful CFDs often requires substantial manual effort and a deep understanding of the underlying domain. It's not just a matter of defining the rules, but understanding the data and the relationships within it. This could be time-consuming and challenging, particularly for large, complex datasets.
- **Dependency Conflict:** There may be scenarios where multiple CFDs conflict with each other, leading to ambiguities and complexities in the data cleaning process.

[29] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang, Interactive and Deterministic Data Cleaning is a technique that involves the direct involvement of users or domain experts in the process of data cleaning and it follows a set of deterministic rules to identify and correct errors in the data.Here's how the process generally works:

1. **Defining Cleaning Rules:** The first step involves defining deterministic cleaning rules. These rules identify what constitutes an error in the data and what corrections should be applied. For example, a rule might state that all phone numbers must follow a specific format.
2. **Automated Cleaning**: The data is then processed based on these deterministic rules. This step can often be automated to scan through the entire dataset and apply corrections where necessary.
3. **Interactive Review:** Despite the deterministic nature of the rules, there can often be instances where automated cleaning might not be appropriate or sufficient. In such cases, flagged potential errors are presented to the users or domain experts for manual review. This allows the users to make judgments based on their expertise and knowledge of the data.

4. **User Corrections:** The users can apply corrections, confirm that flagged instances are not errors, or even redefine the cleaning rules based on the instances that they review.
5. **Iterative Process:** The process is often iterative, with steps 2-4 being repeated until the data reaches an acceptable level of quality. The cleaning rules may be refined over time based on the user feedback and the types of errors encountered in the data.

The advantage of this approach is that it combines the efficiency of automated, rule-based cleaning with the nuanced understanding and decision-making capabilities of human users. However, it can be time-consuming, particularly for large datasets, and heavily relies on the availability of users or domain experts for the interactive review and correction steps. As of my last training cut-off in September 2021, the technique does not specifically address the concept of forbidden itemsets. While the deterministic rules could potentially be set up to flag forbidden itemsets, this would require additional development and adaptation of the technique.

[22] F Chiang and R. J. Miller, proposed "A Unified Model for Data and Constraint Repair" is an innovative approach towards data cleaning that takes into account potential errors both in the data and in the constraints applied over it. Traditionally, data cleaning focuses on adjusting data to fit pre-defined constraints, but this model acknowledges that constraints themselves can also be flawed. Starting with defined constraints, the model identifies violations where data doesn't adhere to these constraints. Typically, the next step would be to modify the data to fit the constraints. However, in this unified model, there's an additional step: constraint repairing. This novel step posits that some constraints might be incorrect or overly restrictive and allows for their modification or removal based on domain knowledge or empirical evidence.

This dual process of data and constraint repair is iterative and continues until an acceptable level of data quality is reached. However, it can be complex and requires significant domain expertise, particularly for constraint repairing.The "Unified Model for Data and Constraint Repair" represents a significant advancement in the field of data cleaning by addressing both data and constraint errors in an iterative and comprehensive

manner. By acknowledging that constraints themselves can be flawed, this model introduces a novel step of constraint repairing alongside traditional data modification processes. This dual approach allows for the identification and rectification of errors on two fronts, ultimately leading to improved data quality and integrity.

Advantages:

- The "Unified Model for Data and Constraint Repair" provides a comprehensive and flexible approach to data cleaning by addressing both data and constraint errors in an iterative process.

Disadvantages:

- This model, while powerful, can be complex and computationally intensive to implement, requires significant domain knowledge for constraint repairs, and as of my last training cut-off in September 2021, does not specifically handle forbidden itemsets, which may require additional development.
- The model's effectiveness heavily depends on the quality of initial constraints and the proficiency of the domain experts involved in the constraint repair process; incorrect modifications or removals could potentially lead to a decline in data quality.

[25] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, Continuous Data Cleaning is an approach to maintaining data quality that involves consistently monitoring and cleaning data in near real-time or at regular intervals, rather than treating data cleaning as a one-time operation.Here's a generalized view of the steps involved in this process:

1. **Setting up the Rules:** The first step involves setting up rules or constraints for data validation. These could be simple rules such as range constraints for numerical values, format constraints for string values, or more complex rules involving multiple attributes.
2. **Real-time Monitoring and Validation:** As new data comes in, it is immediately validated against these rules. This could be achieved by integrating data validation into the data ingestion pipeline, or by setting up a separate process that checks new data as it arrives.

3. **Error Detection and Logging:** If a piece of data violates the rules, it is flagged as an error. Details of the error are logged for review and correction.
4. **Correction:** The flagged errors are then corrected. This could involve simple corrections like formatting a date correctly, or more complex processes that require domain knowledge or manual intervention.
5. **Feedback Loop:** The corrections are fed back into the system, which can help improve the rules and the cleaning process over time. This can lead to more effective and efficient data cleaning in the future.

Continuous Data Cleaning can be highly effective in maintaining data quality, especially in environments where new data is constantly being generated or updated, such as in streaming applications or live databases. However, it can also be complex to set up and manage, and may require significant computational resources depending on the volume and velocity of the data. As of my last training cut-off in September 2021, the technique does not specifically address the concept of forbidden itemsets. While rules could potentially be set up to flag forbidden itemsets, this would require additional development and adaptation of the technique.Continuous data cleaning offers numerous advantages, including the ability to maintain data quality in dynamic and rapidly changing environments. By continuously monitoring and cleaning data, organizations can ensure that their datasets remain accurate, reliable, and up-to-date.

[14] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, proposed **"Dynamic itemset counting and implication rules for market basket data,".**

"Dynamic Itemset Counting and Implication Rules for Market Basket Data" is an approach that addresses data mining in the context of Market Basket Analysis (MBA). This technique is specifically geared towards efficiently discovering patterns and relationships among items in large transactional databases (such as purchase history data from a supermarket).Here's a simplified explanation of how the process generally works:

1. **Itemset Counting:** The technique begins by dynamically counting itemsets in the database. An itemset refers to a set of one or more items. The goal is to identify

frequently occurring itemsets, i.e., sets of items that appear together in a significant number of transactions.

2. **Implication Rule Generation:** Once frequent itemsets are identified, implication rules (also known as association rules) are generated. These rules represent the discovered patterns and suggest a strong relationship between items. For example, an implication rule could be "If a customer buys bread and butter, they are likely to buy milk as well". These rules can be invaluable for decision-making processes such as product placement, sales strategy, and recommendation systems.
3. **Evaluation of Rules:** The derived rules are evaluated using measures such as confidence (how often the rule has been found to be true) and support (the proportion of transactions that include the itemset).
4. **Update as New Data Arrives:** One of the notable features of this approach is its dynamic nature. As new transaction data arrives, the itemset counts and implication rules are updated to reflect the latest data. This makes it particularly suitable for scenarios where data is continually updated.

In the context of data cleaning, this technique doesn't directly address the problem. However, it could potentially be adapted to identify forbidden itemsets by looking for itemsets that should not appear together based on the generated implication rules. By analyzing the generated implication rules, data analysts can uncover patterns and associations within the dataset that may indicate data inconsistencies or anomalies.Additionally, the dynamic nature of the "Dynamic Itemset Counting and Implication Rules" approach offers flexibility in adapting to evolving datasets, making it suitable for scenarios where data undergoes frequent updates or changes.

The adaptability is particularly advantageous in data cleaning processes where the dataset may vary over time due to new entries, modifications, or deletions. By continuously updating the itemset counts and implication rules, the system can effectively capture changes in data patterns and relationships, facilitating ongoing data cleaning efforts and ensuring the accuracy and relevance of the discovered rules.

2.2 Limitations of Existing Techniques

1. **Scalability Issues:** Existing techniques may struggle to efficiently handle the sheer volume of data generated by today's digital activities. This makes it challenging to identify and correct forbidden itemsets in a timely manner.

2. **Domain-Specific Rules:** The rules defining forbidden itemsets are often specific to a particular domain and can change over time. Existing techniques may not be flexible enough to adapt to these changing rules .

3. **Lack of Transparency:** Many existing data cleaning methods can be seen as black boxes, providing little transparency or interpretability. This makes it difficult to understand how and why certain cleaning decisions were made, which is particularly problematic when the cleaned data is used for decision-making processes.

4. **Handling Complex Dependencies:** Existing techniques may struggle to handle complex dependencies between items. For instance, the fact that certain items are forbidden might depend on the presence or absence of other items, or on other contextual factors.

5. **Error Propagation:** Current methods may inadvertently introduce new errors during the data cleaning process, which can lead to further inaccuracies.

6. **Consistency Maintenance:** Existing techniques may fail to maintain consistency in the data after cleaning.

7. **Over-cleaning:** There's a risk of over-cleaning the data, which might lead to the removal of important outliers or rare cases that are actually valid.

8. **Resource Intensive:** Some existing techniques require significant computational resources, which may not be feasible for all organizations.

CHAPTER 3
SYSTEM ANALYSIS

3.1 Requirements Specification

The requirement specification outlines the functional and non-functional needs of the data cleaning system tailored for forbidden itemsets. Functionally, the system must efficiently detect forbidden itemsets and resolve violations while preserving data integrity. Non-functionally, it must meet performance, usability, security, and scalability criteria, ensuring efficient operation, user-friendly interface, data security, and adaptability to varying workload demands. The system's architecture, implementation plan, testing, and documentation strategies are also delineated, ensuring systematic development, validation, and user adoption. Through these specifications, the system aims to effectively address data inconsistencies related to forbidden itemsets, enhancing the reliability and integrity of datasets across domains.

3.1.1 Functional Requirements

Functional requirements for the data cleaning system with forbidden itemsets encompass the following aspects:

1. Forbidden Itemset Detection:
   - The system should be capable of identifying forbidden itemsets within the dataset based on predefined constraints.
   - It must employ efficient algorithms and techniques to detect forbidden itemsets, considering both single and composite itemsets.
   - Detection mechanisms should support various data types and structures commonly encountered in datasets.
2. Violation Resolution:
   - Upon detecting violations of forbidden itemset constraints, the system must provide mechanisms for resolving these inconsistencies.
   - It should offer options for correcting data instances that violate forbidden itemsets while maintaining data integrity and consistency.

3. Constraint Management:
    - The system should enable users to define, manage, and update forbidden itemset constraints dynamically.
    - It must support the specification of complex constraints, including constraints with multiple conditions and constraints applied across multiple attributes.
    - Constraint management functionalities should include validation, modification, and deletion of existing constraints.
4. Integration with Data Pipelines:
    - The system should seamlessly integrate with existing data pipelines or data processing workflows.
    - It must facilitate the incorporation of forbidden itemset detection and resolution processes into the data cleaning pipeline without disrupting data flow or causing delays.
    - Integration capabilities should accommodate various data sources, formats, and processing environments.
5. Feedback Mechanisms:
    - The system should provide feedback mechanisms to users regarding detected violations, resolution actions taken, and outcomes.
    - It must enable users to review and validate resolution decisions, with options for manual intervention when necessary.
    - Feedback mechanisms should support logging and tracking of resolution activities for auditing and monitoring purposes.
6. Performance Optimization:
    - The system should prioritize performance optimization to ensure efficient processing of large datasets and complex constraint rules.
    - It must employ algorithms and data structures that minimize computational overhead and memory usage during forbidden itemset detection and resolution.

