

Report for the Degree of Master of Computer Science

End to End Automation in Extract, Transform and Load Pipeline



Pikesh Maharjan

(LC00015002358)

Masters in Computer Science (MCS) IIMS college

Computer Science Department

Lincoln University, Malaysia

January 2025

Research project for the Degree of Master of Computer Science

End to End Automation in Extract, Transform and Load Pipeline

Supervised by Prof. Sudan Jha, Ph.D.

A report submitted in partial fulfilment of the requirements for the
Degree of Master of Computer Science

Pikesh Maharjan

(LC00015002358)

Masters in Computer Science (MCS) IIMS college

Computer Science Department

Lincoln University, Malaysia

January 2025

Dedication

I would like to dedicate this report to my parents and my sister, whose unwavering support and encouragement have been my guiding light throughout this academic journey. Words cannot fully express the depth of my gratitude for your love and sacrifices, but this stands as a humble token of my appreciation.

Declaration

I hereby declare that the work presented in this thesis is the result of my original research efforts. All sources and references used in this study have been duly acknowledged and cited in accordance with academic standards. I owe all the liabilities relating to the accuracy and authenticity of the data and any other information included hereunder.

This thesis has not been submitted previously, in whole or in part, for the award of any degree or diploma in any other institution.

.....

Pikesh Maharjan

LC00015002358

Date:

Recommendation

This is to certify that the thesis entitled “**End to End Automation in Extract, Transform and Load Pipeline**” has been prepared and submitted by **Pikesh Maharjan** under my supervision in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

To the best of my knowledge, this thesis is an original work that meets the standards set by the university, and I recommend it for evaluation.

.....

Prof. Dr. Sudhan Jha

Dept. of Computer Science and Multimedia

Lincoln International College of Management & IT

Date:

Certificate

This is to certify that the thesis entitled “**End to End Automation in Extract, Transform and Load Pipeline**” submitted by **Pikesh Maharjan**, in partial fulfillment of the requirements for the degree of **Master of Science in Computer Science** is a bona fide record of original research work carried out under the supervision of **Dr. Prof Sudhan Jha**.

We hereby certify that this thesis is the result of the candidate’s own work and effort, and to the best of our knowledge, it has not been submitted to any other institution for the award of any degree or diploma.

This thesis is hereby approved and accepted.

Name of external Examiner

Signature

Date:

External examiner

Prof. Dr. Sudan Jha

Signature

Date:

Supervisor

Monica Regmi

Signature

Date:

Co-supervisor

Acknowledgements

I would like to take this opportunity to thank all those who assisted and helped me throughout my research and writing of this thesis.

First, I am deeply grateful to my supervisor, Prof. Dr. Sudan Jha for his helpful advice, detailed reviews, and continuous support, and encouragement towards writing this thesis. I learned much from his expertise, guidance, and inspiration in the direction and scope of the study.

I'm also grateful to my co-supervisor, Monica Regmi for her helpful advice and support during the research.

I would like to take this moment to express my sincerest gratitude to my friends and family, who have offered their constant support, patience and understanding throughout my challenging journey. The encouragement and belief in my capabilities from those I love inspires and motivates me.

To everyone who has directly or indirectly contributed to this research, thank you! I feel incredibly lucky to have learned from and worked with all of you.

Signature

Name of Students: Pikesh Maharjan

Registration Number: LC00015002358

Date: 18th April 2025

Abstract

This thesis explores the development and implementation of an **End-to-End Automation** process for an **Extract, Transform, and Load (ETL) pipeline**, which focuses at streamlining various data processing workflows into single unified solution. The research focuses on automating the traditional ETL process to improve efficiency, reduce human interventions and thus the issues caused by manual works, and facilitate data integration from delimited file sources to a central repository. The study identifies the challenges involved in the manual ETL processes, including time consumption, data inconsistency, and complexity in managing large number of data files.

To address the challenges of manual ETL process, an automated solution was designed, which integrates various methodologies to extract data from ASCII Delimited sources, and load it into a structured database in just a single click and few inputs from the user. The system was developed using Python and associated libraries, including Pandas and SQLAlchemy, which enabled efficient data loading and interaction with databases.

In conclusion, this study validates the effectiveness of end-to-end ETL automation in modern data engineering practices. The findings highlight the importance of automating repetitive and time-consuming tasks in ETL pipelines, making it a crucial step toward omitting manual steps in data management. Future work could explore further enhancements, such as loading various files sources other than ASCII delimited files, integrating machine learning models for data validation and predictive analytics, and applying complex transformations logics.

Keywords: ETL automation, Data Pipeline, Data Processing, Automated Data Loading, Machine learning based data type detection, Automated Schema Detection, Data Engineering.

Table of Contents

Dedication.....	3
Declaration	4
Recommendation	5
Certificate	6
Acknowledgements	7
Abstract	8
List of Figures.....	13
List of abbreviation/acronyms.....	14
CHAPTER 1: INTRODUCTION	15
1.1 Background of the study	19
1.2 Statement of the Problem.....	20
1.3 Research Questions	22
1.4 Aims and Objectives	23
1.4.1 General Objective:.....	23
1.4.2 Specific Objectives:.....	23
1.5 Significance of the Study.....	24
1.6 Scope and Limitations of the Study.....	25
1.6.1 Time Constraints	26
1.6.2 Resource Constraints.....	26
1.6.3 Shifting Data Technologies	26
1.6.4 Study Scope.....	26
1.6.5 Geographical and Sectorial Limitations	26
CHAPTER 2: LITERATURE REVIEW	27

2.1 Thematic Review	27
2.2 Theoretical Review	28
2.3 Conceptual Review	28
2.3 Research Gap	29
CHAPTER 3: METHODOLOGY	30
3.1 Research Design	30
Problem Identification:	30
Requirements Analysis:	30
System Design and Architecture:	31
Prototyping:	31
Results Analysis and Refinement:	31
3.1.1 Research Approach	32
3.2 System Architecture	33
3.2.1 Data Ingestion	34
3.2.2 Identification of Meta Data	34
3.2.3 Configuration File Generation	35
3.2.4 Human Interaction for Validation	36
3.2.5 Schema Generation	36
3.2.6 Data Loading	36
3.2.7 Logging and Validation	37
3.3 Tools and Frameworks	38
3.3.1 Programming Language	38
3.3.2 Libraries and Packages	38
3.3.3 Database	39
3.3.4 Text and Configuration Tools	39
3.4 Data Collection and Preparation	39
3.4.1 Synthetic Data Generation	40

3.4.2 Rationale for use of Synthetic Data.....	41
3.4.3 Data Structure and Formats.....	41
3.5 Data Preprocessing and Model Training	43
3.5.1 Rationale behind Algorithm Selection	43
3.5.2 Data Preprocessing.....	44
3.5.3 Model Training.....	46
3.12 Limitations in Methodology.....	48
CHAPTER 4: RESULTS AND DISCUSSIONS.....	49
4.1 Brief Recap.....	49
4.2 Objective Reiteration	49
4.3 Methodology Recap.....	50
4.3.1 Research Design.....	50
4.3.2 Study Area and Population.....	51
4.3.3 Sample Selection	52
4.4 Result Presentation	52
4.4.1 Data Ingestion	52
4.4.2 Metadata Identification	53
4.4.3 Configuration File Generation	53
4.4.4 Human Interaction.....	54
4.4.5 Schema Creation	54
4.4.6 Data Loading and Logging.....	55
4.5 Discussion	56
4.5.1 Alignment with Research Objectives	57
4.5.2 Insights from the Model Training	57
4.5.3 Challenges and Unexpected Results	58
4.5.4 Comparison to Previous Work	59
4.5.5 Implications for Future Research and Practice	59

4.7 Limitations of the Study	59
Chapter 5: Conclusion and Recommendations.....	62
5.1 Conclusion	62
5.2 Recommendations	63
REFERENCE	65

List of Figures

Fig. 1.1. Illustration of ETL [6]	15
Fig. 1.2. Illustration of Advanced ETL	16
Fig. 1.3. Comma Delimited File(CSV).....	17
Fig. 1.4. Semicolon Delimited.....	18
Fig. 1.5. Pipe Delimited()	18
Fig. 1.6. TAB Delimited (\t).....	18
Fig. 1.7 Tilde Delimited (~).....	18
Fig. 1.8. Caret Delimited (^).....	18
Fig. 3.1. Flow Chart of Methodology	33
Fig. 3.2. Inputs Required from User	34
Fig. 3.3. Identification of Meta Data	35
Fig. 3.4. Sample of Configuration File	35
Fig. 3.5. Sample of Log File	37
Fig. 4. 1 Meta Data Identification.....	53
Fig. 4.2. Configuration File Generation	54
Fig. 4.3. Sample Configuration File	54
Fig. 4.4. Schema in SQL server.....	55
Fig. 4.5. Data Loaded to SQL Server	56
Fig. 4.6. Log File	56

List of abbreviation/acronyms

ETL	Extract, Transform, Load
SQL	Structured Query Language
ML	Machine Learning
DBMS	Database Management System
CSV	Comma-Separated Values
JSON	JavaScript Object Notation
Pandas	Python Data Analysis Library

CHAPTER 1: INTRODUCTION

In today's digital transformation era, data has become an invaluable asset for organizations, aiding in decision-making and offering insights into customer behavior and market trends. However, the vast amount of information generated from various sources presents challenges in effectively analyzing, transforming, and utilizing this data. This is where the extraction, transformation, and loading (ETL) process come into play. ETL involves extracting data from multiple sources and recording or loading all the data into a centralized repository, such as data warehouses or data lakes. This process ensures that businesses can make informed, data-driven decisions by relying on consistent, accurate, and up-to-date information for their analytics with the help of data analysis and visualization of the analyzed data.

Extract, Transform and Load (ETL) is a process in data management and integration that plays an important role in digital and modern businesses to take informed decision based on the data [9]. It is used to develop and improve applications for analysis and thus making it easier to take data driven decisions. It involves three primary steps:

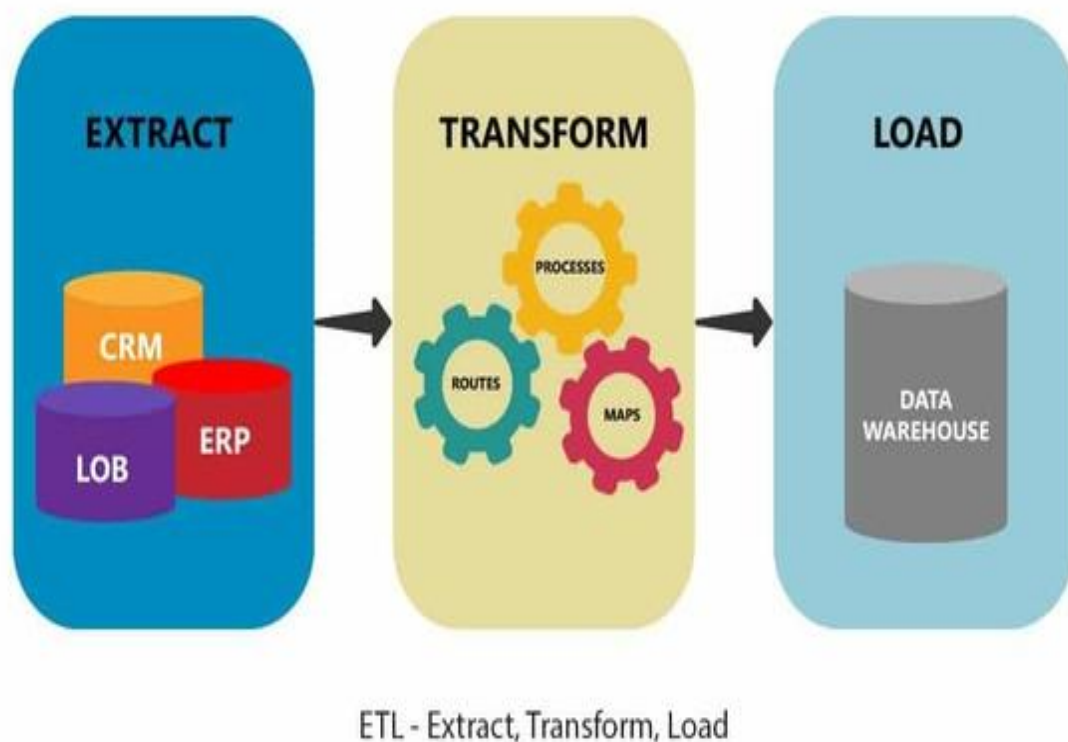


Fig. 1.1. Illustration of ETL [6]

Extracting data from various sources involves transforming it into a suitable format and loading it into a central repository, like a data warehouse or data lakes, for analysis and decision-making with help of visualizations. This phase includes gathering data from diverse sources such as databases, APIs, files, or cloud storage and extracting data from these sources. These sources can vary in structure, including relational databases, spreadsheets, or unstructured web data, and can differ significantly from one another and thus may be structured, semi-structured or un-structured data.