By incorporating performance optimization strategies, the data cleaning system will deliver timely and reliable detection and resolution of forbidden itemset violations, even in high-volume and dynamic data environments.By fulfilling these functional requirements, the data cleaning system will effectively detect and resolve violations.

3.1.2 Non Functional Requirements

A Non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

1. Specify compatible operating systems for hosting the system.
2. Define required database management systems for data storage and management.
3. Identify programming languages and frameworks for system development.
4. List necessary data processing tools and libraries for algorithm implementation.
5. Include integration tools for seamless interaction with existing systems.
6. Specify security software for data encryption and access control measures.

- **User Requirements:**

  1. Users require an intuitive interface for configuring forbidden itemset constraints, facilitating easy management without extensive technical expertise.

  2. Efficient violation reporting is essential, allowing users to quickly identify and review violations, along with suggested resolution actions, for timely decision-making.

- **Software Requirements:**Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application. These requirements or prerequisites are generally not included in the software installation package and need to be installed separately before the software is installed.

  1) Software : Anaconda

  2) Primary Language : Python

  3) Frontend Framework : Flask

  4) Back-end Framework : Jupyter Notebook

- **Hardware Requirements:**

  1) Operating System : Windows Only

  2) Processor : i5 and above

  3) Ram : 8gb and above

  4) Hard Disk : 25 GB in local drive

3.2 UML Diagrams for the Project Work

UMLis an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques. It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words''. By using visual representations, we are able to better understand possible flaws or errors in software or business processes. The elements are like components which can be associated in different ways to make a complete UML picture, which is known as a diagram.

Thus, it is very important to understand the different diagrams to implement the knowledge in real- life systems. Any complex system is best understood by making some kind of diagrams or pictures. These diagrams have a better impact on our understanding. If we look around, we will realize that the diagrams are not a new concept but it is used widely in different forms in different industries. Mainly, UML has been used as a general-purpose modeling language in the field.of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UMLdiagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficiency. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system.

Since all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. Together this use case collection specifies all the ways the system. An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. Associations are the most general of all relationships and consequently the most semantically weak. If two objects are usually considered independently, the relationship is an association. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficiency. Use cases are best discovered by examining the actors and defining what the actor will be able to do with the system. Since

all the needs of a system typically cannot be covered in one use case, it is usual to have a collection of use cases. By default, the association tool on the toolbox is unidirectional and drawn on a diagram with a single arrow at one end of the association.Typically, on class diagrams, a dependency relationship indicates that the operations of the client invoke operations of the supplier. The workflow in this case begins from importing the dataset by the developer and then replacing missing values with mean value of corresponding column, model building, validating that model by generating a confusion matrix and finally predicting the test sample class label.

Transitions are used to show the passing of the flow of control from activity to activity.Unified Modeling Language (UML) diagrams serve as a standardized visual representation for modeling software systems, providing a common language for developers, designers, and stakeholders to communicate and understand system structure, behavior, and interactions. By utilizing UML diagrams, stakeholders gain insights into the system's architecture, design, and functionality, facilitating effective collaboration, requirement analysis, and system documentation throughout the software development lifecycle.UML diagrams offer a powerful toolset for both conceptualizing and documenting software systems, aiding in the visualization and analysis of complex system structures and behaviors. Additionally, UML diagrams serve as valuable artifacts for project documentation, serving as a reference for future maintenance, enhancements, and knowledge transfer within development teams.

The various UML diagrams are:

1. Use Case diagram
2. Activity diagram
3. Sequence diagram
4. Collaboration diagram
5. Object diagram
6. State chart diagram
7. Class diagram
8. Component diagram
9. Deployment diagram

3.2.1 System View Diagrams

Use Case Diagram

Use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and users and generalization among use cases. The use case model defines the outside (actors) and inside (use case) of the system's behavior. Actors are not part of the system. Actors represent anyone or anything that interacts with (input to or receive output from) the system. Use-case diagrams can be used during analysis to capture the system requirements and to understand how the system should work. During the design phase, you can use case diagrams to specify the behavior of the system as implemented. Use case is a sequence of transactions performed by a system that yields a measurable result of values for a particular actor.

The use cases are all the ways the system may be used. Use case diagrams serve as a graphical depiction of actors, use cases, and their relationships within a system. It encapsulates a set of use cases within a system boundary and illustrates communication associations between actors and use cases, as well as generalization relationships among use cases. Actors, distinct from the system itself, represent entities external to the system that interact with it, contributing input or receiving output. They encompass individuals, roles, or external systems that play a role in the system's functionality, serving as key stakeholders or users.

During the software development lifecycle, use-case diagrams fulfill various roles. In the analysis phase, they serve as a tool for capturing system requirements and comprehending the system's intended behavior. They provide a visual representation of how users interact with the system and outline the functionalities required to fulfill user needs. As the project progresses to the design phase, use case diagrams are employed to specify the system's behavior in its implemented state. They detail the sequence of transactions performed by the system to achieve specific outcomes for actors, delineating the various ways in which the system may be utilized. This helps in clarifying system functionality and guiding the design and development efforts towards meeting user expectations and business objectives. In essence, use case diagrams play a crucial role in

guiding system development by providing a clear understanding of user requirements and system behavior across different stages of the software development process.

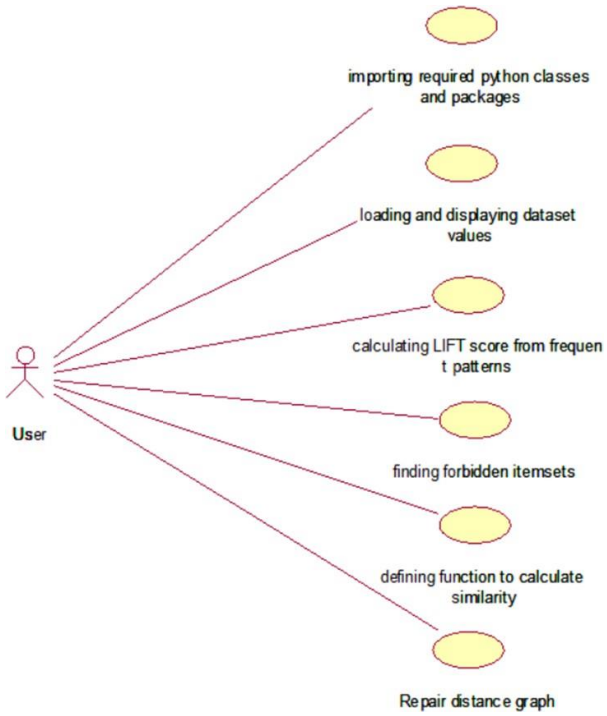

Fig 3.1: Usecase diagram

Activity Diagram

An Activity diagram serves as a graphical representation of workflows and processes within a system, focusing on the sequential execution of activities or operations. It is essentially a specialized form of a state machine, where states are replaced by activities representing the performance of specific operations. Transitions in an Activity diagram occur when activities are completed, indicating the progression from one state to another. This visual representation enables stakeholders to understand the flow of activities within a use case or across different classes, providing insights into how the system behaves under various scenarios.

Within an Activity diagram, activities represent the individual operations or tasks performed within a system. These activities are connected by transitions, which indicate the flow of control from one activity to the next based on the completion of preceding activities. Decision points allow for branching in the workflow, where different paths can be taken depending on certain conditions or criteria. Synchronization bars facilitate

coordination and synchronization between concurrent activities, ensuring proper sequencing and consistency in the system's behavior. The primary purpose of an Activity diagram is to depict the dynamic behavior and flow of activities within a system or process.

By visualizing these flows and interactions, stakeholders can gain a clearer understanding of how the system operates and how different components or classes interact with each other. Activity diagrams play a crucial role in system design, analysis, and communication, enabling stakeholders to identify potential bottlenecks, inefficiencies, or areas for optimization within the system's work flows.

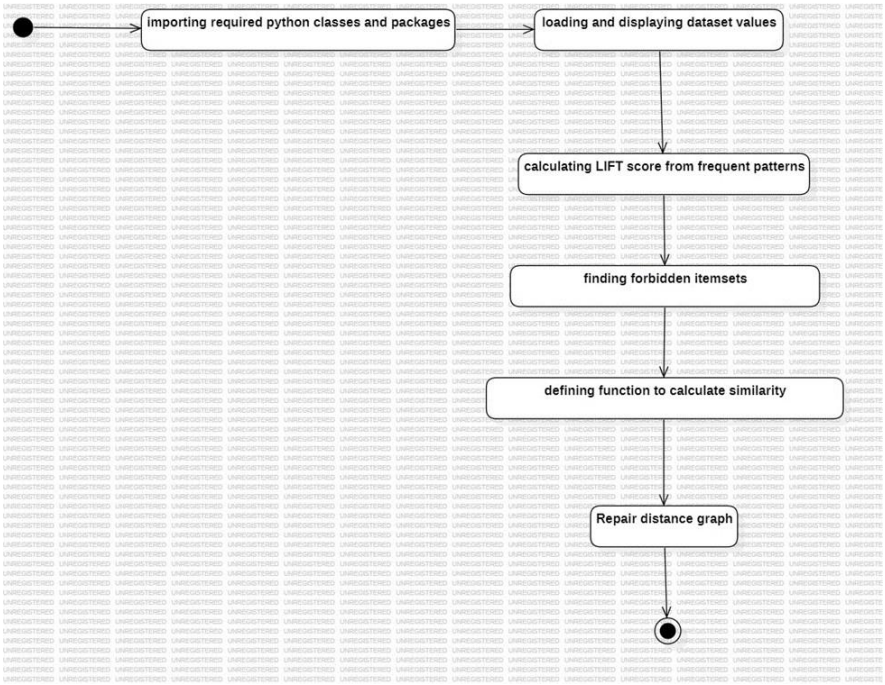

Fig 3.2: Activity Diagram

Sequence Diagram

Sequence diagram is a powerful tool in software engineering used to illustrate the interactions between different processes or objects within a system, showcasing the order in which these interactions occur. Derived from the Message Sequence Chart construct, a sequence diagram presents a visual representation of object interactions over time.

This chronological depiction allows stakeholders to comprehend the dynamic behavior of the system and understand how various components collaborate to accomplish specific functionalities. At its core, a sequence diagram captures the involvement of objects and classes in a given scenario, detailing the sequence of messages exchanged between them to fulfill the requirements of the scenario. Each object is represented as a vertical lifeline, and messages between objects are depicted as horizontal arrows, indicating the flow of communication.

By following the sequence of messages, stakeholders can track the flow of control and data throughout the system, gaining insights into the interactions between different components. Sequence diagrams are often utilized in the context of use case realizations within the Logical View of the system under development.They provide a tangible representation of how system functionalities are realized through the collaboration of various objects and classes. Through sequence diagrams, developers can effectively design, analyze, and communicate the dynamic aspects of their systems.By following the sequence of messages, stakeholders can track the flow of control and data throughout the system, gaining insights into the interactions between different components.

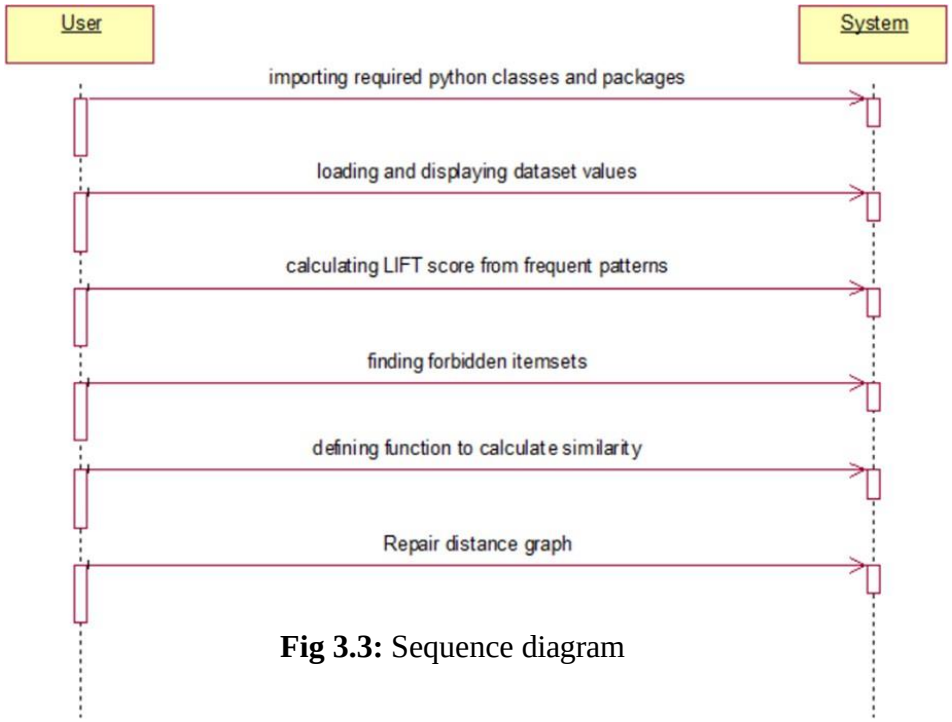

Fig 3.3: Sequence diagram

Collaboration Diagram

Collaboration diagram, also known as a communication diagram, is a graphical representation that illustrates the order of messages exchanged between objects to implement an operation or transaction within a system. Unlike sequence diagrams, which focus on the temporal sequence of messages, collaboration diagrams emphasize the structural relationships between objects and the interactions that occur among them. These diagrams visually depict objects as nodes and messages as links, highlighting the flow of communication between different components of the system.

In a collaboration diagram, objects are represented as nodes, and the links between them signify the messages exchanged during the execution of an operation or transaction. Additionally, collaboration diagrams may include simple class instances and class utility instances to further illustrate the interactions between objects.

By providing a visual representation of object interactions and message exchanges, collaboration diagrams offer stakeholders a clear understanding of how different components collaborate to achieve specific functionalities within the system.

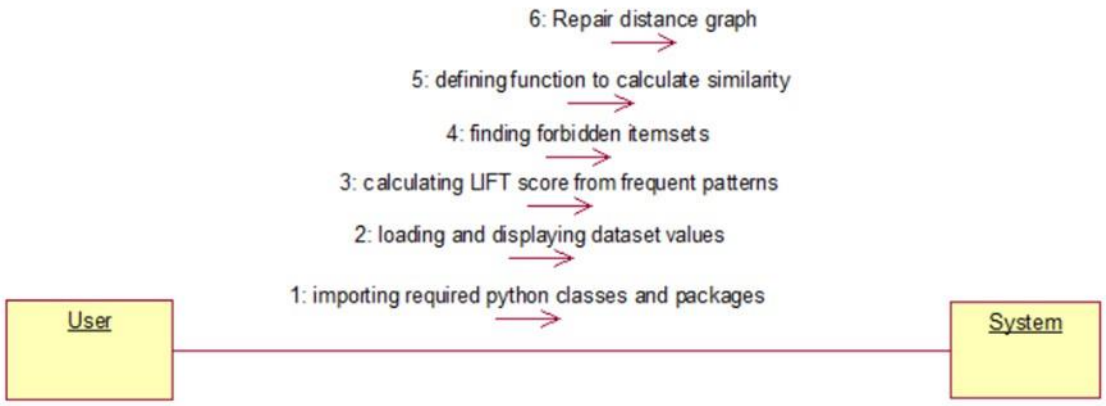

Fig 3.4: Collaboration diagram

Class Diagram

Class diagrams contain icons representing classes, interfaces, and their relationships. You can create one or more class diagrams to represent the classes at the top level of the current model; such class diagrams are themselves contained by the top level of the current model. You can also create one or more class diagrams to represent classes contained by each package in your model; such class diagrams are themselves contained by the package enclosing the classes they represent; the icons representing logical packages and classes in class diagrams. In the UML, classes are represented as compartmentalized rectangles.

1. The top compartment contains the name of the class.
2. The middle compartment contains the structure of the class (attributes).
3. The bottom compartment contains the behavior of the class (operations).

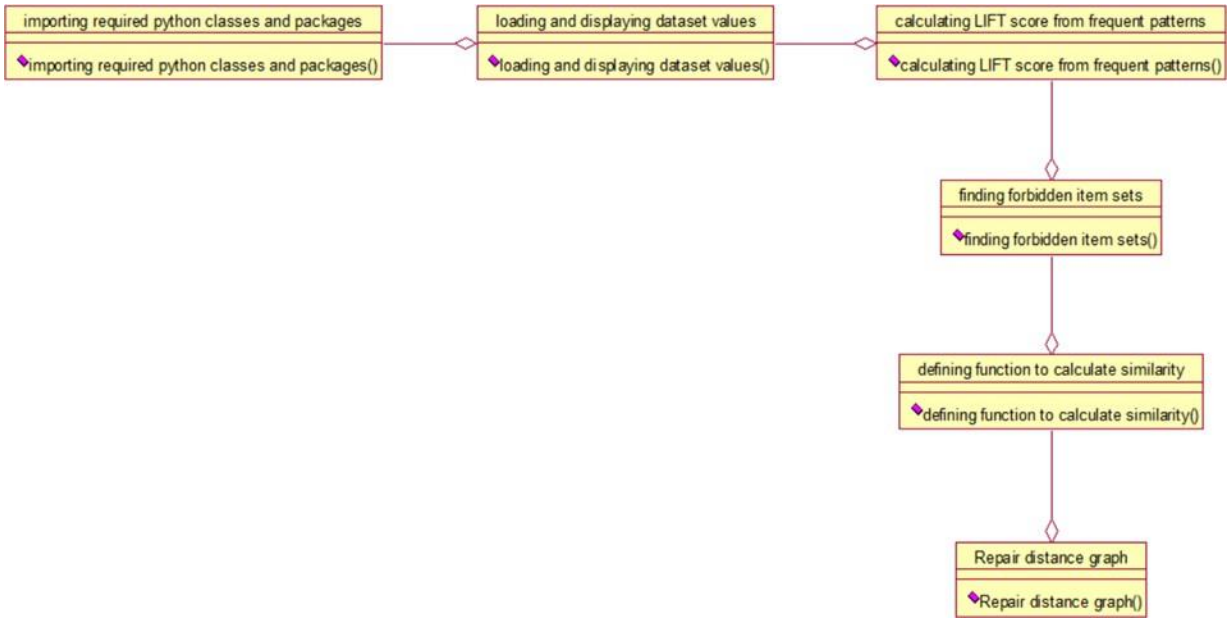

Fig 3.5: Class Diagram

State Chart Diagram

Use Cases and scenarios provide a valuable means of describing the behavior of a system by illustrating interactions between objects within the system. While these techniques primarily focus on the external behavior of the system, there are instances

where it becomes necessary to delve into the internal behavior of individual objects. In such cases,A state chart diagram proves to be a useful tool. This type of diagram provides a visual representation of the states of a single object, along with the events or messages that trigger transitions between these states and the actions associated with state changes. Similar to activity diagrams, state chart diagrams also feature special symbols representing start states and stop states, which denote the initial and final states, respectively, of the object's behavior.

These diagrams serve to capture the dynamic behavior of objects by illustrating how they transition between different states in response to external stimuli or events. However, it's important to note that state chart diagrams are not applicable to every class in the system. They are typically created only for those class objects that exhibit significant behavioral complexity or state-dependent functionality.State chart diagrams offer a structured and systematic approach to modeling the behavior of individual objects within a system. By visually mapping out the states, transitions, and actions associated with state changes, these diagrams provide insights into the internal workings of objects and help stakeholders understand their behavioral dynamics.

Fig 3.6: State Chart Diagram

3.2.2 Detailed View Diagrams

Component Diagram

Component diagrams play a crucial role in software engineering by illustrating the dependencies between various software components within a system. These dependencies can vary depending on the programming languages used for development and may manifest at different stages such as compile-time, link-time, or run-time. In a large-scale project, the system is typically composed of numerous files, each representing a software component.

These files often exhibit dependencies on one another, reflecting the relationships and interactions between different parts of the system. The nature of dependencies depicted in component diagrams can vary based on the language or languages employed in the development process. At compile-time, dependencies between source code files may arise due to the inclusion of headers, libraries, or modules required for compilation.

At link-time, dependencies may involve the linking of object files or libraries to generate executable files or bytecode files. Additionally, dependencies may also exist at run-time, where components interact dynamically during the execution of the software system. Component diagrams provide a visual representation of these dependencies, aiding developers in understanding and managing the complex relationships within the system. In the Unified Modeling Language (UML), component diagrams are categorized as one of the two types of implementation diagrams, the other being deployment diagrams.

While deployment diagrams focus on the physical deployment of software components onto hardware nodes, component diagrams delve into the logical organization and relationships between software components. By depicting the dependencies between components and illustrating how they interact and communicate with each other, component diagrams assist in the design, analysis, and documentation of software systems, ultimately facilitating efficient development and maintenance processes.The nature of dependencies depicted in component diagrams can vary based on the language or languages employed in the development process.

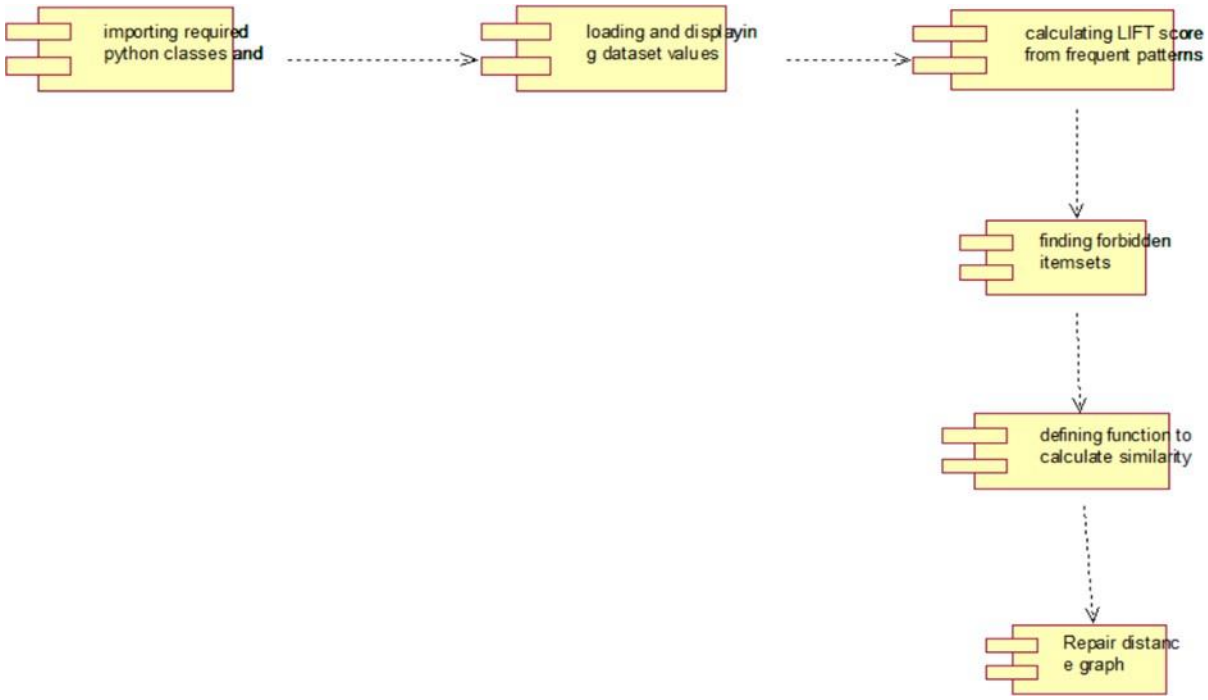

Fig 3.7 Component Diagram

Object Diagram

An Object diagram, a type of UML structural diagram, offers a detailed snapshot of specific instances of classes and their relationships within a system at a particular moment. It provides a visual representation of the runtime state of a system, presenting concrete objects along with their attributes and associations. Each object, depicted as a rectangle, showcases its name atop and lists its attributes and values underneath. Links between objects illustrate how they are related, with labels or multiplicity indicators clarifying the nature and cardinality of these relationships.