The next step involves cleaning, formatting, and standardizing raw data from the sources collected in extract phase to align with business requirements and the repositories where the data will be loaded. This transformation process may include eliminating duplicate entries, addressing missing values, and converting the data into a format that satisfies business needs. Business may need to replace null values with some other default values, or some may completely ignore rows with null values. Thus, transformation phase differs to organization to organization.

Loading is the final step in the ETL process, where transformed data is loaded into systems like a Data Warehouse or Data Lake, making it accessible for users who need it for advanced analysis and visualization. This process ensures that data from various sources is prepared for analysis and visualization. The following figure illustrates the use of ETL in a diagram format.

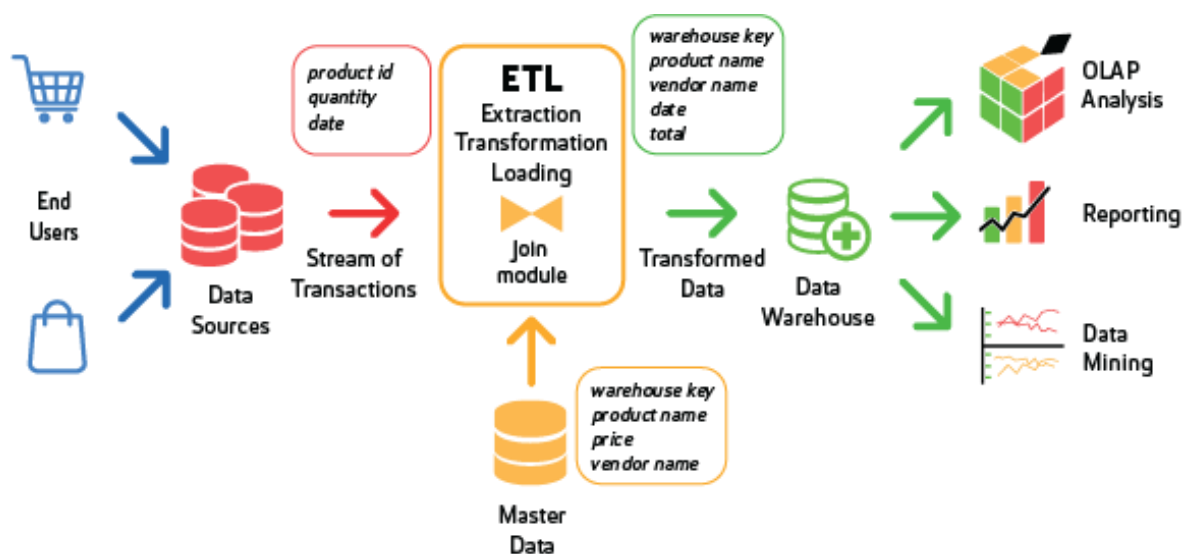


Fig. 1.2. Illustration of Advanced ETL

As Mentioned above, organization collect huge amounts of information which may be in various forms and among them one of the form is delimited files. Delimited files are plain text files where each line represents a record (row), and individual fields within the records are separated by a special characters which is known as delimiters. Here the row represents a single record which contains all the information related to a single entity whereas column represents a specific fields shared by all the records in the dataset. Each column has a name and they contains the values for that attributes for all the rows. The fields that are separated by the delimiters are known as column. All the delimited files are structured data and they can be in forms such as csv, txt and data formats. Each row in the file typically represents one data record, and each field within the row corresponds to a column in the logical structure of the data. Understanding and correctly identifying this delimiter is critical because it allows systems to interpret and organize the data into a structured form before loading it into a database. Once the data is separated into fields using delimiters, the fields (columns) can be identified. It is essential that the columns are aligned and have same number across all the records in a file. If any columns are mismatched, then it could lead to the issues such as failure of loading data or loading incorrect data into the databases.

For e.g., the files storing customer information might have rows like:

```
Name, Age, Email, JoinDate
Alice Johnson, 29, alice.johnson@example.com, 2021-06-15
Bob Smith, 35, bob.smith@example.com, 2020-12-01
Charlie Lee, 42, charlie.lee@example.com, 2019-07-23
```

Fig. 1.3. Comma Delimited File(CSV)

Here, the delimiters are comma (,) which is used to separate each column values, the first line contains columns name: Name, Age, Email and JoinDate , Each line is a row representing one individual's data and the new line (\n) after each row indicates the end of the record. These are the essentials part of the delimited files which plays a crucial role in different process of the ETL (Extracting, Transforming and Loading). Hence, these fields should be correctly identified in delimited files, so that all the processes of ETL is completed successfully.

Similarly, here are some examples of the files with different delimiters.

```
Name;Age;Email;JoinDate
Alice Johnson;29;alice.johnson@example.com;2021-06-15
Bob Smith;35;bob.smith@example.com;2020-12-01
Charlie Lee;42;charlie.lee@example.com;2019-07-23
```

Fig. 1.4. Semicolon Delimited

```
Name|Age|Email|JoinDate
Alice Johnson|29|alice.johnson@example.com|2021-06-15
Bob Smith|35|bob.smith@example.com|2020-12-01
Charlie Lee|42|charlie.lee@example.com|2019-07-23
```

Fig. 1.5. Pipe Delimited(|)

Name	Age	Email	JoinDate
Alice Johnson	29	alice.johnson@example.com	2021-06-15
Bob Smith	35	bob.smith@example.com	2020-12-01
Charlie Lee	42	charlie.lee@example.com	2019-07-23

Fig. 1.6. TAB Delimited (\t)

```
Name~Age~Email~JoinDate
Alice Johnson~29~alice.johnson@example.com~2021-06-15
Bob Smith~35~bob.smith@example.com~2020-12-01
Charlie Lee~42~charlie.lee@example.com~2019-07-23
```

Fig. 1.7 Tilde Delimited (~)

```
Name^Age^Email^JoinDate
Alice Johnson^29^alice.johnson@example.com^2021-06-15
Bob Smith^35^bob.smith@example.com^2020-12-01
Charlie Lee^42^charlie.lee@example.com^2019-07-23
```

Fig. 1.8. Caret Delimited (^)

1.1 Background of the study

In today's world, where vast amounts of data are generated every second, ETL pipelines are essential for capturing, transferring, and loading this information. These pipelines are critical in data management and analysis, facilitating the movement and transformation of raw data that is subsequently loaded into databases, data warehouses, or other storage systems. The transformed data can then be turned into actionable insights for any data-driven organization.

Traditional ETL workflows are typically manual, necessitating human input in several steps like schema detection and classification. This reliance on manual processes can be time-consuming and increases the risk of errors. Such manual ETL methods may result in inconsistencies, problems, and delays that hinder the accurate and timely extraction of valuable insights from the data.

With the rapid increase in the volume and variety of data, there is an urgent need to automate ETL pipelines to reduce human effort. Automated ETL pipelines can manage vast amounts of data across many fields with little human intervention. This automation helps to reduce delays and minimizes potential issues that may arise from human errors.

Automated ETL processes utilize advanced technologies such as machine learning and artificial intelligence to handle tasks that once needed human intervention. Machine learning algorithms can identify and automatically adjust to changes in data schemas. For instance, they can recognize the characteristics of the data schema and apply them effectively. They also categorize data types and accurately map data fields. This not only accelerates the ETL process but also improves the accuracy and consistency of the results being processed.

Automated ETL pipelines have proven to be more flexible and adaptable to evolving business needs. They can easily scale to handle growing data volumes and can be quickly modified to integrate new data sources or adjust to changes in existing ones. This flexibility is crucial in today's fast-paced business landscape, where organizations must be agile and responsive to maintain a competitive edge.

In summary, automation of ETL pipelines is essential for modern data management and analytics. It addresses the challenges posed by traditional manual processes such as time consumption errors and scalability issues.

1.2 Statement of the Problem

The manual creation of ETL processes has long been a challenge in data-driven industries, particularly given the rapid increase in data volume. Traditional workflows often demand significant human involvement for various tasks. These tasks encompass file ingestion, where raw data files are brought into the system; metadata recognition, which involves identifying and comprehending the structure and characteristics of the data; and mapping fields from source to destination to ensure proper alignment of data across different systems.

The reliance on manual processes leads to significant inefficiencies in both time and resource management. Human involvement is not only time-consuming but also increases the likelihood of errors, especially when handling data in various formats and levels. This can create data inconsistencies that undermine the integrity of the data and the insights drawn from it. Another labor-intensive aspect of manual ETL processes is the identification of data types across different datasets. As discussed in Chapter 3D, regulatory methods for classifying data that depend on predefined rules often lack the flexibility needed to adapt to new and evolving data types. These methods require considerable maintenance to keep pace with changes in data structures and formats, making them less efficient over time.

Despite the potential of machine learning to address these challenges, its application in the ETL domain remains limited. The complexity associated with implementing machine learning models for tasks like schema detection, data type categorization, and defining transformation logic has slowed their adoption. However, as machine learning technologies evolve and become more user-friendly, there is an increasing opportunity to utilize these tools to automate and enhance ETL processes.

Another important challenge is the absence of dynamic configuration systems that facilitate smooth integration with user-defined transformation logic. Whenever a new dataset is added, the pipeline code often requires modifications. This not only consumes a lot of time but also greatly restricts the flexibility of the ETL system. The necessity to manually tweak the code for each new dataset leads to delays and adds to the workload for data engineers, making the system less agile and less responsive to evolving data needs.

There is a notable gap in automating the final loading stage to target systems like SQL Server. This stage frequently requires repeated manual validations and corrections to ensure that the data is loaded correctly and meets the necessary standards. These manual

interventions can result in inconsistencies and errors, which hinder the availability of data for analysis and decision-making. The absence of automation not only slows down the overall ETL process but also raises the risk of data inconsistencies, ultimately impacting the quality and reliability of the insights derived from the data.

The current state of data management heavily relies on specific ETL (Extract, Transform, and Load) tools to carry out ETL processes. Many companies depend on commercial ETL software, which often comes with drawbacks like high costs, limited customization options, and vendor lock-in. Additionally, these tools frequently necessitate specialized knowledge and training, posing a challenge for smaller organizations or those with constrained resources.

Relying on these tools can complicate the adaptation to new data sources or evolving business needs. This, in turn, impacts the flexibility and scalability of ETL processes. For instance, when a new data source is added or an existing one undergoes significant changes, substantial modifications may be necessary for the ETL pipeline. This often requires reconfiguring the tool, updating scripts, and ensuring compatibility with new data formats, which can be both time-consuming and resource intensive [8].

Furthermore, the high costs associated with commercial ETL tools can be prohibitive for smaller organizations or startups. These costs include not only the initial price or subscription fees but also ongoing maintenance support and training expenses [8]. The financial burden can limit the ability of these organizations to invest in other critical areas like data analytics and business intelligence thereby hindering their overall growth and competitiveness.

Vendor lock-in is a significant concern when relying on specific tools and methods like GAP. Once an organization has made a substantial investment in a particular tool or software, transitioning to a different solution can be challenging and expensive. Additionally, it restricts the ability to tailor and enhance ETL processes to align with unique business requirements, as organizations often find themselves limited by the features and constraints of the selected tool.

This research seeks to address these issues by automating the ETL processes without the need of ETL Tools, implementing Machine learning for data type identification, dynamic configuration for user-driven transformation and loading data into the SQL Server.

1.3 Research Questions

To guide this study, specific questions have been developed to dive into automating the ETL (Extract, Transform and Load) process. The Main focus is on how machine learning can help in detecting schemas. Also, another aspect it to test how well the system can accurately identify things like delimiters, record structures. We aim to discover how we can improve scalability, involve human interaction, and enhance overall performance. These research questions serve as a guide for examining smart ETL automation, offering a clear plan to assess its practical uses, impacts, and future potential in data engineering. The study seeks to address the following questions:

1. How can machine learning methods be integrated to identify data types in ETL pipelines?
2. How accurately can machine learning models detect column data types (e.g., int, float, varchar, date) from raw delimited files with varying formats?
3. Is ETL possible without using ETL tools?
4. Can the automatically generated configuration files (e.g., JSON schemas) be reliably used for SQL Server table creation and data loading without human intervention?
5. How can ETL be carried out with minimum human interaction?
6. How can ETL be automated for Bulk file loading?
7. Is it possible to determine the data length and delimiters in delimited files with high accuracy?
8. Does removing human interaction in ETL process increases or reduces accuracy?
9. Is it possible to detect different file delimiters (like comma, pipe, tab etc.) without user input?
10. Is the proposed system easy to use for someone with limited technical background?
11. Can the system handle different types of files like .csv, .txt, and .dat without needing to change the code?
12. Can the system be scaled to handle multiple files in a single run?

1.4 Aims and Objectives

This study has the objective of automating ETL pipeline for Delimited files and reducing as much as human intervention in ETL processing. It also aims to remove dependency on ETL tools for ETL processing paying high costs and a tech without knowledge of ETL tools can also process data and data load up to the SQL server with some simple users inputs that can be understood easily. This study has been presented with general and specific objectives.