These diagrams serve several purposes, notably aiding in the verification of system behavior during implementation and testing phases. By offering insights into the current state of objects and their interactions, object diagrams facilitate debugging and troubleshooting processes. Through their depiction of actual instances and their relationships, they provide a tangible understanding of the system's runtime dynamics, enabling developers to validate the correctness of the system's design and behavior.

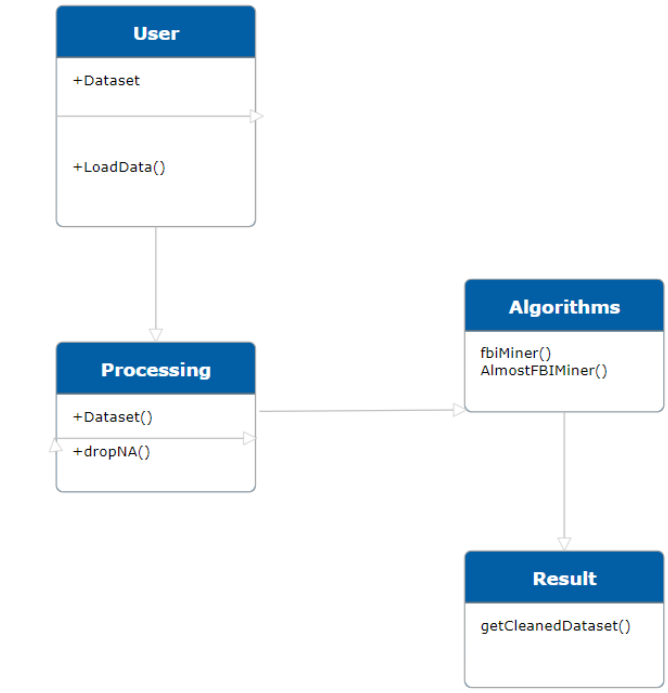

Fig 3.8 Object Diagram

Deployment Diagram

Deployment diagrams serve as a powerful tool in software engineering for visualizing the configuration of run-time processing elements within a system, along with the software components and processes residing on them. These diagrams provide a comprehensive overview of the physical infrastructure and the distribution of software components across different nodes. In a deployment diagram, nodes represent the various processing elements, such as computers, servers, or other hardware devices, while communication associations depict the network connections and protocols used for communication between these nodes.

Nodes in deployment diagrams are not limited to representing computers or servers; they can also depict other processing resources, such as people or mechanical devices involved in the system. Each node is typically depicted as a 3D view of a cube or rectangular prism, providing a visual representation of the physical entities in the system architecture. Communication associations between nodes illustrate the network connections and protocols used for communication, enabling stakeholders to understand how data flows between different components within the system. The simplicity of

deployment diagrams lies in their ability to convey complex system architectures in a clear and intuitive manner. By visualizing the deployment of software components onto physical nodes and illustrating the communication pathways between these nodes, deployment diagrams aid in system design, analysis, and communication among stakeholders.

These diagrams serve as valuable documentation artifacts, providing insights into the runtime configuration of the system and facilitating efficient management and maintenance of software applications.A deployment diagram in the context of your project, "cleaning data with forbidden itemsets," would depict the physical deployment of software components across various nodes or hardware devices.

Fig 3.9 Deployment Diagram

CHAPTER 4

SYSTEM DESIGN

4.1 Architecture of the Proposed System

The architecture of the Contextual User Preference Attention Model can be summarized as follows:

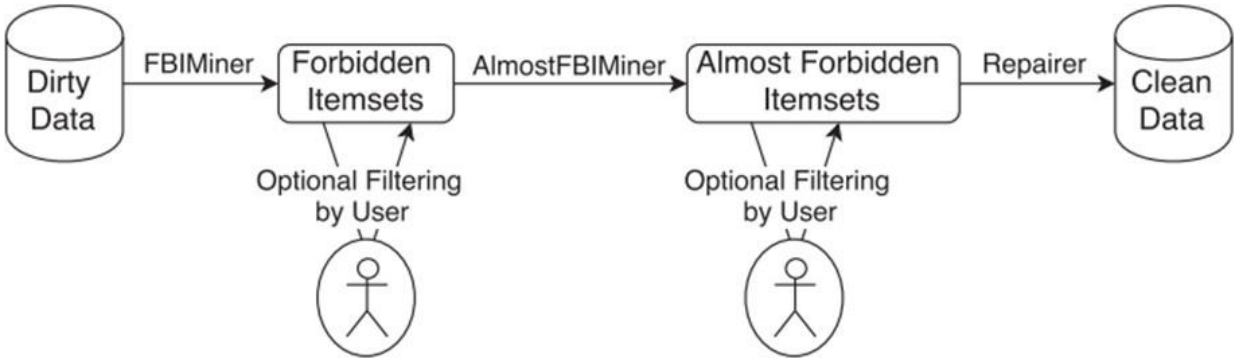

Fig 4.1: System Architecture

Forbidden itemsets are a powerful concept used to identify and rectify inconsistencies in datasets. They represent combinations of attribute values that should not co-occur in any valid record.

1. Input: Dirty Data

A dataset that possibly contains inconsistencies, errors, or anomalies is considered initially. This dataset is referred to as the "dirty data".

2. FBIMiner: Identifying Forbidden Itemsets

FBIMiner is a hypothetical algorithm/tool designed to mine or discover forbidden itemsets from this dirty data.

3. **Optional Filtering in FBIMiner**
   - Once `FBIMiner` comes up with potential forbidden itemsets, not all of them might be relevant or true positives.

4. **AlmostFBIMiner: Identifying Near-Forbidden Patterns**
   - `AlmostFBIMiner` delves into patterns that are close to being forbidden but aren't strictly so. It identifies infrequent or suspicious combinations which might or might not be errors.
   - These are patterns that might be on the brink of becoming errors or might be rare yet valid occurrences.

5. **Optional Filtering in AlmostFBIMiner**
   - Given the ambiguous nature of the patterns identified by `AlmostFBIMiner`, user discretion becomes even more critical.
   - The user can apply optional filters to decide which of the detected patterns should be treated as errors or inconsistencies. This ensures that valid but rare data points aren't erroneously flagged or modified.

6. **Repairing Algorithm**

Once you have a finalized list of forbidden itemsets (from both `FBIMiner` and filtered results of `AlmostFBIMiner`), the next step is to rectify the records containing these problematic patterns.This could involve:

1. Modifying an erroneous value to a valid one.
2. Deleting records if they're deemed irreparable.
3. Flagging them for manual inspection if the algorithm cannot confident

7. **Output: Clean Data**

After the repairing algorithm has done its job, the resulting dataset should be free of the identified forbidden itemsets. This "cleaned" dataset is ready for further processing or analysis.

4.2 Workflow of the Proposed System

Step 1: importing Required python classes and packages.

Step 2: Loading and Displaying dataset values.

Step 3: Selecting Relevant Columns to perform Frequent Patterns

Step 4: Generate Frequent Patterns for the given dataset.

Step 5: Calculating LIFT score from frequent Patterns.

Step 6: Finding Forbidden Itemsets based on the lift values.

Step 7: Defining Function to calculator similarity.

Step 8: Plot Repair Distance Graph.

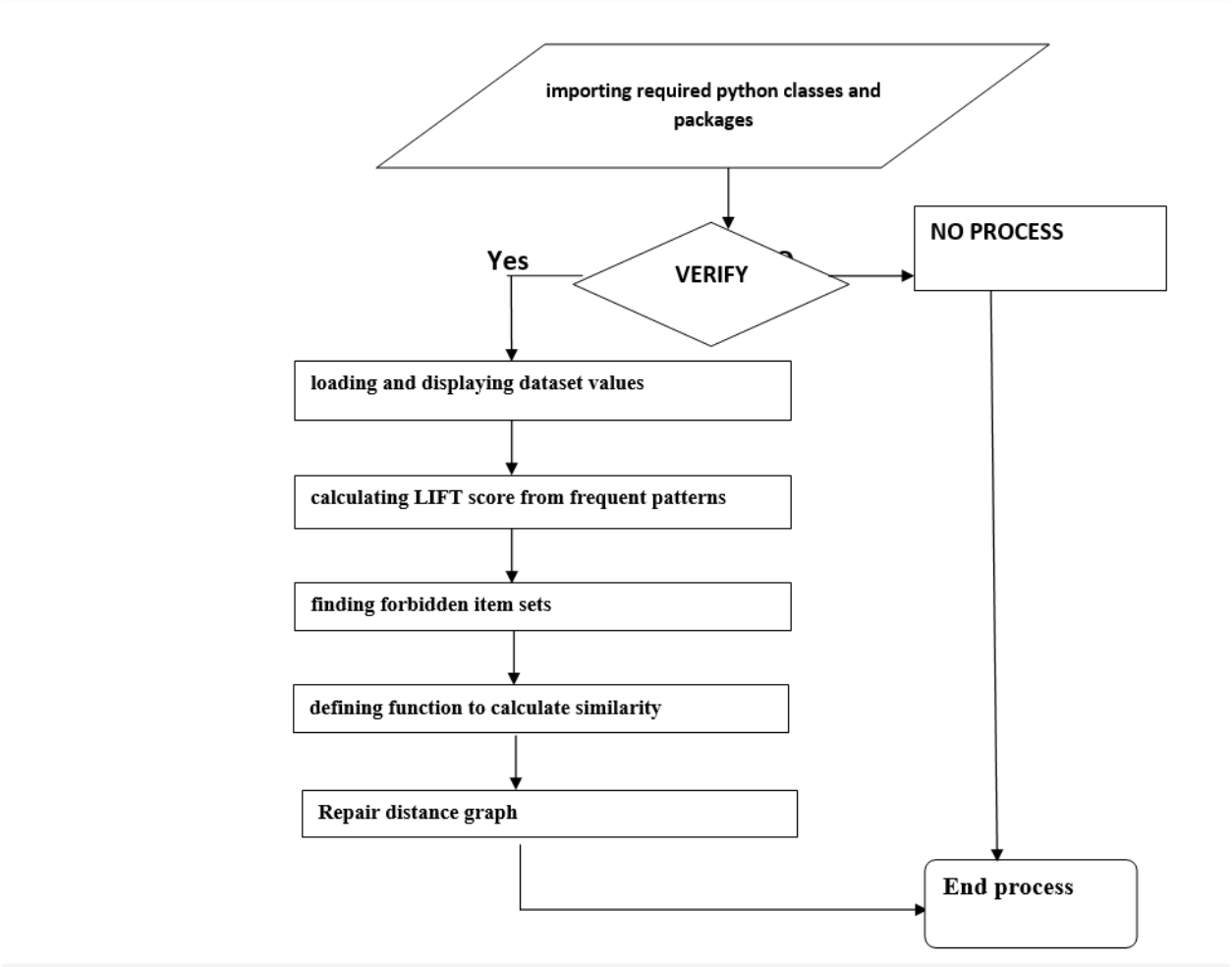

Fig 4.2: DataFlow Diagram

4.3 Modules Description

Forbidden ItemSets:

Age	Marital Status	Relation	Sex	Income
31-50	Married	Husband	Female	<50K
<18	Married	Husband	Male	<50K

Table 4.1: Example data from UCI Adult Dataset

Certainly, here's a brief summary of the provided content:

The UCI Adult Census dataset was analyzed to identify suspicious value co-occurrences termed as "forbidden itemsets." Two examples were given:

- Husband and Female: Though there are many records labeled "husband" and many labeled "female," very few contain both.
- Age < 18 and Married: There are many records with individuals under 18 and many with "married" status, but barely any record contains both.

The peculiarity of these co-occurrences is quantified using the "lift" measure, which will be defined later.The core challenge arises when attempting to repair these inconsistencies:

Repairing a "husband-female" inconsistency might lead to a new forbidden itemset. For example, changing the relation from "Husband" to "Unmarried" while keeping the marital status as "Married" creates another contradiction.Similarly, correcting an age inconsistency could impact other attributes. Adjusting someone's age from "< 18" to "18-21" might end up making their income status seem peculiar for their age group.

Thus, every attempt at correcting a forbidden itemset might lead to the creation of a new one. With countless possible modifications, it's impractical to rediscover forbidden itemsets after every change to ensure no new contradictions are introduced.so,we have to deal with new errors,

Inconsistencies.To overcome this, the approach identifies "almost forbidden itemsets" in advance. These are itemsets that could potentially become forbidden with slight data changes. By having this pre-computed set, the repair process can quickly check against it, ensuring modifications don't inadvertently introduce new inconsistencies.In essence, the cleaning mechanism ensures that while correcting existing data issues, it doesn't unintentionally introduce new ones by referencing a list of potentially problematic itemsets.

Forbidden Itemsets	**Dataset**	**Threshold Value**
Sex=Female,Relation=Husband Sex=Male, Relation=Wife Age=<18,Marital-status=Married Age=<18,Relationship=Husband Relation=Not-in-family,Marital=Married	Adult	0.01
aquatic=0,breathes=0(clam) type=mammal,eggs=1(platypus) milk=1,eggs=1(platypus) type=mammal,toothed=0(platypus) milk=1,toothed=0(platypus) eggs=0,toothed=0(scorpion) tail=1,backbone=0(scorpion)	Zoo	0.1
bruises=t,habitat=l population=y,cap-shape=k cap-surface=s,odor=n,habitat=d cap-surfaces=s,gill-size=b,habitat=d edible=e,habitat=d,cap-shape=k	Mushroom	0.025

Table 4.2: list of forbidden itemsets

The above is the list of forbidden itemsets generated in the Adult,Zoo and Mushroom datasets.Forbidden itemsets can identify a significant number of errors, many of which are also detected by other formal methods.

Intuition Behind Forbidden Itemsets:

Forbidden itemsets and their generation can be linked to association rule mining, where metrics like support, confidence, and lift are frequently used to evaluate the significance and strength of itemsets or rules. To explain how forbidden itemsets are generated using the lift measure, we first need to understand the basics of lift.

- Lift Measure:

  Lift is a measure used in association rule mining to signify the strength of a rule. For two items A and B, the formula is:The lift value has the following interpretations:

  Lift = 1: A and B are independent.

  Lift > 1: A and B are positively correlated (i.e., occurrence of A increases the likelihood of occurrence of B more than we'd expect by chance).

  Lift < 1: A and B are negatively correlated (i.e., occurrence of A reduces the likelihood of occurrence of B).

Generating Forbidden Itemsets Using Lift

1. Data Mining: First, mine the dataset for all potential itemsets (combinations of values across attributes).

2. Calculate Lift for Itemsets: For every identified itemset, calculate the lift value to understand the association strength between the items in the itemset.

3. Filter by Threshold:

- Define a threshold value for lift, below which itemsets are considered "forbidden" .

- If the lift value is considerably less than 1 (based on the threshold), it suggests that the items in the itemset rarely (or never) occur together, and their combination could be anomalous or forbidden.

4. Validation with Domain Knowledge: After identifying potential forbidden itemsets based on lift, it's essential to validate them using domain knowledge. Some rare combinations might be legitimate, while others might indeed be errors.

5. Iterative Refinement: Continuously refine the list of forbidden itemsets based on feedback from domain experts, downstream data processing tasks, or as new data becomes available.

Discovering Almost Forbidden Itemsets:

"Almost forbidden itemsets" refer to combinations of attribute values within a dataset that are near, but not strictly, forbidden. These are patterns or combinations that are infrequent or appear suspicious within the data, suggesting that they might be anomalies or errors, but they haven't been explicitly defined as invalid or forbidden.

The content revolves around the discovery of "almost forbidden itemsets" using a specific algorithm, A-FBIMINER. Here's a simplified explanation:

1. Objective:The aim is to discover "almost forbidden itemsets" from data, which are combinations of values that are close to being forbidden but aren't strictly so.
2. Minimal Possible Lift Measure:Before mining these itemsets, there's a need to define a "minimal possible lift measure". Lift is a metric that indicates the strength of a rule in association rule mining. In this context, a lower lift value for an itemset would indicate its rarity or unusualness.
3. A-FIB MINER Algorithm:A-FBI MINER is the proposed algorithm to mine almost forbidden itemsets. While it bears similarities to FBIMINER (presumably an algorithm to mine strictly forbidden itemsets), it uses the new minimal possible lift measure.
4. Challenges with Pruning:One of the benefits of using algorithms in data mining is the ability to prune, or eliminate, unnecessary paths early, optimizing runtime. If you were to repair or clean data entries one at a time, strong pruning is possible, but the algorithm would need to be run repeatedly, which is inefficient.
5. Batch Processing:To strike a balance between efficiency and effectiveness, the content suggests repairing "dirty objects" (data entries with errors) in batches rather than one-by-one or all at once.
6. Prior Publications:More details about the pruning properties, specifics of the A-FBI MINER algorithm, and discussions on batch sizes have been previously published in another source.

In essence, the goal is to efficiently discover almost forbidden itemsets in data, considering the trade-offs between algorithm runtime and accuracy.

Repairing Algorithm:

The repairing algorithm, when using forbidden itemsets, is a systematic approach to ensuring data quality. By identifying and rectifying invalid combinations of data values, it maintains the consistency and reliability of the dataset.

Steps involved in repairing algorithm:

1. Identification: Traverse the dataset and identify records that match the criteria for forbidden itemsets. These represent combinations of values or attributes that shouldn't coexist based on domain knowledge or data analysis.

2. Determine Repair Action: Depending on the nature of the forbidden itemset, determine the best course of action:

- Modify an erroneous value to a more probable one.
- Delete a certain value or the entire record.
- Flag the record for manual inspection or review.

3. Execute Repair: Implement the determined action on the identified forbidden itemsets. This rectification process should adhere to predefined rules or use machine learning models trained to correct such inconsistencies.

4. Validation: After a repair action, re-scan the record to ensure that the repair hasn't introduced a new forbidden itemset. Ensure that the data's integrity remains intact post-repair.

5. Log and Document: Maintain a comprehensive log detailing all repairs, including the nature of the forbidden itemset, the action taken, and any other relevant metadata. This documentation is critical for transparency, traceability, and future refinements.

6. Manual Intervention (if necessary): Some forbidden itemsets might be complex or ambiguous. In such cases, instead of an automatic repair, the algorithm could flag them for expert review. A domain expert can then decide on the best course of action.

7. Iterative Repairing: The cleaning process might be iterative. Once initial repairs are made, the dataset is re-evaluated for forbidden itemsets, ensuring that no new inconsistencies were introduced.

8. Feedback Mechanism: Incorporate feedback from the manual reviews and other data processing stages to refine and optimize the repairing algorithm over time.

CHAPTER 5
IMPLEMENTATION

5.1 Algorithms

1. Apriori algorithms for Frequent Itemsets
2. Association Rule Mining
3. Similarity Based Repair

Certainly! Let's delve into each algorithm employed in the implementation of the data cleaning system with forbidden itemsets:

5.1.1 Apriori Algorithm for Frequent Itemsets:

The Apriori algorithm is a classic algorithm used for mining frequent itemsets in transactional databases. It operates based on the principle of "apriori property," which states that if an itemset is frequent, then all of its subsets must also be frequent. The algorithm works in two main steps: candidate generation and frequent itemset generation. In the candidate generation step, candidate itemsets of length k are generated from frequent itemsets of length k-1. Then, in the frequent itemset generation step, the support of each candidate itemset is calculated, and those that meet the minimum support threshold are retained as frequent itemsets.

In the context of the data cleaning system, the Apriori algorithm plays a crucial role in identifying frequent item sets from the dataset. By setting a minimum support threshold, the algorithm efficiently discovers patterns of data values that occur frequently together. These frequent itemsets serve as the foundation for further analysis, including the detection of forbidden itemsets. By mining frequent itemsets, the algorithm provides valuable insights into the associations and dependencies present in the dataset, which are essential for understanding data patterns and constraints.One of the key advantages of the Apriori algorithm is its scalability and efficiency in handling large transactional datasets. It employs a level-wise approach to candidate generation.

5.1.2 Association Rule Mining:

Association rule mining is a data mining technique used to discover interesting relationships, or associations, between variables in large datasets. It aims to uncover rules of the form "if X, then Y," where X and Y are sets of items. The strength of association between X and Y is quantified using metrics such as support, confidence, and lift. Support measures the frequency of occurrence of the rule in the dataset, confidence measures the conditional probability of Y given X, and lift measures the degree of association between X and Y beyond what would be expected by chance.

In the context of the data cleaning system, association rule mining is employed to generate rules that capture relationships between different data values. These rules are derived from frequent itemsets identified by the Apriori algorithm.
By setting appropriate thresholds for support and confidence, the algorithm filters out spurious associations and retains only meaningful rules. These association rules serve as a basis for detecting violations of predefined constraints, such as forbidden itemsets, within the dataset.

Association rule mining offers several advantages for data cleaning and analysis. It provides a systematic and automated approach to uncovering interesting patterns and relationships in the data, which may not be immediately apparent through manual inspection. Additionally, association rules offer actionable insights that can inform decision-making processes, such as identifying data quality issues or informing data governance policies.