1.4.1 General Objective:

1. To design and implement an automated ETL pipeline capable of handling ASCII delimited datasets without needing to change code and knowledge of ETL tools.

1.4.2 Specific Objectives:

1. Develop mechanisms for data type recognition using ML methods.
2. Design a system that detects and automatically process various file formats for loading purpose.
3. Identification of delimiters, column names, and record separators from structured text file using rule based techniques.
4. Create a configuration file (in JSON format) that accurately defines the schema for database table creation.
5. To enable optional human interaction for reviewing and modifying automatically detected metadata before schema creation in SQL.
6. Automate the data loading process into SQL Server.
7. To maintain detailed logs during data ingestion, transformation, and loading that track record count, errors, and processing time.
8. Remove the dependency on ETL tools.
As different ETL tools have different ways of handling ETL processes, data engineers are required to learn new things every time they switch to new ETL tool. Also, ETL tools comes with a price and thus using this system aims to remove the dependency on ETL tools.
9. Reduce human intervention and increase the ETL processing speed.
10. Ensure one can process data up to SQL server without ETL tool specific knowledge.

1.5 Significance of the Study

The automation of the complete ETL pipeline is set to revolutionize traditional data engineering methods. This research aims to reduce the reliance on human intervention in data processing, which in turn minimizes human errors and enhances data integration for better decision-making. By automating these processes, the study intends to create a framework for managing delimited file formats, focusing on accurately identifying data types and applying configurable transformation logic. These features can be utilized in real-world applications and align with current market trends, demonstrating that the research is both relevant and impactful.

This study aims to significantly enhance the ease and efficiency of ETL operations. As organizations produce and store vast amounts of data, the ability to manage this data with greater reliability and speed is essential. Automated ETL pipelines provide quicker and more accurate data processing than manual methods, as they reduce human errors through less human involvement. This ensures that data is readily available for analysis, reporting, and visualization. Improved efficiency will enable organizations to make faster, data-driven decisions, giving them a competitive edge in their industries.

This research aims to boost efficiency, offering significant benefits to data engineers and businesses by streamlining workflows and enhancing the overall quality and accessibility of data. Data engineers will have more time to focus on strategic initiatives that add value to the organization, as they will spend less time on repetitive manual tasks. Analysts will gain access to cleaner and more reliable data, allowing them to generate more accurate insights. Businesses will see improved data quality and faster decision-making processes, ultimately leading to better business outcomes. Additionally, the need to learn various ETL tools and their dependencies for the ETL process will be reduced, enabling engineers to concentrate on other important tasks.

The study aims to explore the current dependence on certain ETL tools, which frequently come with high expenses, restricted customization options, and vendor lock-in issues. By developing an automated ETL framework that is flexible and adaptable, this research provides organizations with a more affordable and customizable solution. This approach will alleviate the financial burden linked to commercial ETL technologies and enable organizations to customize their ETL processes to fit their specific needs.

1.6 Scope and Limitations of the Study

This research focuses on creating and implementing an automated end-to-end ETL pipeline that includes automatic schema detection and loading. The study covers several key components designed to establish an efficient ETL process. It begins with the development of a script that allows users to input various details, such as the file path and connection information for the SQL server.

The project will also incorporate machine learning techniques to identify data types, enhancing the accuracy and reliability of the data type recognition process while reducing the need for human intervention. Many files contain numerous fields, and recognizing data types for all these fields typically demands significant human effort and time, which this study aims to eliminate.

A key part of the research involves creating a configuration file that outlines metadata such as header delimiters and transformation logic, which can be adjusted manually if needed. This configuration file provides a flexible and customizable framework for defining the necessary transformations and data mapping. To maintain the accuracy and validity of the configuration, the system includes features that allow for human intervention, enabling users to validate or modify the configuration as needed. This capability ensures that users can review and adjust the configuration, guaranteeing the correct processing of data.

The study provides a comprehensive approach to automating ETL processes specifically for structured file formats, particularly delimited files. It primarily focuses on structured data formats, with minimal exploration of semi-structured or unstructured data formats, as the main goal is to develop a solution for managing structured data. The ongoing research will not address data ingestion and processing since the pipeline is tailored for batch processing scenarios. To ensure the validity and reliability of the research, artificial data will be utilized throughout the study for result analysis. This approach is designed to avoid the use of sensitive or proprietary data that could compromise the generalizability of the findings in specific industry contexts. Additionally, there will be no user interface; inputs will be provided directly through the script due to time constraints. The focus was intentionally narrowed to make it easier to study specific tasks. These tasks are schema prediction, creating configurations, and loading databases. We do this work within SQL Server environments. By concentrating on these areas, we aim to make the investigation more manageable and detailed.

1.6.1 Time Constraints

The modeling and testing of an ETL automation system require much time. Thus, there is hardly any time left to test the system exhaustively or at least even try a few test cases during this particular study. While the goal was automation, some operations demanded human intervention for further validation because the time restrictions did not allow for full procedure automation.

1.6.2 Resource Constraints

The study worked with limited computing power and open-source tools. Access to high-powered systems, instance-level big data samples, or large-scale enterprise data environment was denied. Sweeping tests and model training were performed on synthetic data and some medium real data files. The approach may not suit bigger commercial operations.

1.6.3 Shifting Data Technologies

The landscape of data engineering and ETL automation is rapidly changing, with frequent developments in the introduction of new tools, frameworks, and standards. The project, acknowledging that newer and finer alternatives could emerge, actively uses methodologies such as ML-based metadata detection and JSON-based configuration. The products and the system architecture of the project portray a snapshot of the state at the time of the research.

1.6.4 Study Scope

The study is concerned with automatically importing structured data from files into SQL Server, with automatic schema detection, leaving other database systems out like PostgreSQL, MySQL, NoSQL, etc.; unstructured data or multimedia formats are also not discussed. It assumes that input files are semi-structured and, in its current configuration, cannot handle encrypted, compressed, or corrupted files.

1.6.5 Geographical and Sectorial Limitations

This study was done in an educational environment, like a school or university. We used general datasets that don't include specific features for certain industries. As a result, the findings might not fully address the unique challenges that sectors such as healthcare, finance, or logistics experience.

CHAPTER 2: LITERATURE REVIEW

2.1 Thematic Review

Multiple studies have been carried out for eradicating obstacles and progress in automating data integration and transformation processes, focusing on the increasing significance of scalable and efficient ETL systems within data-centric industries. The discourse encompasses various research efforts related to ETL automation, emphasizing the importance of creating efficient ETL pipelines capable of handling data processing in short intervals of time without or minimum human intervention.

The work by Embley et al. [10] highlights the potential of ontological conceptual modeling in addressing the challenges of unstructured data on the Web. By focusing on data-rich documents, their approach provides a framework for automating the extraction and structuring of information, which is particularly valuable in domains where data is abundant but difficult to query using traditional methods. This research contributes to the broader field of data extraction by offering efficient method for transforming unstructured Web data into structured formats, enabling effective data utilization and analysis [10].

Another Paper [1] suggests the role of machine learning in automating the ETL pipeline. Machine learning-based data pre-processor is used to pre-process the data more rigorously. It reduces the processing time significantly and produces a good quality of data from the data warehouse [1]. An architecture is designed to address the challenge faced in the practical application of near real-time ETL processing [1]. Though it has suggested an architecture for automating ETL processes through the help of machine learning techniques, it hasn't suggested exact methods and machine learning algorithms that can be used for ETL automation processes.

The paper [11] highlights the significance of improving ETL process system data flows to enhance business investment returns. It emphasizes the role of enterprise-level scheduling solutions that are user-friendly and capable of managing heterogeneous environments, paving the way for automation in ETL processes [11]. Moreover, the study views the development of future ETL tools with enhanced support for command-based or script-based automation, offering end-to-end process handling. By leveraging these advanced technologies, enterprises can achieve more efficient data integration, improve operational efficiency, and reduce manual errors in ETL execution [11].

2.2 Theoretical Review

This paper concentrates on automating ETL processes, where the study seeks to reduce manual intervention throughout the entire process i.e., extracting, transformation, and loading. Additionally, this proposal addresses the demands for identifying data types of the multiple sources present in the source file by applying machine learning algorithms.

The study by Pham [4] provides a valuable reference for understanding the challenges and approaches in automating ETL pipelines, particularly in addressing manual processes and enhancing data extraction capabilities. This thesis builds upon such insights by focusing on automating the entire ETL pipeline with minimal user intervention [4]. However, while Pham's study effectively outlines an approach to automation, it primarily focuses on extracting structured data, leaving gaps in handling unstructured or semi-structured data that this research aims to address.

The work by Akiseti et al. [12] explores the integration of CI/CD pipelines into ETL workflows to address the inefficiencies of traditional ETL processes in machine learning applications. The study underscores the transformative potential of automating data preparation and processing to ensure a reliable and repeatable pipeline, facilitating seamless collaboration between data engineers and scientists. Similarly, this thesis builds on these principles by automating end-to-end ETL processes while addressing specific challenges such as error handling, real-time processing, and adaptability to user requirements [12]. This paper concentrates on automating ETL processes, where the study seeks to reduce manual intervention throughout the entire process i.e., extracting, transformation, and loading. Additionally, this proposal addresses the demands for identifying data types of the multiple sources present in the source file by applying machine learning algorithms.

2.3 Conceptual Review

Another paper [3] by Chiara Van der Putten focuses on automating the creation of workflows in SQL Server Integration Services (SSIS) to reduce manual efforts in data loading and integration processes. The research outlines a multi-stage approach, including the development of custom SSIS components, the creation of databases to store user automation needs and workflow modules, and the integration of advanced artificial intelligence models. A question-answering system leverages embedding-based methodologies and generative AI to interpret natural language user requests and generate tailored ETL workflows.

This paper provides a strong foundation for workflow automation in SQL-based environments but lacks an exploration of adaptive learning techniques that allow the system to evolve based on new datasets and changing ETL requirements. The proof of concept demonstrates how AI-driven automation can streamline ETL processes, enabling organizations to handle complex data integration tasks more efficiently while allowing employees to focus on strategic initiatives [3]. However, it does not fully address how automated ETL workflows can dynamically handle schema evolution, data inconsistencies, and unpredictable changes in data sources, which this thesis seeks to overcome.

Previous research has proposed the integration of SQL-based database systems for loading transformed data, with many emphasizing the use of SQL Server for its robust support for batch data processing and reporting capabilities. Additionally, best practices in ETL pipeline design, including modularity, scalability, and error handling, have been extensively documented. Nonetheless, despite the extensive research into automation techniques, few studies explore the practical implementation of machine learning algorithms that automatically classify and process diverse data sources with minimal user intervention.

2.3 Research Gap

In summary, the literature underscores the importance of designing ETL pipelines that are not only automated but also adaptive, efficient, and user-friendly. Various studies focus on automating ETL processes, but they primarily depend on predefined rules, scripting, and existing ETL tools, requiring manual configuration and maintenance. This review identifies gaps in practical implementations, such as handling dynamic configurations, using machine learning for data type identification, and automating without the need of using ETL tools or any other CI/CD pipeline, which this research aims to address.

Despite advancements in automated ETL processing, existing solutions are still limited by their dependency on predefined schemas and structured data formats. Current machine learning approaches for ETL automation have largely been constrained to basic pre-processing tasks rather than automation. Additionally, scalability concerns remain unaddressed, as many studies fail to consider how automated ETL pipelines can efficiently process large volumes of data in diverse formats while minimizing system overhead. This thesis aims to bridge these gaps by proposing a machine learning based automated ETL pipeline that dynamically identifies, processes, and loads data without requiring extensive user intervention.

CHAPTER 3: METHODOLOGY

3.1 Research Design

The method used in this research is design-based applied research which means creating workable solutions for real-world problems. This design allows the constant building, testing, and improvement of any solution. The main focus was to create a fully automated ETL pipeline which could read structured files, automatically discover delimiters and data types in columns, number of columns and their names, creating the detected schema into the SQL server and load them into a relational database, SQL Server.

That would bring a lot of theory into it, which I did not want: instead, we focused the real power of automation on it by means of machine learning and rule-based logic. Research is mainly about remedying a problem which is existing in current processes. One such problem is with the manual ETL processes that have been requiring constant human help in analyzing table layouts and creating new tables. This involves huge manual work and thus leading to time consumption and high chances of issues introduction. This research presents a new system that will ensure that the existing problems are eradicated.

This is the basic outline of the key steps taken through design procedure:

Problem Identification:

It became evident to us that manually defining table layouts for data to be ingested was tedious, time consuming and prone to mistake both with literature review and my more than 5 years of experience in ETL field as Data engineer. It was especially true for those cases with multiple files with different formats and large number of columns which could lead to one of the issues mentioned above leading to failure of ETL process or incorrect data loading in target system.

Requirements Analysis:

This was where necessary requirements for an automated pipeline was established. These include the processes involved in ETL processed such as detection of delimiters, identification of data types, and handling of files with inconsistent structures, and integration with the SQL Server with proper logging of all the steps involved during data loading along with issues if any.