By leveraging association rule mining, the data cleaning system can effectively identify and address violations of constraints, thereby improving the quality and reliability of the dataset for downstream analysis or use.In addition to identifying association rules, association rule mining also plays a crucial role in quantifying the strength of the discovered relationships. Metrics such as confidence and lift provide valuable insights into the significance and reliability of the association rules.

5.1.3 Similarity-Based Repair:

Similarity-based repair is a technique used to address violations of constraints, such as forbidden itemsets, within the dataset by finding suitable replacements for dirty itemsets. It relies on measuring the similarity between dirty itemsets and a predefined cluster of valid itemsets to identify the most appropriate replacements. Common similarity measures include the Levenshtein distance algorithm, which computes the minimum number of edits required to transform one string into another, and cosine similarity, which measures the cosine of the angle between two vectors representing the itemsets.

In the data cleaning system, similarity-based repair complements association rule mining by providing a mechanism to rectify violations detected in the dataset. By computing the similarity between dirty itemsets and a cluster of valid itemsets, the algorithm selects replacements that are most similar to the original values while adhering to the predefined constraints. This ensures that the repaired dataset maintains data integrity and compliance with constraints, thereby improving its quality and reliability for downstream analysis or application.

One of the key advantages of similarity-based repair is its flexibility and adaptability to different types of data and constraints. The algorithm can be customized to use various similarity measures and clustering techniques based on the specific characteristics of the dataset and the nature of the constraints. Additionally, similarity-based repair offers a systematic and automated approach to repairing violations, reducing the need for manual intervention and streamlining the data cleaning process. Overall, similarity-based repair enhances the effectiveness of the data cleaning system by providing a robust mechanism for addressing violations of constraints and ensuring dataset quality.

Furthermore, similarity-based repair offers a robust approach to rectifying violations of constraints by leveraging the concept of similarity between data values. By computing the similarity between dirty itemsets and a predefined cluster of valid itemsets.

5.2 Datasets

UCI Adult Census Dataset:

This dataset has been used to illustrate the concept of forbidden itemsets. The dataset contains demographic information which can be used to predict if an individual earns more than $50,000/year. Suspicious value co-occurrences, like the combination of "Husband" and "Female", can be treated as forbidden itemsets.The Adult dataset contains demographic data with multiple attributes.It is often used in the machine learning community for experiments due to its easily interpretable nature.The Adult dataset was referenced in the context of the experiments. It served as one of the primary datasets on which forbidden itemsets were evaluated.The research seemed to focus on determining how suitable t-forbidden itemsets are for discovering inconsistencies. This dataset was employed to assess the precision of the discovered forbidden itemsets and their effectiveness in error detection.The text also implies that this dataset was used to evaluate runtime performance of their repair algorithm, comparing sequential and parallel implementations.

Soccer Dataset:

The Soccer dataset seemed to play a crucial role in the experimentation phase of the research.Artificial errors were introduced into this dataset using the BART error generator. The goal was to create a "dirty" version of the dataset to test the effectiveness of the forbidden itemset method in identifying and repairing inconsistencies.Comparisons were made between forbidden itemsets and other error detection methods, such as Constant CFDs and Constant Denial Constraints. This dataset helped in understanding the precision and recall of the proposed method in detecting dirty objects.It also helped in analyzing the effectiveness of the repair algorithm. After the dataset was "dirtied," the repair methods were used to bring the data closer to the original "ground truth" of the dataset.The impact of different repair schemes (Single Linkage, Complete Linkage, and Mean Linkage) was assessed on this dataset to determine which scheme was the most robust.

5.3 Metrics Calculated

1. Support (supp):
   - Formula: $|D \cap I| / |D|$
   - Description: Measures the frequency of occurrence of an itemset I in the dataset D.

2. Lift (lift):
   - Formula: $lift(I;D) = |D| \times supp(I;D) / (\min_{J \subset I}(supp(J;D) \times (supp(I \cap J;D))))$
   - Description: Measures the discrepancy between the actual support of co-occurrence of itemsets and the expected support if they were statistically independent. High lift values indicate stronger associations.

3. Almost t -Forbidden Itemsets:
   - Formula: $Lift_k (I;D) <= t$
   - Description: Identifies itemsets that could become t -forbidden after applying at most k modifications to the dataset.

4. Minimal Possible Lift After k Modifications ($mplift_k$):
   - Formula:

$$mplift_k (I;J;D) = \min_{(DI,DJ,D \cap J)} (|D| \times (supp(I;D)+DI)/(supp(J;D)+DJ) \times (supp(I \cap J;D)+D \cap J)))$$

   - Description: Estimates the minimum possible lift of itemsets after k modifications to the dataset.

5. Forbidden Itemsets (FBI):
   - Description: Sets of itemsets that are not allowed to co-occur in the dataset due to domain-specific constraints.

These metrics help in understanding the relationships between itemsets in the dataset and identifying potentially problematic associations that violate certain constraints or rules.

5.4 Methods Compared

5.4.1. Traditional Data Cleaning Methods:

Traditional data cleaning methods encompass a range of techniques commonly used to preprocess and clean datasets. These methods often include rule-based cleaning, which involves applying predefined rules or heuristics to identify and correct errors in the data. For example, in rule-based cleaning, specific conditions may be defined to flag outliers, missing values, or inconsistencies based on domain knowledge or predefined thresholds. Additionally, traditional methods may involve techniques such as outlier detection algorithms or imputation methods to address missing or erroneous data points. By comparing your forbidden itemset-based approach with traditional data cleaning methods, you can evaluate its effectiveness in terms of accuracy, efficiency, and scalability.

Traditional methods may excel in scenarios where rules can be easily defined and applied, but they may lack the adaptability and flexibility needed to handle complex data inconsistencies effectively.Traditional data cleaning methods encompass a variety of techniques and strategies aimed at identifying and rectifying errors, inconsistencies, and inaccuracies within datasets. These methods typically rely on manual inspection, domain expertise, and rule-based heuristics to detect and correct data anomalies. One common approach involves the use of data profiling techniques to assess the quality and completeness of the dataset, identifying potential issues such as missing values, outliers, or inconsistencies in data formats.

Data profiling tools generate summary statistics and visualizations to provide insights into the distribution and characteristics of the data, guiding subsequent cleaning efforts.Another traditional data cleaning method involves the application of rule-based validation and correction procedures to enforce data integrity constraints. These rules are typically defined based on domain knowledge and business requirements, specifying conditions that must hold true for the dataset to be considered valid. For example, a rule may dictate that certain attributes should fall within predefined ranges or that specific combinations of values are not allowed.

5.4.2. Association Rule Mining Algorithms:

Association rule mining algorithms, such as Apriori or FP-growth, are commonly used for discovering patterns in transactional datasets, particularly in market basket analysis. These algorithms identify frequent itemsets and association rules to reveal relationships between items in the dataset. While association rule mining algorithms are adept at discovering frequent itemsets and patterns, they may not explicitly capture forbidden itemsets or constraints.

Your approach offers a complementary perspective by focusing specifically on forbidden itemsets, which can provide valuable insights into unexpected data co-occurrences and help enforce data integrity constraints effectively.

Evaluating the performance of your approach against association rule mining algorithms will help determine its suitability for data cleaning tasks and its potential advantages over existing techniques.Certainly! When comparing your forbidden itemset-based approach with association rule mining algorithms, it's important to consider the specific focus and capabilities of each method.

Association rule mining algorithms, such as Apriori or FP-growth, excel at identifying frequent itemsets and association rules within transactional datasets.Association rule mining algorithms, such as Apriori and FP-growth, are widely used in data mining and machine learning for discovering interesting patterns, relationships, and associations within transactional datasets. These algorithms operate on the principle of identifying frequent itemsets, which are subsets of items that frequently occur together in transactions. By analyzing the co-occurrence of items, association rule mining algorithms can extract meaningful insights about the underlying patterns and dependencies within the dataset. One of the key advantages of association rule mining algorithms is their ability to uncover both frequent itemsets and the association rules derived from them, which provide valuable information about the relationships between different items.

The Apriori algorithm is one of the most well-known association rule mining algorithms, which employs a breadth-first search strategy to discover frequent itemsets by iteratively generating candidate itemsets.

5.4.3. Constraint-Based Data Cleaning:

Constraint-based data cleaning approaches, such as constant functional dependencies (CFDs) or denial constraints, enforce conditions that must hold true over a dataset. These constraints define logical rules or patterns that govern the data's integrity, such as unique keys, referential integrity, or value dependencies. By comparing your forbidden itemset-based approach with constraint-based methods, you can evaluate its ability to complement or enhance existing techniques in terms of accuracy and coverage. While constraint-based methods offer a systematic way to enforce data integrity constraints, they may struggle to capture complex dependencies or unexpected patterns in the data.

Your forbidden itemset approach provides a complementary perspective by focusing on unlikely co-occurrences of values, which can help identify subtle inconsistencies that traditional constraints may overlook. By comparing the performance of your approach with constraint-based methods, you can assess its effectiveness in detecting and repairing data inconsistencies across various scenarios and datasets.Constraint-based data cleaning is a methodology that leverages predefined integrity constraints to detect and rectify errors, inconsistencies, and anomalies within datasets.

These constraints, which are typically specified based on domain knowledge and business rules, define conditions that the data must satisfy to be considered valid. Constraint-based data cleaning approaches operate by evaluating the dataset against these constraints and identifying violations that indicate potential errors or inconsistencies. Once violations are detected, appropriate actions are taken to rectify them, such as imputing missing values, correcting erroneous entries, or removing outliers. By enforcing integrity constraints, constraint-based data cleaning methods ensure that the dataset adheres to predefined rules, thereby enhancing data quality and reliability.

One common type of constraint used in constraint-based data cleaning is the functional dependency (FD), which specifies a relationship between two sets of attributes in the dataset.Functional dependencies express dependencies or associations between attributes, indicating that the values of certain attributes are uniquely determined by the values of others. For example, a functional dependency may specify that the combination of a customer's ID and their email address uniquely determines their contact information. By enforcing functional dependencies, constraint-based data cleaning methods can identify inconsistencies or violations that occur when the specified dependencies do not hold true in the dataset.Another type of constraint commonly used in constraint-based data cleaning is the denial constraint (DC), which expresses conditions that cannot hold simultaneously over the dataset.

Denial constraints specify combinations of attribute values that are considered invalid or prohibited, indicating scenarios that should not occur in the dataset. For example, a denial constraint may specify that a customer cannot have both an active and inactive status at the same time. By detecting violations of denial constraints, constraint-based data cleaning methods can identify erroneous or inconsistent data entries that need to be corrected or removed to ensure data integrity.Constraint-based data cleaning approaches offer several advantages over other methods, including transparency, interpretability, and the ability to incorporate domain knowledge and business rules into the cleaning process.

By explicitly defining integrity constraints, these methods provide a clear framework for assessing data quality and identifying errors or inconsistencies. Additionally, constraint-based data cleaning methods can be customized to enforce specific constraints relevant to the application domain, allowing for flexible and adaptable cleaning strategies. Despite these advantages, constraint-based data cleaning methods may face challenges when dealing with complex datasets or when the specified constraints are too restrictive, leading to potential trade-offs between data integrity and completeness.

5.4.4. Machine Learning-Based Approaches:

Machine learning-based approaches for data cleaning leverage algorithms and models to automatically detect and correct errors in the data. These approaches may include supervised learning methods, such as classification or regression, or unsupervised learning techniques, such as clustering for anomaly detection. By comparing your forbidden itemset-based approach with machine learning methods, you can evaluate its performance in terms of accuracy, efficiency, and scalability. While machine learning approaches offer the advantage of automated error detection and repair, they may require labeled training data or extensive computational resources to train complex models. Your forbidden itemset approach provides a complementary perspective by focusing on explicit constraints and patterns in the data, which can help guide the cleaning process and ensure data integrity.