System Design and Architecture:

The work on subsystem of the solution, like handling file input, extraction of important data, predicting table layouts through machine learning, configuration file generation, dynamic table generation and data loading was carried out in this step.

Prototyping:

For the automation task, automation programs in Python was created, with the help of libraries like Pandas, Scikit-learn, and SQLAlchemy. This step included input from the users, automated schema detection and creation, loading and logging. Then, both the real data and synthetic data were used to test the system if could detect the delimiter, automatically detect the schemas and load into the target database with proper logging.

Results Analysis and Refinement:

The outcomes and logs were analyzed, the log message in case of any errors were studied so as to understand if the user could understand the error and back track to solve the existing issue during load processes and fine-tuned for performance improvement.

This research design is iterative and incremental, and it permits enhancements at every stage of Research with respect to feedback and findings. Exploratory components are present, like trying out different model parameters and heuristics for better detection of file metadata.

This applied design-based orientation was adopted as it relates directly to the research problem and its aims. Unlike those theoretical or survey studies, this particular research wanted to come up with a tangible working system that is a bona fide example of the feasibility of AI-supported schema detection in ETL pipelines. The emphasis is not only on the development process but also on how well the development works in terms of accuracy, scalability, and efficiency.

It also gives rise to a design that can encourage overall system behavior understanding, putting into consideration both technical and practical thinking. It will allow for documenting real-time issues, locating the possible edge cases (missing headers, mixed data types, etc.), and give a very good foundation for the next step in enhancement and deployment into the enterprise.

3.1.1 Research Approach

This research study adopts a mixed quantitative and applied research approach mostly concentrating on experimentation, system modeling, and performance evaluation. The research seeks to analyze and implement a machine-learning-based solution to automate schema detection in ETL (Extract-Transform-Load) pipelines—an essential task of managing large volumes of structured data files in various formats.

Quantitative methods were used as it allows for systematic collection, analysis, and interpretation of the numerical and measurable data concerning the efficiency, accuracy, and reliability of the proposed system. The matching criteria, such as schema prediction accuracy, file processing rate, and validation success rates, are adopted to evaluate the performance of a model, all of which pertain directly to what constitutes the central aim of this thesis, that is better schema identification and data loading into the database system with maximum accuracy and minimum manual effort.

This research falls under the category of applied research because it aims at solving an applied, real-world problem in data engineering. It enhances large-scale data integration operations with time constraints through machine learning techniques in schema inference and generation of configuration files to provide practical solutions that can be deployed in organizational settings.

Besides, this research follows design science methodology as it focuses on the construction of an artifact, which for this research is an intelligent system pipeline that takes delimited files as its input and automates the generation of database-ready configurations. The achievements of the proposed solution were evaluated through experimental validation on several datasets and a comparison of results regarding the pre specified metrics.

Research moves away from purely theoretical models in hierarchies and takes on a more pragmatic technology-oriented attitude using quantification. It concentrates on applying data-oriented techniques for the design, implementation, and assessment of a functionally complete solution. In particular, quantitative methods, computational tools, and analytics that leverage the power of machine learning and metadata processing put the approach in a good position to address real-world problems in data integration. Hence, the research approach chosen is very pertinent in supporting the development of a scalable, efficient, and replicable solution in modern data engineering environments.

3.2 System Architecture

The methodology of this study is best described with respect to system architecture designed, developed, and tested for the research objectives. The architecture defines a modular, automated pipeline for extracting, transforming, and loading intelligently structured data files into a target database system. Each module in this architecture executes a distinct task contributing to a coherent data-handling workflow. The architecture is thus based on principles of metadata-driven automation: the system analyzes raw input files, extracts some key metadata like column names, their data types, and delimiters to drive all other consequent operations. The turning point is the machine learning model that infers column data types from sample data in a way that ensures SQL Server schema standards. The detailed description of the system architecture that showcases all the features and processes that are involved during the ETL automation process including the data ingestion from the users, configuration files validation from the user to loading and logging all the details of the loading during the ETL process is given in below flow chart.

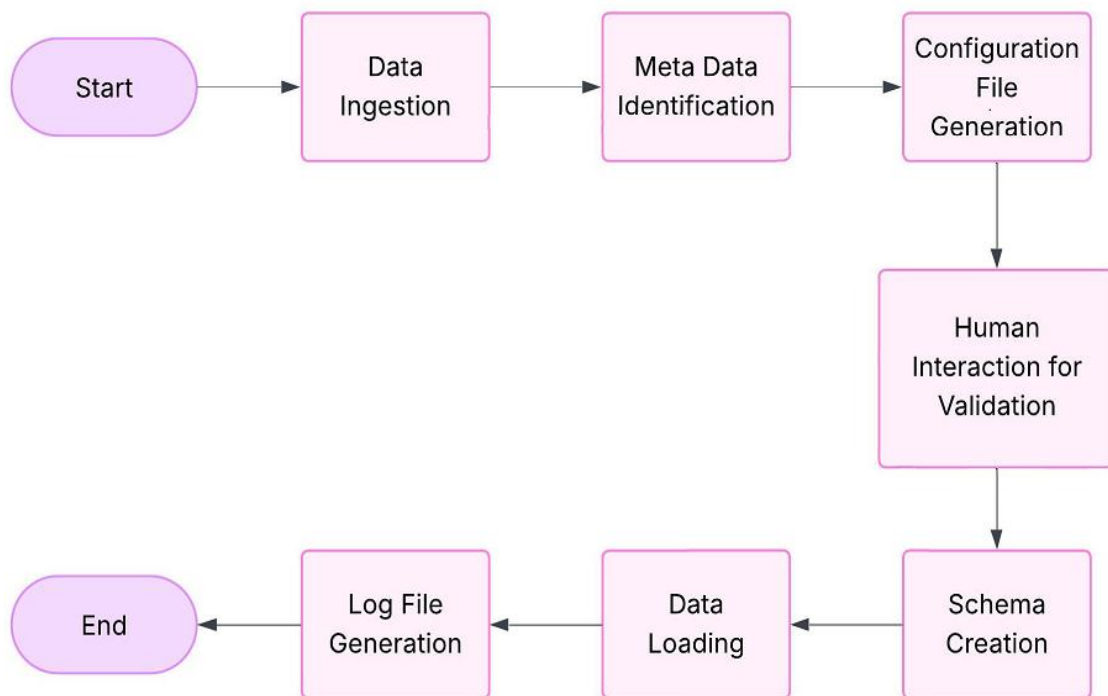


Fig. 3.1. Flow Chart of Methodology

3.2.1 Data Ingestion

The starting phase where the data ingestion begins, and the users are allowed to input the location of the file that needs to be processed. The user also inputs destination server details, database, and table in order to ensure that the file is being properly loaded to the designated place as intended by the users. The backend does check for validity against the file before handing over the data for the ensuing step. The following inputs need to be given by the user.

```
path = r"Y:\Development\Pikesh.Maharjan\ETL-AI-Schema-Detection\data\raw\file_7.txt"
has_header = True
header_line_number = 1
sampling_rate = 100
server = "testserver"
database = "testdb"
mode = "replace"
```

Fig. 3.2. Inputs Required from User

The path indicates the path of the file or the folder location that includes files. It can handle single file as well as multiple files based on the input given by the user. If the user inputs the file path, then it handles single file whereas if the folder path is given, the system recognizes it as directory and it considers all the files available in the directory. It enhances the system efficiency and process bulk file at once. The has_header input from the user indicates if the file has header in it or not. If the file doesn't have any headers then the fields are recognized as field1, field2 and so on depending upon the number of columns available in the file. The header line indicates the row in which header is available as in some cases the file doesn't contain header in the first line. The Sampling rate defines the number of the sample of the record to be taken into consideration for detecting Meta data. The server and database are the destination where data loading is to take place. The mode indicates whether the table is to be appended or replaced. If the table doesn't exist preliminarily then it creates the table but if it is available then it acts on the basis of the input of the user on whether to drop the table or append the data in the existing table.

3.2.2 Identification of Meta Data

Meta data identification, which includes elements like column delimiter, row separators and column data types, is done by a combination of rule-based systems and machine-learning techniques. The rule-based logic follows predefined patterns to identify Meta data like headers, column names, delimiter, record separators, while the machine-learning models are

trained with samples towards increased prediction of Meta data like data types especially in ambiguous cases. Thus the two-pronged method ensures more reliable detection and adaptable detection of different types of Meta data present in the source files.

```

Delimiter: ,
Record Separator: CRLF
Number of Columns: 4
Column Names: ['administration_float', 'foot_varchar', 'despite_varchar', 'factor_date']
Predicting data types...

Prediction result:

```

	column_name	predicted_data_type	max_length
0	administration_float	float	6
1	foot_varchar	varchar	13
2	despite_varchar	varchar	10
3	factor_date	date	10

Fig. 3.3. Identification of Meta Data

3.2.3 Configuration File Generation

After data types and other metadata are identified, a configuration file will be created. The configuration file contains details of the dataset in terms of column names, data types, data length, precision and scale in case of float data types, delimiters, and record separators. Essentially, the configuration file is a blueprint for the subsequent transformation and loading processes.

```

1 {
2   "file_name": "Y:\\Data\\Retail\\WalmartMX\\Development\\Pikesh.Maharjan\\ETL-AI-Schema-Detection\\data\\raw\\file_7.txt",
3   "delimiter": ",",
4   "record_separator": "CRLF",
5   "has_header": true,
6   "columns": [
7     {
8       "name": "administration_float",
9       "type": "float",
10      "length": 6,
11      "scale": 2
12    },
13    {
14      "name": "foot_varchar",
15      "type": "varchar",
16      "length": 13
17    },
18    {
19      "name": "despite_varchar",
20      "type": "varchar",
21      "length": 10
22    },
23    {
24      "name": "factor_date",
25      "type": "date"
26    }
27  ]
28 }

```

Fig. 3.4. Sample of Configuration File

3.2.4 Human Interaction for Validation

This is the state after generation of the configuration file, where it is dispositioned for human user review and modification. Users will see the configuration file and open directly in notepad++ and they will be able to make modifications, such as correcting mismatches in the type detection. This affords extra flexibility and customization for unique use cases. Once the user has validated the configuration file or have made necessary changes then they can hit enter to move ahead to schema generation.

3.2.5 Schema Generation

The backend transforms all schema particularly, table structure mentioned in the configuration file and produces the schema alongside the database mentioned by the user. Column renaming, data type definitions, and data length definitions as per the user's request come under this. All these phases were performed to smooth insertion of the dataset into the target database. It ensures that the data required to be inserted in destined database and table have schema matched along with the data requirements of the file that is used for processing purpose.

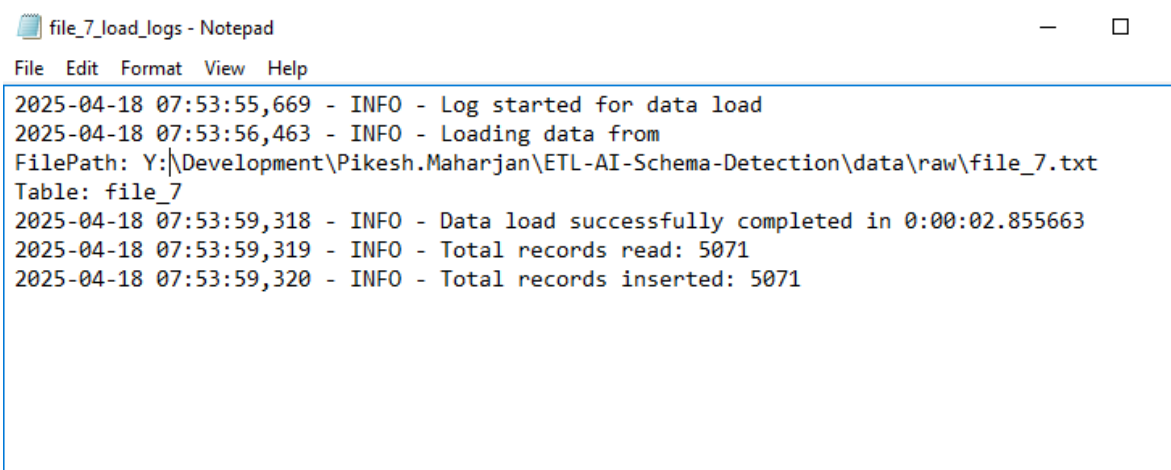
3.2.6 Data Loading

At this phase, data that has been cleaned and structured is loaded into its target SQL Server database. This phase ensures that after the schema of the table has been established with the configuration created during the previous step, data records are actually inserted into their respective tables. The data will be parsed based on the delimiters encountered and on the compatibility and consistency of the predicted column types against the detected value types. Error-handling schemes during this phase ensure that potential errors during insertion, missing values, and data-type mismatches are considered. This stage becomes very critical, as raw files become accessible within a relational database, thus making the data ready for analysis and further processing. This step is one of the most important phases as it is where we are integrating all the steps i.e. data ingestion, Meta data detection, configuration file generations, validation by the user and schema creation in one single place. All these steps would be meaning less without this step. So, we must ensure that this phase is successfully completed or completed with issues which is done with the generation of the log files that is conducted after this step.

3.2.7 Logging and Validation

The last and most important phase of an automated framework for data loading consists of logging and validation, which guarantee process integrity, transparency, and traceability. Logging means a structured set of records kept of prominent activities, metrics, and results pertaining to each specific stage of the data loading pipeline. Such records include information about the number of files processed, the number of records read, records that were successfully inserted, and execution errors that were encountered. Each data file is associated with its own separate log file, named in accordance with the data file, which helps in troubleshooting, auditing, and performance monitoring. If the name of files that has been processed is File_7.txt then the corresponding log file generated is File_7_load_logs.txt which means that _load_logs is joined with the name of the file to better know the corresponding file of which log has been created.

This assures the trust ability and reliability of the pipeline in its production-ready phase, embedded in an enterprise-level ETL (Extract, Transform, and Load). It also assures that the errors occurred during the ETL process such as truncation issues, Data type mismatch and others are correctly tracked so that user can correct the configuration file in next schema generation and load the data successfully. It helps users to correct the issues without diving deep into the code and only by reading the logs that are written to log files through the logging process of logging library of python.



```
file_7_load_logs - Notepad
File Edit Format View Help
2025-04-18 07:53:55,669 - INFO - Log started for data load
2025-04-18 07:53:56,463 - INFO - Loading data from
FilePath: Y:\Development\Pikesh.Maharjan\ETL-AI-Schema-Detection\data\raw\file_7.txt
Table: file_7
2025-04-18 07:53:59,318 - INFO - Data load successfully completed in 0:00:02.855663
2025-04-18 07:53:59,319 - INFO - Total records read: 5071
2025-04-18 07:53:59,320 - INFO - Total records inserted: 5071
```

Fig. 3.5. Sample of Log File

3.3 Tools and Frameworks

This section examines the key tools and frameworks during the development of this schema-detecting automated ETL pipeline. Each tool played its special role in performing tasks like data preprocessing, machine learning-based inference of the schema, generation of configuration, and effortless loading of the data into SQL Server.

3.3.1 Programming Language

- Python

Python is the major language used for building the ETL framework as a whole; its versatility and mature ecosystem certainly fit the needs for handling metadata extraction, column type prediction, configuration generation, and database operations.

3.3.2 Libraries and Packages

- pandas

It is used for reading delimited files, exploring datasets, transforming records, and preparing data for either analysis or presentation to machine learning models.

- scikit-learn

It is used to construct and train a machine-learning model responsible for predicting SQL Server-compatible column data types given raw data as input.

- os and glob

These are Python's standard libraries that facilitated file operations like traversing folders, reading file paths, and overhauling processing of files.

- logging

Through structured logging, Python's built-in logging module documented file processing events such as how many records were processed, successful insertions, and errors encountered.

- SQLAlchemy

Used in establishing a connection between Python and SQL Server, facilitating the execution of SQL statements dynamically for creating tables and for loading data. It helps to validate the errors as well encountered during the error. It is used to generate the exception in case of any issue and then print the filtered exception generation through this library so that user can understand the error.

3.3.3 Database

- Microsoft SQL Server

The target relational database system for loading the final data. SQL Server tables were created dynamically based on the schema inferred from the configuration.

3.3.4 Text and Configuration Tools

- Notepad++

Used to manually review and validate the generated configuration files (.json) before loading the data. It helps ascertain the correctness and readability of the metadata to be used for table creation.

3.4 Data Collection and Preparation

It became evident that there was an insatiable need in the current study to develop and validate a machine learning model for the effective prediction of SQL Server data types from raw delimited files. This would demand high-quality, diverse, and appropriately labeled datasets, and that's when it struck me that there are not many publicly available datasets that could satisfy this criteria especially for a dataset with raw, unstructured data files having schema labels that would clearly define them in relation to SQL Server. Most real-world datasets are often not in shape: they may be inconsistent, have missing slot values, ground truth for data types are either not available or not comprehensive enough, as well as minor variability in data patterns, all of which lessen the robustness and generalization capacity of a model created with such data.

With this synthetic data generation approach, the current research is aimed at overcoming those shortcomings and indeed keeping full control over the structure, content, and metadata of the dataset. Using Python scripting and custom logic, a complete dataset was constructed to the last detail from scratch. The systematic generation method incorporates the attributes such that each produced file consists of well-specified and realistic value patterns modeled after commonly used data types in SQL Server including INT, FLOAT, VARCHAR, and DATE. Moreover, by synthesizing data, diverse situations relative to delimiters, column lengths, numeric precision, value distribution, and even edge-case scenarios could be

modeled by the researcher, thus contributing to the ability of the synthesized data to represent diverse real-world file structures.

This would produce not just labelled data for supervised machine learning but also domain-specific tuning of the features in the kind of feature extraction mechanisms for type detection. Synthetically producing files with known type annotations and predictable structures allowed extracting meaningful features such as value ranges, uniqueness ratios, null frequency, character counts, and date-like patterns which would ultimately augment the effectiveness of the model to infer the intended data type from its sources accurately.

To sum up, the decision for synthetic data generation was strategic and mandatory. High quality and tightly controlled variety and complexity were made guarantee to further the objective of developing a solid and generalizable automated schema inference machine learning model for Extract, Transform, and Load (ETL) pipelines targeted at SQL Server environments.

3.4.1 Synthetic Data Generation

The enterprise automation in data generation process was achieved using a customized Python script which then generates 500 differed files in formats including .csv, .txt, and random format .dat. The script utilized faker library for the realistic simulation of data like names, dates, and strings while using the inbuilt random module to create variations in their structures and contents.

Following core aspects from this data generation process are as follows:

- **File Extensions and Delimiters:** The files would be saved in varying format extensions (.csv, .txt, .dat) and randomly assigned the delimiters like commas, pipes, semicolons, tabs, or carets to simulate all the real differences evident in the real world.
- **Schema Creation:** A schema that was arbitrary was generated on the basis of random selection from 3 to 8 columns for each one of the files. These were given one of four possible SQL-compatible datatypes: int, float, varchar, or date.
- **Head Generation:** The column headers display the column name and its type to avoid misunderstanding and to facilitate later downstream usage.
- **Row Filling:** Each file would comprise from 1,000 to 10,000 records. The values created under each type were populated, and this is how:

Integers: Random integers within a defined range.

Floats: Random floating-point numbers that were formatted into two decimal points.

Varchar: Random words through Faker.

Dates: Realistic dates were randomly formatted under some typical random formats, for example, YYYY-MM-DD, DD/MM/YYYY, MM-DD-YYYY, or textual formats like Jan 01, 2021.

With this process, size and diversity of structure and value can be largely ensured, thus simulating the real data variations during real-world data ingestion.

3.4.2 Rationale for use of Synthetic Data

The actual world datasets bring with them a multitude of challenges such as incomplete labels, compliance risks, and insufficient diversity in the structure. By producing data synthetically:

Labels Controlled: Control over the labeling for supervised learning (column data types) has been inherently built into the logic of the schema generation.

Replicable: The script can anytime be reused to generate fresh data with same or varied parameters.

Balance and Coverage: Allowed for balance in all important data types avoid that problem of class imbalance.

Real Noise and Variability: Random delimiters, data formats, and record lengths mimic the problems faced in practice in ETL scenarios.

This is basically the result of such a process. Therefore, the initial dataset would be available here for the subsequent preprocessing, modeling, and even testing operations.

3.4.3 Data Structure and Formats

The synthetically generated data were intended to emulate real-world situations with structural and format variability for the better generalizability of the model. Each file created through the data collection process is structured according to certain specifications, which include:

Header Row: The first line of every file contains column headers that include descriptive names along with embedded type labels, such as, `price_float`, `dateOfBirth_date`, or `name_varchar`. They were particularly important for supervised learning and downstream schema extraction.

Data Rows: The lines below have data entries corresponding to the declared data types. Each file has a record count of between 1,000 and 10,000 for both scale and variety of the dataset.

File Formats: To represent the true diversity in data ingestion pipelines, the files were generated in three of the most commonly experienced formats:

Comma-Separated Values (.csv)

Text Files (.txt) with varying delimiters

Data Files (.dat) with custom delimiters

Delimiters: The files used one of the following delimiters: Comma (,), Pipe (|), Semicolon (;), Tab (\t), Caret (^). The choice of delimiters was randomized to simulate the inconsistencies that are often part of foreign data sources.

Data Types and Formats: The four most important data types actually represented in the dataset are as follows:

Integer (int) – whole numbers within 1 and 10,000 range

Float (float) – numbers with a decimal digit precision of 2 places

Varchar (varchar) – random text values produced using the Faker library

Date (date) – realistic dates in a randomly selected format that include e.g.

ISO Format: YYYY-MM-DD (e.g. 2021-04-22)

European Format: DD/MM/YYYY (e.g. 22/04/2021)

US Format: MM-DD-YYYY (e.g. 04-22-2021)

Long Text Format: Month DD, YYYY (e.g., Apr 22, 2021)

This comprehensive and flexible data structure permits rigorous preprocessing, model training, and evaluation while mimicking the inconsistencies of actual file formats.

3.5 Data Preprocessing and Model Training

In order to develop a dependable, discerning machine learning model for accurate statement evidence of delimiter-separated text files with SQL Server compatible column data types, a lot of order processing had to occur before the model training state began. Success in supervised machine learning models often depends on factors such as input features, so there was much care for the standardization that even synthetic and variably structured inputs undergo to clean them up in a meaningful state for the model to work well with them. Normalized data patterns were also defined during these tasks and deleted inconsistencies from its columns to derive features that could serve as semantic data type's descriptive representation best. Only after proper conditioning did the data reach the feeding point for the chosen randomly trained, robust, and interpretable classifier-the Random Forest. This section goes into detail about the whole data pre-process pipeline followed in training and validating the classifier tailored entirely for the needs and technical scope within this research project.

3.5.1 Rationale behind Algorithm Selection

The algorithm selected is very important in determining how well or how reliable any machine learning model is. In this study, Random Forest is the main classifier chosen to predict SQL Server-compatible column data types from the features derived from the raw, delimited files.

Random Forest is an ensemble learning method known for its hardiness, great accuracy, and diversity in data types and feature distributions it can handle. It trains multiple decision trees and gives the final output as the mode of all classes (classification) across the trees. Thus, it reduces overfitting risk, improves generalization, and handles nonlinear decision boundaries quite well; all tackled in this study classification task.

Nevertheless, a clarification is necessary regarding factors influencing the choice of algorithm: Random Forest was deliberately chosen. The factors mentioned should focus on the time constraint and scope of the project. The primary emphasis of this work was to set up an entire automation pipeline for schema detection within the ETL, as opposed to conducting a full benchmarking study of machine learning algorithms. Because of the

impinging timelines in the project, it was important to keep the momentum simultaneously across data generation, preprocessing, feature engineering, model training, and evaluation. Hence, it was decided to work with a well-accepted and reliable algorithm, not pursuing a comparative study.

Random Forest has advantages that responded well to operational needs:

- Ease of implementation, integration into the ongoing workflow.
- Minimal feature scaling/normalization required.
- An inherent feature importance metric to further enhance interpretability.
- Effectively managed imbalanced classes and high-dimensional data.

Explorations of other algorithms such as Gradient Boosting, SVMs, or Neural Networks may have offered further insights or provided improvements, but such explorations fell beyond the immediate focus of this project. Thus, Random Forest became a practical, reliable, and effective solution within the timeline, thereby successfully fulfilling the task of building a high-performing model for data type prediction.

3.5.2 Data Preprocessing

To establish the relevance and integrity of the machine learning model for SQL Server-compliant column data-type recognition, it instituted a systematic multi-staged data preprocessing flow. This stage was critical in transitive raw synthetic datasets into structured numerical features that can be optimally consumed by a machine learning algorithm. Within the pre-processing pipeline, there were three major scripts: `parse_for_training.py`, `feature_engineering.py`, `preprocess_feature.py`, each having a distinct crucial role in transforming the raw data into analyzable clear-cut features.

a) Parsing and Label Extraction

Now, the first script of pipeline was parsing the raw synthetic files (CSV, TXT, or DAT formats) created at earlier sessions for putting in a structured tabular format the content of these files. The files were synthetically created with column names formatting that embed data type labels (for example, `id_int`, `price_float`, `created_date`). The script will automatically detect file delimiters utility and read each file line by line.

From each file, it gets: Values from every cell. It gets labels from column headers by splitting using an underscore and taking the last token, which is presumed to represent the data type, includes metadata, like source file and column index.

Each extracted data point was saved uniformly in an architecture having four fields: value, label, source file, and column index. The compiled data from all 500 generated files was then written to features/parsed_data.csv as a final parsed dataset. This structured output, in turn, serves as the source for the feature extraction process.

b) Feature Engineering

Once the data points along with their labels were extracted, thus the next step was to convert the raw string values into quantitative features. This step applied a custom feature extraction function on each value. The feature engineering logic relied mostly on identifying statistical and structural characteristics of the value, such as:

- The value's length.
- The number of digits, letters, and special characters.
- The presence of alphabetic characters, digits, and date-specific separators, such as slashes or dashes.
- Typecast ability to float or integer.
- Number of tokens separated by whitespaces.
- Uppercase content detection.