CHAPTER 6
TESTING

6.1 Introduction to Testing

Testing is a fault detection technique that tries to create failure and erroneous states in a planned way. This allows the developer to detect failures in the system before it is released to the customer. Note that this definition of testing implies that a successful test is a test that identifies faults. We will use this definition throughout the definition phase. Another often used definition of testing is that it demonstrates that faults are not present.Testing is a fundamental aspect of software development, encompassing various techniques and methodologies aimed at validating the functionality, correctness, and reliability of software systems.

It involves systematically evaluating the system's behavior against specified requirements, identifying defects or discrepancies, and ensuring that the software meets the intended criteria for quality and performance. Testing serves as a critical quality assurance mechanism throughout the software development lifecycle, helping to uncover issues early in the process and facilitating their timely resolution. By rigorously testing software systems, developers can ensure their reliability, robustness, and effectiveness in meeting user needs and expectations.

Testing can be done in two ways:

1. Top-down approach: This type of testing starts from upper-level modules. Since the detailed activities usually performed in the lower-level routines are not provided, stubs are written.
2. Bottom-up Approach: Testing can be performed starting from smallest and lowest level modules and proceeding one at a time. In this project, the bottom-up approach is used where the lower-level modules are tested first and the next ones having much data in them.

6.2 Testing Strategies

Testing strategies for a project involving cleaning data using forbidden item sets are crucial to ensure the accuracy of identifying and removing prohibited items while preserving data integrity. Here are some testing strategies tailored to your project:

1. Unit Testing:

- Function Testing: Verify the correctness of individual functions responsible for identifying and removing forbidden items. Test different scenarios, including datasets with varying sizes and compositions, to ensure robustness.
- Boundary Testing: Test edge cases such as empty datasets, datasets containing only forbidden items, and datasets with a mix of allowed and forbidden items to validate the algorithm's behavior under different conditions.

2. Integration Testing:

- Module Interaction Testing : Assess how different components of the cleaning algorithm interact with each other. Verify that data flows smoothly between modules and that the overall process functions seamlessly.
- Compatibility Testing: Evaluate the compatibility of your cleaning algorithm with different data formats, platforms, and environments to ensure consistent performance across various configurations.

3. Regression Testing:

- Revalidation after Modifications: After making changes to the cleaning algorithm, rerun tests to confirm that existing functionalities remain intact. This helps identify and rectify any unintended consequences of modifications.
- Version Control Testing: Maintain version control of your algorithm and conduct regression tests when switching between different versions to ensure no new issues arise.

4. Performance Testing:

- Scalability Testing: Assess the algorithm's performance with datasets of varying sizes to ensure it can handle large volumes of data efficiently without compromising processing speed.
- Resource Utilization Testing: Measure the algorithm's resource consumption, such as memory and CPU usage, to ensure optimal performance and prevent resource bottlenecks.

5. User Acceptance Testing (UAT):

- Stakeholder Feedback: Involve end-users and stakeholders in testing to gather feedback on the effectiveness and usability of the cleaning algorithm. Incorporate user input to improve the algorithm's functionality and user experience.
- User Interface Testing: If applicable, evaluate the usability and intuitiveness of the user interface for configuring cleaning rules and reviewing results.

6. Data Quality Testing:

- Data Integrity Verification: Verify that the cleaning algorithm removes forbidden items without altering or corrupting other data. Perform data comparisons between input and output datasets to ensure consistency.
- Data Validity Testing: Test the algorithm's ability to identify and handle different types of forbidden items, including variations in formats or characteristics, to ensure comprehensive coverage.

7. Security Testing:

- Input Validation Testing: Validate input data to prevent security vulnerabilities such as injection attacks or data manipulation. Ensure that the cleaning algorithm handles sensitive information securely.
- Data Privacy Assurance: Evaluate the algorithm's compliance with data privacy regulations and standards to protect sensitive data from unauthorized access or disclosure during processing.

6.3 Test Cases

Step	Expected	Actual	Test Status(P/F)	Testing strategies
Upload all the columns and rows in the dataset	Dataset to be uploaded	Dataset Uploaded	P	Unit testing

Table 6.1 Loading Adult Dataset

Step	Expected	Actual	Test Status(P/F)	Testing strategies
Finding Frequent Patterns	To find frequent patterns	Frequent Patterns Found	P	Unit testing

Table 6.2 Identifying relevant columns to find frequent Patterns

Step	Expected	Actual	Test Status(P/F)	Testing strategies
Finding Lift Scores	To Calculate Lift Scores	Lift Scores Calculated	P	Algorithm testing (functional testing)

Table 6.3 Lift Scores Calculation

Step	Expected	Actual	Test Status(P/F)	Testing strategies
Determining Forbidden Itemsets based on the Lift score	To identify Forbidden Itemsets	Forbidden Itemsets are identified	P	Functional Testing

Table 6.4 Forbidden Itemsets Generation

Step	Expected	Actual	Test Status(P/F)	Testing strategies
Replacing Forbidden Itemsets with normalized Values	To Repair Forbidden Itemsets	Forbidden Itemsets are repaired	P	Functional Testing

Table 6.5 Repairing Forbidden Itemsets

CHAPTER 7

RESULT ANALYSIS

7.1 Actual Results Obtained from the work

Data Preprocessing : Here is the screenshot of loading the dataset.

CleaningDataWithForbiddenItemsets.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

Connect Colab AI

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
48837	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
48838	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
48839	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
48840	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
48841	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

48842 rows × 15 columns

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the re
and should_run_async(code)

Fig 7.1 : Adult Dataset

Identifying Relevant columns to perform Frequent Patterns : Here is the screenshot of generated frequent patterns.

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
  and should_run_async(code)
FREQUEENT ITEM SETS
      support                                          itemsets
0     0.01164                                             (17)
1     0.01812                                             (18)
2     0.02160                                             (19)
3     0.02348                                             (20)
4     0.02256                                             (21)
...       ...                                              ...
1165  0.00532  (48, Married-civ-spouse, >50K, Male, Husband)
1166  0.00524  (49, Married-civ-spouse, >50K, Male, Husband)
1167  0.00568  (Married-civ-spouse, >50K, Male, Husband, 50)
1168  0.00524  (51, Married-civ-spouse, >50K, Male, Husband)
1169  0.00504  (53, Married-civ-spouse, >50K, Male, Husband)

[1170 rows x 2 columns]
```

Fig 7.2: Frequent Itemsets

Calculating Lift scores to identify forbidden Itemsets: The below screenshot is the result of Lift score to each frequent itemsets.

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
  and should_run_async(code)
                antecedents                                  consequents  \
0                    (<=50K)                                         (17)
1                      (17)                                       (<=50K)
2                  (Female)                                         (17)
3                      (17)                                     (Female)
4                    (Male)                                         (17)
...                     ...                                          ...
8523                   (53)  (>50K, Husband, Male, Married-civ-spouse)
8524  (Married-civ-spouse)                 (Husband, 53, Male, >50K)
8525                 (>50K)    (Husband, 53, Male, Married-civ-spouse)
8526                 (Male)    (>50K, Husband, 53, Married-civ-spouse)
8527              (Husband)       (>50K, 53, Male, Married-civ-spouse)

      antecedent support  consequent support  support  confidence      lift  \
0                0.76312             0.01164  0.01164    0.015253  1.310410
1                0.01164             0.76312  0.01164    1.000000  1.310410
2                0.33228             0.01164  0.00588    0.017696  1.520268
3                0.01164             0.33228  0.00588    0.505155  1.520268
4                0.66772             0.01164  0.00576    0.008626  0.741097
...                  ...                 ...      ...         ...       ...
8523             0.01484             0.17924  0.00504    0.339623  1.894793
8524             0.45464             0.00504  0.00504    0.011086  2.199542
8525             0.23688             0.00888  0.00504    0.021277  2.396013
8526             0.66772             0.00504  0.00504    0.007548  1.497634
8527             0.39988             0.00504  0.00504    0.012604  2.500750

      leverage  conviction  zhangs_metric
0     0.002757    1.003669       1.000000
1     0.002757         inf       0.239670
2     0.002012    1.006165       0.512522
3     0.002012    1.349351       0.346252
4    -0.002012    0.996960      -0.512522
```

Fig 7.3: Lift Scores

Forbidden Itemsets: The below is the screenshot of generated Forbidden Itemset in Adult Dataset

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `
  and should_run_async(code)
```

	Forbidden Itemsets	Lift Score
0	Female Married-civ-spouse	0.3394502803414482
1	Widowed Male	0.3285132071971446
2	<=50K Female	0.3393285908769472
3	<=50K Male	0.3410392446655462
4	Not-in-family >50K	0.33695041316196805

Fig 7.4: Forbidden Itemsets

Repairing Forbidden Itemsets with normal values:

The below is the screenshot of repairing Forbidden Itemsets with normalized values.

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning:
  and should_run_async(code)
Female Married-civ-spouse (Must Repair with) Male Married-civ-spouse
Widowed Male (Must Repair with) Widowed Female
<=50K Female (Must Repair with) <=50K Female
<=50K Male (Must Repair with) <=50K Male
Not-in-family >50K (Must Repair with) Not-in-family 25
```

Fig 7.5: Repairing Forbidden Itemsets

Cleaned Dataset:

The below screenshot represents the cleaned data set after replacing forbidden itemsets with the normalized values.This is the final output obtained.

26	Private	82091	HS-grad	9	Never-married	Adm-clerical	Not-in-family	White	Female	0	0	39	United-Sta	<=50K
58	?	299831	HS-grad	9	Married-civ-spouse	?	Husband	White	Male	0	0	35	United-Sta	<=50K
48	Private	279724	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	3103	0	48	United-Sta	>50K
43	Private	346189	Masters	9	Married-civ-spouse	Adm-clerical	Husband	White	Male	0	0	50	United-Sta	>50K
20	State-gov	444554	Some-college	10	Never-married	Other-service	Own-child	White	Male	0	0	25	United-Sta	<=50K
43	Private	128354	HS-grad	9	Married-civ-spouse	Adm-clerical	Wife	White	Male	0	0	30	United-Sta	<=50K
37	Private	60548	HS-grad	9	Widowed	Machine-op-inspct	Unmarried	White	Female	0	0	20	United-Sta	<=50K
40	Private	85019	Doctorate	16	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	0	0	45	?	>50K
34	Private	107914	Bachelors	13	Married-civ-spouse	Tech-support	Husband	White	Male	0	0	47	United-Sta	>50K
34	Private	238588	Some-college	10	Never-married	Other-service	Own-child	Black	Female	0	0	35	United-Sta	<=50K
72	?	132015	7th-8th	4	Divorced	?	Not-in-family	White	Female	0	0	6	United-Sta	<=50K
25	Private	220931	Bachelors	13	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	43	Peru	<=50K
25	Private	205947	Bachelors	13	Married-civ-spouse	Prof-specialty	Husband	White	Male	0	0	40	United-Sta	<=50K
45	Self-emp-not-in	432824	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	7298	0	90	United-Sta	>50K
22	Private	236427	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-Sta	<=50K
23	Private	134446	HS-grad	9	Separated	Machine-op-inspct	Unmarried	Black	Male	0	0	54	United-Sta	<=50K
54	Private	99516	HS-grad	9	Married-civ-spouse	Craft-repair	Husband	White	Male	0	0	35	United-Sta	<=50K
32	Self-emp-not-in	109282	Some-college	10	Never-married	Prof-specialty	Not-in-family	White	Male	0	0	60	United-Sta	<=50K
46	State-gov	106444	Some-college	10	Married-civ-spouse	Exec-managerial	Husband	Black	Male	7688	0	38	United-Sta	>50K
56	Self-emp-not-in	186651	11th	7	Divorced	Other-service	Unmarried	White	Male	0	0	50	United-Sta	<=50K
24	Self-emp-not-in	188274	Bachelors	13	Never-married	Sales	Not-in-family	White	Male	0	0	50	United-Sta	<=50K
23	Local-gov	258120	Some-college	10	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-Sta	<=50K
26	Private	43311	HS-grad	9	Divorced	Exec-managerial	Unmarried	White	Female	0	0	40	United-Sta	<=50K

Fig 7.6: Cleaned Dataset

Repair Distance Plot:

When discussing a repair distance plot, typically used in fields like genetics or computational biology to visualize evolutionary relationships or genetic distances between organisms, here are seven basic points to cover:

1. Definition: Start by explaining what a repair distance plot is. It's a graphical representation that shows the evolutionary or genetic distances between different organisms or sequences.