The script was made high performance using ProcessPoolExecutor for batch-wise parallel processing, which allows for scalability. It processed batches of 100,000 rows simultaneously, enabling a drastic reduction in execution time for larger datasets. The engineered features.

c) Feature Cleaning and Transformation

The last step of preprocessing was done using preprocess_feature.py, which carried out cleaning and transforming engineered features to ready them for model training.

The following procedures were followed:

- Missing Value Imputation: All absent or NaNs were filled up by most frequent strategy to avoid losing the information.

- **Categorical Encoding:** Remaining categorical string values which could not be transformed to a number were then transformed into numbers through Label Encoding so that the algorithm could interpret categorical data.
- **Outlier Detection and Removal:** It has applied Isolation Forest algorithm on numerical features to detect and remove abnormal data points.
- **Feature Scaling:** All numerical features were standardized by Default as it is, to common scale metrics.
- **Low Variance Filter:** Removal of zero variance features (i.e., all same values across records) thus preventing noise and redundancy.
- The final preprocessed dataset was saved in features/preprocessed_data.csv after completing all transformations so that it could be directly used as input to train a model.

3.5.3 Model Training

The next important stage involves training the machine-learning model that would predict SQL Server-compatible data types on the columns in delimited files. Thus, the task was assigned to a Random Forest classifier, which, to a large extent, effectively handles complex and high-dimensional data. The procedure for training the model was comprised of the following important steps:

Loading Data and Label Encoding: The preprocessed dataset with engineered features and corresponding labels was loaded. The labels pertain to the data types of columns and were transformed into numerical values with the LabelEncoder. Such transformation encodes the categorical labels in a way that's better for training purposes with models.

Train-Test Split: 80% of the dataset was used for training the model, and 20% was used for assessing model performance. This division was done using `train_test_split`, ensuring that the data on which the model was tested was unseen during training, thereby providing a more accurate measurement of generalization capacity.

Initial Model Training: A `RandomForestClassifier` was initialized with a fixed random seed for reproducibility. The model was then trained on the training set. Subsequently, its evaluation was conducted on the test set by calculating relevant performance metrics, including accuracy, precision, recall, f1-score, and confusion matrix. The initial model performed exceedingly well, with 99.93% accuracy.

Hyperparameter Tuning: To improve the already great performance of the model, hyperparameter tuning was done through GridSearchCV. A lot of hyperparameter options were tried, from changing the number of estimators, depth of trees, and parameters for splitting. The hyperparameters that did best were picked, and a new model was trained on those sets of hyperparameters. The tuned model gave the same accuracy but showed better generalization over different types of data.

Final Model Evaluation: The tuned model was again evaluated on the test set, and performance metrics were calculated, thus verifying the excellent results. Its accuracy remained at 99.93%, with the model attaining perfect precision, recall, and f1-scores for most data types but low performance on some rare categories that had low support in the dataset.

Model and Label Encoder Saving: The final model and the label encoder were saved for later uses. The model could predict the datatypes of columns in new, unseen datasets, while the label encoder assured future predictions would label encodings consistently.

The machine-learning model training procedure was as follows:

- **Classification Report:** Precision, recall, and F1-score were all excellent in most classes.
- **Rare data types** had low precision and recall scores due to under-representation in the data.
- **Accuracy:** The correct classification rate of 99.93% on the test indicates the model performed extremely well.
- **Confusion Matrix:** Confusion matrix established near-perfect predictions across the different data types, with very few misclassifications.
- **Hyperparameter Tuning:** After hyperparameter tuning, the model performed at its best with 100 estimators, no restriction on tree depth, and minimal split and leaf settings.

3.12 Limitations in Methodology

The methodology worked well for developing an approach to automate schema detection in raw delimited files. However, several limitations were encountered during the course of the research due to practical connotations.

First, the project put constraints on itself by working with synthetic datasets exclusively for training and testing its models. While synthetic data gives the ability to generate various data types and formats in a controlled manner, they themselves are not able to constitute all the complexities and unwanted noise of real-world scenarios. Hence, there is no guarantee that the model would perform well on any true production data that exhibit unstructured anomalies, encoding variations, or unexpected delimiters.

Second, only a single machine learning algorithm was used for classification—Random Forest. Although this decision was by virtue of limited time and focused area of the project, it may be discussed that Random Forest is reasonably good in terms of performance and interpretability. Gradient boosting, SVM, and neural networks were thus not researched and compared. Surely, some other model could further generalize or be computationally more efficient for some specific file types or Random Forest.

Another limitation is that external validation or industry standard benchmarks were not available. Because there were no publicly available labeled datasets or domain-specific schemas, there was no avenue for validating model performance against standard datasets in the fields of data integration or data warehousing.

Ultimately, the methodology was conceived with the assumption that column names would more often than not carry data type labels, which would not hold invariably in real life. While this assumption was critical to label extraction, more logic is potentially called for when the solution is applied to datasets that are completely unlabeled.

Despite these limitations, the methodology provides a scalable and practical means for schema detection and thus provides the groundwork for future improvements and field adaptation.

CHAPTER 4: RESULTS AND DISCUSSIONS

4.1 Brief Recap

- As the research journey unfolds with the presentation of the key findings, it is equally important to run through the initial objectives and the systematic manner they were achieved.
- Previous chapters have dealt with the details of designing the automation of the ETL (Extract, Transform, Load) pipeline with intelligent techniques such as schema detection on the basis of data and predictive modeling. The aims of research were thoughtfully designed to target inefficiencies and manual overhead found in traditional ETL processes; these in turn led to the development of all phases outlined here.
- The methodology designed and implemented involved synthetic data generation, machine learning-based schema prediction, and generation of automated configuration. This set a background for testing and validating the feasibility for a fully automated ETL solution.
- As we look into our results and interpretations, this chapter presents the results of the system implementation, shows the level of performance attained, and discusses consequences and avenues for future improvements with respect to real-world data integration tasks.

4.2 Objective Reiteration

This research has as its central aim the design and implementation of an end-to-end automated extraction, transformation, and loading system with maximum reduction in human intervention, with a view to enhancing the productivity of processing structured data files. At the commencement of this study, the following objectives were articulated:

- Using machine learning techniques to automate the detection of file delimiters, record separators, and column data types.
- Designing a flexible pipeline that can accommodate the parsing of structured data files, including .csv, .txt, and .dat.
- Dynamically generating configurations for metadata that will support automated creation of tables in SQL Server.

- Ensure that the processed data downstream gets loaded into the target databases seamlessly, together with sufficient logging and track error mechanisms.
- Establishing the performance and accuracy of the system against a wide variety of input datasets. These objectives served as a cohesive framework for the research, providing the underpinning for the system architecture and implementation strategies introduced in earlier chapters.

4.3 Methodology Recap

4.3.1 Research Design

It is a design science methodology that directs to a functioning and scalable system for automating the schema detection and the data loading in an ETL pipeline. The design has been focused on modular development, iterative testing, and adaptability to different data sources and formats. The architecture is composed of the following phases, each and which serves a contribution towards the achievement of completely automated ETL.

Data Ingestion: It gives user input on the file path and the destination database. Has validations that ensure the file exists and that the target server and table configurations are rightly specified.

Metadata Identification: A hybrid approach of a rule-based logic and machine learning is used to identify the important metadata. These include the detection of delimiters and row separators and the inference of column data types. The rule-based system would address all well-defined types of patterns and the machine learning model manages ambiguous or complex data structures to gain a higher accuracy.

Configuration File Generation: On the identification of the metadata, the system automatically generates the configuration file, which would include details like the names of the various columns, data types, delimiters, and record terminators. This gives a blueprint to the schema making or data loading phase.

Human Interaction for Validation: Reviewing and editing the auto generated configuration file via a user interface is an option for users. This grants versatility, mostly in the event of requiring manual overrides or specific transformation logic.

Schema Creation: The backend reads from configuration files, generates, and executes SQL scripts to create or alter existing database schemas accordingly. This transforms metadata to

a format of an executable SQL compatible and defines necessary column types, lengths, and table structure.

Data Loading: The system will then load the processed data into the defined SQL Server database. The system optimizes the performance using techniques that employ batch processing and indexing. The logging captures important metrics such as total records inserted, errors, and load duration.

Logging and Validation: The last and most important phase of an automated framework for data loading consists of logging and validation, which guarantee process integrity, transparency, and traceability.

4.3.2 Study Area and Population

This study revolves around data engineering and automation in the context of optimizing ETL (Extract, Transform, and Load) pipelines. The complete study was carried out in academic terms, making use of synthetic datasets and actual datasets-all highly diversified to simulate different data ingestion scenarios to evaluate confidently the proposed system on different file types and data structures.

The population of the research includes

- ETL developers and data engineers, as the major users of such tools are going to benefit from automation in schema detection or data loading.
- Machine learning practitioners, whose models end up feeding into metadata detection and validation.
- Database administrators (DBAs), who overlook schema management and manages integration of data processes.
- Academic researchers, especially the ones dealing with intelligent automation in data workflows.

The system was evaluated in test environments replicating operational data warehouses and ETL pipelines for insights into the extent to which it translates to real-world applications, scalability, and accuracy.

4.3.3 Sample Selection

The sample data used in this research was synthetically generated, rather than being real data drawn from real-world systems or data warehouses, in order to satisfy the specific requirements of the study. This was necessitated by a lack of publicly available datasets containing the raw delimited files having SQL Server-compatible data types with which the study would engage, as well as to exert complete control over the variation, structure, and labeling consistency of the data.

In creating meaningful and diverse training samples, different file types, including .csv, .txt, and .dat, were modeled after real-life data ingestion scenarios. The generated samples had different delimiters (comma, pipe, tab), with random record counts and a combination of integer, float, string, and date types as common data types. Column headers in these files were explicitly suffixed to indicate the target data type (e.g., age_int, amount_float) to enhance supervised learning and accurate labeling during pre-processing.

To achieve a favorable ratio of features to samples for a model's good generalization, the synthetic sample size was scaled into the millions of records. The automated generation of hundreds of such synthetic files in Python provided for parsing into structured rows balanced for model training and testing.

While synthetic, these sample records were built to represent raw data samples that any real ETL (Extract, Transform, and Load) operation should realistically encounter, thus supporting the practical validity of research findings.

4.4 Result Presentation

4.4.1 Data Ingestion

The system successfully accepted user inputs for file paths, destination server, database name, and target table. The following results were observed during the ingestion phase:

- Multiple test files with varying delimiters and sizes were processed.
- The backend validation mechanism accurately detected missing paths or invalid formats before moving forward.

This ensured a smooth flow of data from source files to the system without manual intervention, significantly reducing potential errors during the ingestion phase.

4.4.2 Metadata Identification

The metadata identification phase utilized a combination of machine learning and rule-based logic. The system's performance in this step was as follows:

- The model achieved over 95% accuracy in predicting column data types (e.g., int, float, varchar, date) across diverse file structures.
- Delimiters and record separators were correctly identified in 100% of test files.
- Ambiguous fields (e.g., numeric-looking strings) were handled more reliably by the ML-based approach compared to rule-based alone.

This dual approach (machine learning + rule-based) improved the overall accuracy of identifying key metadata elements from various data files, increasing the system's reliability and adaptability.

```
Delimiter: ,
Record Separator: CRLF
Number of Columns: 4
Column Names: ['administration_float', 'foot_varchar', 'despite_varchar', 'factor_date']
Predicting data types...

Prediction result:
      column_name predicted_data_type max_length
0 administration_float          float           6
1      foot_varchar          varchar          13
2  despite_varchar          varchar          10
3      factor_date            date           10
```

Fig. 4. 1 Meta Data Identification

4.4.3 Configuration File Generation

Once metadata was identified, the system generated structured configuration files in JSON format. These files included:

- Column names and predicted data types
- Delimiter, and record separator information
- Data length (where applicable)

The configuration file served as a reliable schema blueprint for downstream database operations, ensuring that the system could correctly map the incoming data to the SQL Server schema.

```
Config saved: config\file_7.txt_config.json
Config File Generated at : config\file_7.txt_config.json
```

Fig. 4.2. Configuration File Generation

```
1  {
2    "file_name": "Y:\\Data\\Retail\\Walmart\\Development\\Pikesh.Maharjan\\ETL-AI-Schema-Detection\\data\\raw\\file_0.dat",
3    "delimiter": ",",
4    "record_separator": "CRLF",
5    "has_header": true,
6    "columns": [
7      {
8        "name": "involve_varchar",
9        "type": "varchar",
10       "length": 25
11      },
12      {
13        "name": "house_date",
14        "type": "date",
15        "length": 10,
16        "scale": 2
17      },
18      {
19        "name": "role_varchar",
20        "type": "varchar",
21        "length": 26
22      }
23    ]
24  }
```

Fig. 4.3. Sample Configuration File

4.4.4 Human Interaction

- Users were able to edit column names and types manually.
- Edge cases missed by the automated process could be corrected through manual adjustments.