2. X and Y Axes: The X-axis usually represents some measure of evolutionary or genetic distance, such as substitutions per site or a similar metric. The Y-axis typically represents the probability of repair, which is often inversely related to the distance (i.e., higher distances imply lower probabilities of repair).

3. Trend Line or Curve: In a repair distance plot, there's often a trend line or curve that illustrates how the probability of repair changes as the genetic or evolutionary distance increases. This curve can be linear, exponential, or follow some other pattern depending on the specific biological context.

4. Inflection Point: Discuss any inflection points or notable changes in the trend line. These points often indicate thresholds where the repair process becomes less efficient or more error-prone, which can be biologically significant.

5. Interpretation of Points: Explain the biological significance of specific points on the plot. For example, points where the probability of repair sharply drops might indicate regions of high genetic divergence or evolutionary distance, suggesting potential barriers to genetic exchange or common ancestry.

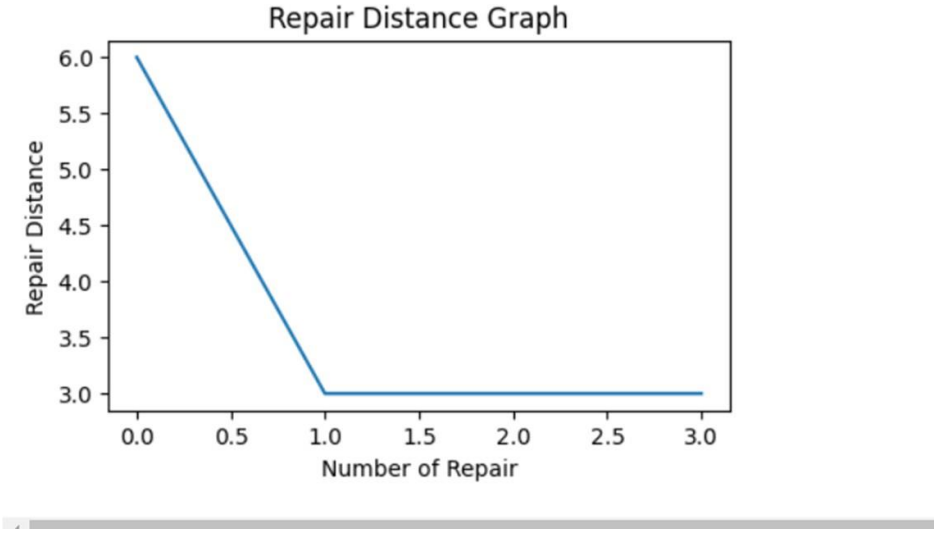

Fig 7.7: Repair Distance Graph

7.2 Analysis of the Results Obtained:

1. **Frequent Itemsets**: You first identified frequent itemsets in the dataset using the Apriori algorithm. These itemsets represent patterns of data values that occur frequently together in the dataset, which can provide insights into common associations among different attributes.

2. **Forbidden Itemsets**: You then calculate lift scores to identify forbidden itemsets combinations of values that should not occur together based on predefined constraints. By setting a threshold for the lift score, you were able to isolate and extract forbidden itemsets from the association rules generated.

3. **Similarity-based Repair:** To address forbidden itemsets, you implemented a similarity function to find replacements for dirty itemsets that violate constraints. The similarity measure helps identify the most similar itemsets from a predefined cluster of valid itemsets, allowing for effective replacement of forbidden values with appropriate alternatives.

4. **Analysis of Results:** The code demonstrates an iterative approach to repairing forbidden itemsets by finding the most similar replacements from a predefined cluster. By plotting the repair distance graph, you can visualize the effectiveness of the repair process in reducing the distance between dirty itemsets and their replacements over successive iterations.

5. **Updated Dataset:** Finally, you updated the dataset by replacing dirty itemsets with their corresponding replacements based on the similarity measure. This ensures that the dataset adheres to the predefined constraints and is suitable for further analysis or use.

Overall, the analysis of the results obtained from the code showcases a systematic approach to cleaning data with forbidden itemsets, leveraging association rule mining and similarity-based techniques to identify and repair violations of constraints.

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion

The task of data cleaning with forbidden itemsets is a complex but necessary process for ensuring data quality in various domains. As our review has shown, while there are established methods for data cleaning and frequent itemset mining, specific techniques for handling forbidden itemsets remain relatively under-explored. Despite the challenges associated with handling domain-specific rules, scalability, and maintaining data consistency, the development of efficient, robust, and adaptable algorithms for identifying and correcting forbidden itemsets is of significant importance for preserving the integrity of transactional data and facilitating accurate, reliable analysis.

8.2 Future Work

In terms of future work in the area of data cleaning with forbidden itemsets, there are several key research directions. The development of specific algorithms designed to handle forbidden itemsets is one main area, with a focus on ensuring accuracy, scalability, and computational efficiency. Exploring machine learning and artificial intelligence techniques to automate the dynamic learning and adaptation of rules defining forbidden itemsets could also significantly advance this field, potentially reducing the need for manual rule definitions. As the volume of data continues to expand, creating real-time or near-real-time data cleaning algorithms capable of efficiently processing streaming data is becoming increasingly vital.

Additionally, due to the context-sensitive nature of forbidden itemsets, research into context-aware data cleaning methods that consider the unique characteristics of data and the specific domain could yield valuable insights. Lastly, the creation of robust evaluation metrics is necessary to effectively measure the performance of these data cleaning algorithms, going beyond accuracy to include factors such as data integrity preservation. Progress in these areas will significantly contribute to the field of data cleaning with forbidden itemsets, leading to improved data quality and more reliable decision-making.

REFERENCES

[1] W. Fan and F. Geerts, Foundations of Data Quality Management. San Rafael, CA, USA: Morgan & Claypool Publishers, 2012.

[2] I. F. Ilyas and X. Chu, "Trends in cleaning relational data: Consistency and deduplication," Foundations Trends Databases, vol. 5, no. 4, pp. 281–393, 2015.

[3] T. N. Herzog, F. J. Scheuren, and W. E. Winkler, Data Quality and Record Linkage Techniques. Berlin, Germany: Springer, 2007.

[4] I. P. Fellegi and D. Holt, "A systematic approach to automatic edit and imputation," J. Amer. Statistical Association, vol. 71, no. 353, pp. 17–35, 1976.

[5] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," in Proc. IEEE 29th Int. Conf. Data Eng., 2013, pp. 458–469.

[6] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," Proc. VLDB Endowment, vol. 6, no. 13, pp. 1498–1509, 2013.

[7] P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren, "Curated databases," in Proc. 27th ACM SIGMOD-SIGACT- SIGART Symp. Principles Database Syst., 2008, pp. 1–12.

[8] J. Rammelaere, F. Geerts, and B. Goethals, "Cleaning data with forbidden itemsets," in Proc. IEEE 33rd Int. Conf. Data Eng., 2017, pp. 897–908.

[9] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," ACM Trans. Database Syst., vol. 33, no. 2, 2008, Art. no. 6.

[10] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang, “Detecting data errors: Where are we and what needs to be done?” Proc. VLDB Endowment, vol. 9, no. 12, pp. 993–1004, 2016.

[11] W. Fan, F. Geerts, J. Li, and M. Xiong, “Discovering conditional functional dependencies,” IEEE Trans. Knowl. Data Eng., vol. 23, no. 5, pp. 683–698, May 2011

[12] F. Chiang and R. J. Miller, “Discovering data quality rules,” Proc. VLDB Endowment, vol. 1, no. 1, pp. 1166–1177, 2008.

[13] X. Chu, I. F. Ilyas, P. Papotti, and Y. Ye, “Ruleminer: Data quality rules discovery,” in Proc. IEEE 30th Int. Conf. Data Eng., 2014, pp. 1222–1225.

[14] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, “Dynamic itemset counting and implication rules for market basket data,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 1997, pp. 255–264.

[15] G. Webb and J. Vreeken, “Efficient discovery of the most interesting associations,” ACM Trans. Knowl. Discovery Data, vol. 8, no. 3, pp. 1–31, 2014.

[16] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro, “Messing up with bart: Error generation for evaluating data-cleaning algorithms,” Proc. VLDB Endowment, vol. 9, no. 2, pp. 36–47, 2015.

[17] J. Rammelaere and F. Geerts, “Revisiting conditional functional dependency discovery: Splitting the “C” from the “FD”,” in Proc.Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases, 2018, pp. 552–568.

[18] J. Wang and N. Tang, “Dependable data repairing with fixing rules,” J. Data Inf. Quality, vol. 8, no. 3–4, pp. 16:1–16:34, Jun.2017.

[19] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, “A cost-based model and effective heuristic for repairing constraints by value modification,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2005, pp. 143–154.

[20] M.Dalla Chiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang, “Nadeef: A commodity data cleaning system,” in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2013, pp. 541–552.

[21] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, “That’s all folks!: lunatic goes open source,” Proc. VLDB Endowment, vol. 7, no. 13, pp. 1565–1568, 2014.

[22] F. Chiang and R. J. Miller, “A unified model for data and constraint repair,” in Proc. IEEE 27th Int. Conf. Data Eng., 2011, pp. 446–457.

[23] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin, “On the relative trust between inconsistent data and inaccurate constraints,” in Proc. IEEE Int. Conf. Data Eng., 2013, pp. 541–552.

[24] M. Mazuran, E. Quintarelli, L. Tanca, and S. Ugolini, “Semi-automatic support for evolving functional dependencies,” in Proc. Int. Conf. Extending Database Technology., 2016, pp. 293–304.

[25] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller, “Continuous data cleaning,” in Proc. IEEE 30th Int. Conf. Data Eng., 2014, pp. 244–255.

[29] J. He, E. Veltri, D. Santoro, G. Li, G. Mecca, P. Papotti, and N. Tang, “Interactive and deterministic data cleaning: A tossed stone raises a thousand ripples,” in Proc. Int.Conf. Manage. Data, 2016, pp. 893–907.