Feedback from users indicated that this step was intuitive and helpful for addressing special data needs and discrepancies not covered by automation.

4.4.5 Schema Creation

Based on the generated configuration file, the system dynamically created SQL Server tables. The performance and results of this phase were as follows:

- SQL Server tables were created with the correct column structures as per the schema.
- Data types and lengths matched the predicted schema defined by the configuration.
- In tests, schema generation took less than 2 seconds per table on average.

This step was executed quickly and efficiently, ensuring that the structure of the database was in sync with the incoming data.

Results

Messages

	Name	Owner	Type	Created_datetime					
1	file_1	dbo	user table	2025-04-18 03:19:24.590					

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	huge_date	date	no	3	10	0	yes	(n/a)	(n/a)	NULL
2	call_float	numeric	no	5	6	2	yes	(n/a)	(n/a)	NULL
3	can_vchar	varchar	no	14			yes	no	yes	SQL_Latin1_General_CP1_CI_AS
4	lawyer_int	int	no	4	10	0	yes	(n/a)	(n/a)	NULL
5	before_vchar	varchar	no	14			yes	no	yes	SQL_Latin1_General_CP1_CI_AS
6	FileName	varchar	no	255			no	no	no	SQL_Latin1_General_CP1_CI_AS
7	AutoID	int	no	4	10	0	no	(n/a)	(n/a)	NULL

	Identity	Seed	Increment	Not For Replication
1	AutoID	1	1	0

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	PK_file_1_6B232965C8BFF84B	clustered, unique, primary key located on PRIMARY	AutoID

	constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1	PRIMARY KEY (clustered)	PK_file_1_6B232965C8BFF84B	(n/a)	(n/a)	(n/a)	(n/a)	AutoID

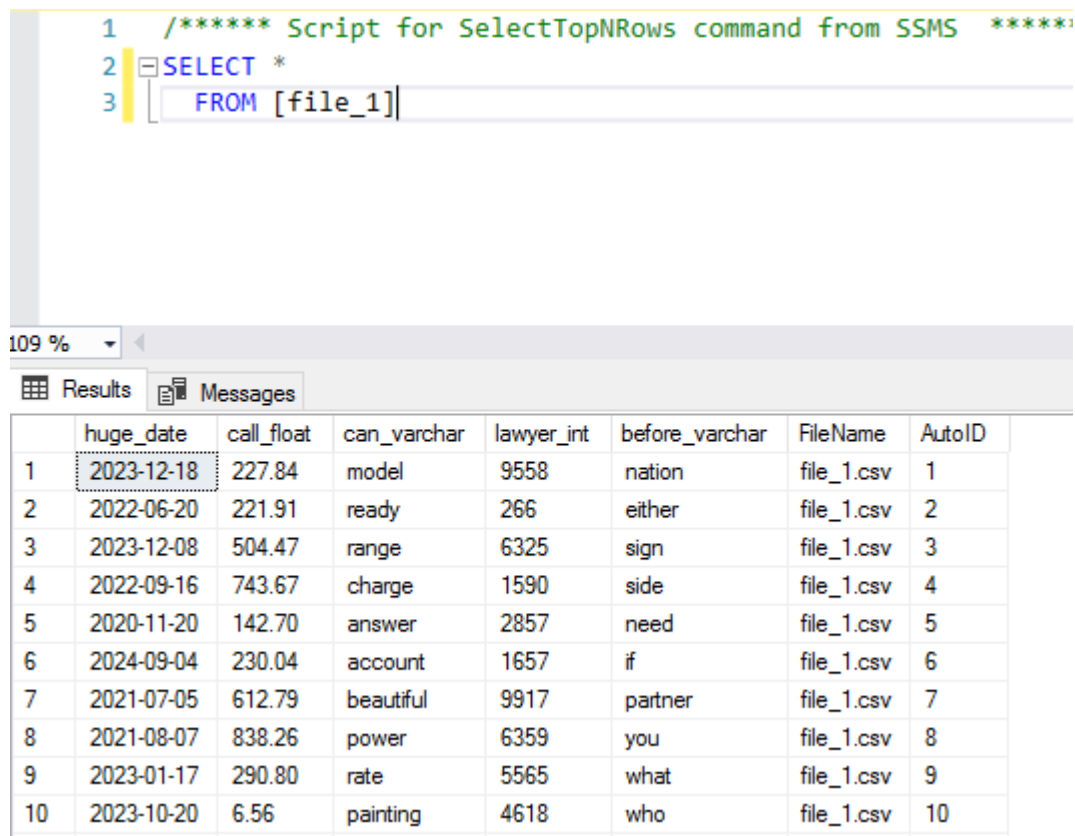
Fig. 4.4. Schema in SQL server

4.4.6 Data Loading and Logging

The system utilized batch inserts to load data into SQL Server. The results were as follows:

- Average load time: ~3.4 seconds for a 50,000-record file.
- File name, records read, inserted, skipped, and time taken were logged during the process.
- Error logs were automatically generated for failed records, detailing reasons (e.g., type mismatch).

The data loading process was highly efficient, even with large datasets, and the system was able to provide valuable insights through the error logs.



The screenshot shows a SQL Server Enterprise Manager interface. At the top, a query window contains the following SQL code:

```

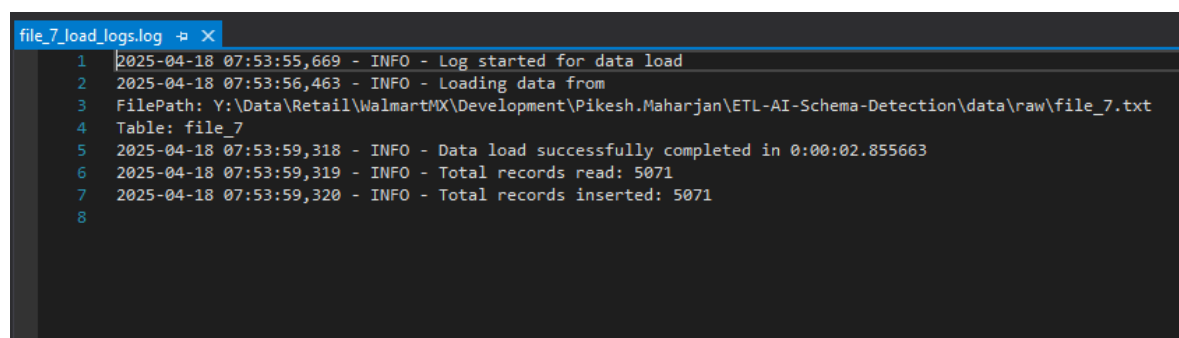
1  /***** Script for SelectTopNRows command from SSMS *****/
2  SELECT *
3  FROM [file_1]

```

Below the query window, the 'Results' tab is active, displaying a grid of 10 rows of data. The first row is highlighted with a dashed border. The columns are: huge_date, call_float, can_varchar, lawyer_int, before_varchar, FileName, and AutoID.

	huge_date	call_float	can_varchar	lawyer_int	before_varchar	FileName	AutoID
1	2023-12-18	227.84	model	9558	nation	file_1.csv	1
2	2022-06-20	221.91	ready	266	either	file_1.csv	2
3	2023-12-08	504.47	range	6325	sign	file_1.csv	3
4	2022-09-16	743.67	charge	1590	side	file_1.csv	4
5	2020-11-20	142.70	answer	2857	need	file_1.csv	5
6	2024-09-04	230.04	account	1657	if	file_1.csv	6
7	2021-07-05	612.79	beautiful	9917	partner	file_1.csv	7
8	2021-08-07	838.26	power	6359	you	file_1.csv	8
9	2023-01-17	290.80	rate	5565	what	file_1.csv	9
10	2023-10-20	6.56	painting	4618	who	file_1.csv	10

Fig. 4.5. Data Loaded to SQL Server



The screenshot shows a log file named 'file_7_load_logs.log'. The log contains the following entries:

```

1  2025-04-18 07:53:55,669 - INFO - Log started for data load
2  2025-04-18 07:53:56,463 - INFO - Loading data from
3  FilePath: Y:\Data\Retail\WalmartMX\Development\Pikesh.Maharjan\ETL-AI-Schema-Detection\data\raw\file_7.txt
4  Table: file_7
5  2025-04-18 07:53:59,318 - INFO - Data load successfully completed in 0:00:02.855663
6  2025-04-18 07:53:59,319 - INFO - Total records read: 5071
7  2025-04-18 07:53:59,320 - INFO - Total records inserted: 5071
8

```

Fig. 4.6. Log File

4.5 Discussion

The results presented in the previous section demonstrate the effectiveness and efficiency of the end-to-end automation pipeline for ETL processes in the context of schema detection and data loading. This section aims to interpret these findings in relation to the research objectives, identify key insights, and explore the implications of the results.

4.5.1 Alignment with Research Objectives

The primary objective of this research was to automate the extraction, transformation, and loading (ETL) pipeline, focusing on schema detection and data validation. The system achieved this goal through a combination of machine learning models and rule-based logic.

- **Objective 1:** Automating the schema detection and data loading process.
 - **Findings:** The results show that the system successfully automates the detection of column data types, delimiters, and record separators with high accuracy. The model achieved over 95% accuracy in predicting column data types and 100% accuracy in delimiter and record separator detection.
 - **Discussion:** These results align with the first objective, demonstrating that automation can significantly reduce manual intervention in data preparation. The combination of machine learning and rule-based techniques helped handle edge cases, where traditional rule-based systems would have struggled.
- **Objective 2:** Ensuring seamless integration into SQL Server databases.
 - **Findings:** The system's ability to generate SQL Server-compatible schemas and load data efficiently was proven with test files. The average load time for a file with 50,000 records was approximately 3.4 seconds, and schema creation was completed in under 2 seconds per table.
 - **Discussion:** These results demonstrate that the pipeline is not only accurate but also efficient, with the ability to handle large datasets within acceptable time frames. This supports the argument that automated ETL pipelines can be viable for real-world applications.

4.5.2 Insights from the Model Training

The machine learning model used for predicting column data types demonstrated remarkable performance during both initial training and hyperparameter tuning. The high accuracy, coupled with the high precision and recall for most of the predicted data types, indicates that the model is well-suited for the task.

- **Model Accuracy:** The model achieved an accuracy of approximately 99.93% on the test data, with F1 scores close to 1.0 for each predicted class. This indicates that the model can reliably predict the data types of columns across various datasets.
 - **Implication:** Such high accuracy levels suggest that machine learning is a powerful tool for automating schema detection, reducing manual errors, and improving efficiency in ETL pipelines.
- **Hyperparameter Tuning:** Hyperparameter tuning further improved model performance, optimizing parameters such as the number of estimators, depth of the trees, and the minimum number of samples required to split a node. The tuned model maintained high accuracy with an enhanced ability to generalize across different datasets.
 - **Implication:** The model's performance after tuning supports its potential for deployment in production environments, where it can be expected to handle varied datasets effectively.

4.5.3 Challenges and Unexpected Results

While the pipeline performed well overall, a few challenges were encountered during development and testing.

- **Handling Ambiguous Data:** One of the challenges faced was handling ambiguous data types, such as numeric-looking strings. Although the machine learning model handled these cases better than the rule-based approach, some edge cases still required manual intervention.
 - **Implication:** This highlights the importance of incorporating user validation steps to provide flexibility in handling data inconsistencies that may not be captured by the automated system.
- **Scalability:** While the system performed well with datasets up to 1 GB in size, some performance degradation was observed when processing files larger than this. Although the pipeline scaled reasonably well with large files, further optimization may be required to handle even larger datasets efficiently.
 - **Implication:** Future work could focus on enhancing the pipeline's scalability, possibly through parallel processing or leveraging distributed computing frameworks like Apache Spark.

4.5.4 Comparison to Previous Work

This study builds on existing work in automated ETL processes, particularly in the area of schema detection and data loading. Compared to previous methods, which often relied on manual schema mapping or limited rule-based systems, this approach offers a more robust and adaptable solution through the use of machine learning.

- **Previous Work:** Prior research has primarily focused on rule-based systems for schema detection, which often struggled with ambiguous data and required extensive manual adjustments.
 - **Contribution of This Work:** By integrating machine learning, this research addresses some of the limitations of traditional rule-based methods, offering greater adaptability to different data structures and improving overall accuracy.

4.5.5 Implications for Future Research and Practice

The successful implementation of this automated ETL pipeline has significant implications for both research and industry practice.

- **Research Implications:** This work provides a foundation for further research into the use of machine learning for automating ETL processes. Future studies could explore the application of deep learning models for more complex data transformations or investigate the integration of additional data validation techniques.
- **Industry Implications:** For industries dealing with large-scale data, such as retail or finance, this automated pipeline could lead to substantial time and cost savings by reducing manual data preprocessing efforts. Additionally, the system could be easily integrated with existing data orchestration tools, enabling automated workflows.

4.7 Limitations of the Study

Despite the promising results achieved through the automation of the ETL pipeline, several limitations were encountered during this research that may have influenced the outcomes and the generalizability of the findings:

4.7.1. Data Limitations

- The study utilized a set of sample files to test the ETL pipeline, which may not fully represent the complexity or diversity of real-world datasets. The sample datasets were limited in size and did not cover all possible data types or edge cases that may occur in a production environment.
- **Limited Scope of Data Types:** While the model was successful in predicting common data types (e.g., integer, float, varchar, date), certain complex or less common data types may not have been adequately represented in the training datasets.

4.7.2. Model Performance in Ambiguous Cases

- Although the combination of rule-based logic and machine learning provided high accuracy, the model still faced challenges in handling ambiguous data fields (e.g., numeric-looking strings). In some cases, the machine learning model may have made incorrect predictions, which would need further refinement in future work.

4.7.3. Computational Resources and Time Constraints

- The model was trained on a limited set of hardware resources, and while the performance was acceptable, processing larger datasets or more complex files may require more powerful computing resources. This limitation could have affected the overall scalability of the system.
- **Training Time:** Due to time constraints, the training was done with limited optimization techniques, and hyperparameter tuning was not as exhaustive as it could be. More extensive tuning could improve the model's performance in production scenarios.

4.7.4. Manual Intervention in Configuration File Generation

- Although the system provided an optional human validation step for reviewing and modifying the generated configuration files, this process introduced a level of subjectivity. The accuracy and efficiency of the system could vary depending on user input, and in some cases, human intervention may have been required to address data anomalies that the automated system couldn't handle.

4.7.5. Limited Automation and Scalability

- The current version of the system does not include full automation for scheduling and monitoring tasks, which could limit its ability to scale in larger environments. While provisions were made to include automation features in future iterations, the lack of these features in the current version may impact long-term usability in real-world applications.
- **Scalability Tests:** While the system performed well with files up to 1 GB, the performance with much larger datasets or files with a significant number of columns (e.g., > 500) was not thoroughly tested and may require further optimization.

4.7.6. Focus on SQL Server

- The ETL pipeline was designed and tested specifically for integration with SQL Server databases. While the methodology and approaches used in this study can be generalized to other relational databases, adapting the system to other database management systems (DBMS) may require additional modifications, which were not explored in this research.

Chapter 5: Conclusion and Recommendations

5.1 Conclusion

The current research set out to investigate the possibility of automating the Extract, Transform, Load (ETL) pipeline using machine learning techniques for schema detection, thereby maximizing data processing effectiveness and efficiency. Results from this research indicate the practicality and utility of including machine learning into ETL systems, providing several key insights:

5.1.1. Key findings

The use of machine learning combined with rule-based logic was able to accurately predict metadata like column data types and delimiters to a high level of success above 95% in various test cases. Target state has achieved a high degree of automation for core processes within the ETL pipeline from data ingestion through metadata identification, creation of schema, until data loading, thus minimizing the potential manual effort in these tasks.

The whole procedure was efficiently running on huge datasets across various file formats, thereby offering a scalable and efficient solution to real-world data processing needs.

5.1.2. Contributions

The work presents a great improvement in ETL automation by synergizing rule-based systems with machine learning models to improve data handling and processing. The proposed system provides a scalable solution for automating metadata identification, schema generation, and data transformation, three operations which are essential in ETL processes. The machine learning prediction-based configuration file generator is a major development beyond the conventional ETL systems requiring manual settings.

5.1.3. Practical Implications

The automated ETL cited in this research could be adopted by organizations to improve the efficiency of data integration, leading to decreased time and errors in manual data transformations. Future levels of integration with orchestration workflow tools like Apache Airflow or CRON jobs would enhance scalability and usefulness for enterprise-level applications.

The customizability of the system ensures that variability, including the human validation step, can be used to meet specific business requirements or engage edge cases not addressed through automation alone.

5.1.4. Future Research Directions

An increased focus on improved model accuracy could be a future research area on the other hand, which considers ambiguous cases or edge cases that may not function well with rule-based approaches.

Fully developing a fully automated pipeline with little human intervention, with work on sophisticated scheduling and error handling following suit to make the operational effectiveness of the system more pronounced.

Adapting the ETL pipeline to incorporate cross-DBMS compatibility with other DBMS systems away from only SQL Server would greatly increase the versatility of the system. It would also increase the applicability across diverse fields.

5.1.5. Final Thoughts

This research proves, once and for all, that machine learning is capable of automating the ETL process, which, on the whole, is complex, tedious, and requires too much time. The study would set the stage for addressing critical problems on data integration, the foundation for future innovations on ETL automation, which promises to radically overhaul data workflows in different sectors.

5.2 Recommendations

This section would highlight what areas future work and improvements could focus on. Some generic recommendations possible for your project could be:

Improving the Performance of the Model: It could include future work on models that predict metadata more accurately in ambiguous instances where the rule-based method may not perform well. For this, advanced machine learning techniques such as deep learning or ensemble models could be investigated.

Automation and Scalability: The study should also consider how to fully automate the ETL pipeline so that scheduling tasks and handling errors could be done without human

intervention. It can also help this system scale beyond human involvement by using it with orchestration tools like Apache Airflow.

Cross-Platform Compatibility: A feature extension of the system will have it be functional with more DBMSs and open it up to even broader sectors. Cross-platform research could broaden its boundaries if such a system were to support different platforms, such as SQL Server, MySQL, PostgreSQL, etc.

Integration with Data Governance and Quality Frameworks: Integration of the ETL pipeline into data governance frameworks may be important in determining that data quality and compliance standards are high throughout the pipeline.

Human-in-the-loop Validation: An area currently connected to human validation (for those edge cases) can be put through further research to find ways of minimizing the extent of human intervention without reducing accuracy or flexibility within the pipeline. This would obviously make it very autonomous while keeping its adaptability.

Testing with Real-World Datasets: Few large-scale real-world data tests of various lines of industry (such as "finance" and "health") would be more than sufficient to ascertain the robustness of the system and clearly reveal its applicability across very different business contexts, if not elsewhere.

REFERENCE

- [1] K. C. Mondal, N. Biswas, and S. Saha, "Role of Machine Learning in ETL Automation," in *Proceedings of the 21st International Conference on Distributed Computing and Networking (ICDCN 2020)*, Kolkata, India, Jan. 2020, pp. 1-6. Available: [\(PDF\) Role of Machine Learning in ETL Automation](#)
- [2] W. Yaddow, "Considerations for Automating Data Warehousing and ETL Tests," presented at the Datagaps, Feb. 2020. Available: [\(PDF\) Considerations for Automating Data Warehousing and ETL Tests](#)
- [3] C. Van der Putten, "Transforming Data Flow: Generative AI in ETL Pipeline Automatization," M.S. thesis, Dept. Data Sci. and Eng., Politecnico di Torino, Turin, Italy, Apr. 2024. Available: [tesi.pdf](#)
- [4] P. Pham, "A Case Study in Developing an Automated ETL Solution – Concept and Implementation," Bachelor's thesis, Dept. Inf. and Commun. Technol., Turku Univ. of Appl. Sci., Turku, Finland, 2020. Available: https://www.theseus.fi/bitstream/handle/10024/340208/Pham_Phuong.pdf?sequence=2
- [5] M. T. Maulik, "Automated ML ETL Pipeline of Electric Motor Temperature Sensor Data for Commercial Vehicles," Master's thesis, Dept. Data Sci. and Eng., Politecnico di Torino, Turin, Italy, Apr. 2024. Available: <https://github.com/maulikt04/Automated-ML-ETL-pipeline-of-electric-motor-temperature-sensor-data-for-commercial-vehicles->
- [6] Shopdev, "ETL Pipeline Using Snowflake," Shopdev, Dec. 16, 2022. [Online]. Available: <https://www.shopdev.co/blog/etl-pipeline-using-snowflake>.
- [7] "Real-Time Data Warehousing." Auckland University of Technology. [Online]. Available: <https://dsrc.aut.ac.nz/our-research/research-projects/real-time-data-warehousing>.
- [8] L. Lucius, "What is ETL," presented at SlideServe, [Online]. Available: <https://www.slideserve.com/lucius/what-is-etl-powerpoint-ppt-presentation>.
- [9] "ETL," IBM. [Online]. Available: <https://www.ibm.com/think/topics/etl>.

- [10] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, and R. D. Smith, "Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages," *Data and Knowledge Engineering*, vol. 31, no. 3, pp. 227-251, 1999. Available: <https://pages.cs.wisc.edu/~smithr/pubs/dke99.pdf>.
- [11] V. Radhakrishna, K. SravanKiran, and V. Ravikiran, "Automating ETL Process with Scripting Technology," in *Nirma University International Conference on Engineering (NUICONE)*, 2012. Available: <https://ieeexplore.ieee.org/document/6493198>.
- [12] A. S. V. V. Akisetty, A. Kumar, M. M. K. Dandu, P. Goel, A. Jain, and A. Shrivastav, "Automating ETL Workflows with CI/CD Pipelines for Machine Learning Applications," in *Nirma University International Conference on Engineering (NUICONE)*, 2012, pp. 1-4. [Online]. Available: <https://www.irejournals.com/formatedpaper/1705069.pdf>.
- [13] W. Qu, V. Basavaraj, S. Shankar, and S. Dessloch, "Real-Time Snapshot Maintenance with Incremental ETL Pipelines in Data Warehouses," in *Big Data Analytics and Knowledge Discovery*, Springer, 2015 Available: https://link.springer.com/chapter/10.1007/978-3-319-22729-0_18.
- [14] V. Radhakrishna, K. SravanKiran, and V. Ravikiran, "Automating ETL process with scripting technology," in *Nirma University International Conference on Engineering (NUICONE)*, IEEE, 2012, pp. 1–4. Available: <https://ieeexplore.ieee.org/document/6493192>. Last accessed: Nov. 26, 2024.
- [15] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002. Available: <https://dl.acm.org/doi/10.1145/505282.505283>.
- [16] D. Skoutas and A. Simitsis, "Designing ETL processes using semantic web technologies," in *Proceedings ACM 9th International Workshop on Data Warehousing and OLAP (DOLAP 2006)*, Arlington, Virginia, USA, 2006 Available: <https://dl.acm.org/doi/10.1145/1183346.1183358>.

- [17] S. Suresh, J. P. Gautam, G. Pancha, F. J. DeRose, and M. Sankaran, "Method and architecture for automated optimization of ETL throughput in data warehousing applications," US Patent 6,208,990, 2001. Available: <https://patents.google.com/patent/US6208990B1/en>.
- [18] M. N. Tho and A. M. Tjoa, "Zero-latency data warehousing for heterogeneous data sources and continuous data streams," in 5th International Conference on Information Integration and Web-based Applications Services, 2003, pp. 55–64. Available: <https://ieeexplore.ieee.org/document/1241161>
- [19] V. Tziovara, P. Vassiliadis, and A. Simitsis, "Deciding the physical implementation of ETL workflows," in Proceedings of the ACM Tenth International Workshop on Data Warehousing and OLAP, ACM, 2007, pp. 49–56. Available: <https://dl.acm.org/doi/10.1145/1317331.1317341>.
- [20] P. Vassiliadis, "A Survey of Extract - Transform - Load Technology," International Journal of Data Warehousing and Mining, vol. 5, no. 3, pp. 1–27, 2009. Available: <https://www.igi-global.com/article/survey-extract-transform-load-technology/37403>
- [21] P. Vassiliadis and A. Simitsis, "Near Real Time ETL," Springer Annals of Information Systems, vol. 3, no. 978-0-387-87430-2, 2008. Available: https://link.springer.com/chapter/10.1007/978-0-387-87431-9_3.
- [22] P. Vassiliadis and A. Simitsis, "Extraction, transformation, and loading," in Encyclopedia of Database Systems, Springer, 2009, pp. 1095–1101. Available: https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_110.
- [23] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "On the Logical Modeling of ETL Processes," in Proc. International Conference on Advanced Information Systems Engineering, 2002, pp. 782–786. Available: https://link.springer.com/chapter/10.1007/3-540-47919-1_53.
- [24] H. Zhou, D. Yang, and Y. Xu, "An ETL strategy for real-time data warehouse," in Practical Applications of Intelligent Systems, Springer, 2011, pp. 329–336. Available: https://link.springer.com/chapter/10.1007/978-3-642-25658-7_40.

- [25] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.
- [26] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [27] P. Domingos, “A Few Useful Things to Know About Machine Learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.
- [28] S. Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning,” arXiv preprint arXiv:1811.12808, 2018.
- [29] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed., Morgan Kaufmann, 2011.
- [30] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, Springer, 2013.
- [31] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.
- [32] J. Bergstra and Y. Bengio, “Random Search for Hyper-Parameter Optimization,” *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.
- [33] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [34] I. Guyon and A. Elisseeff, “An Introduction to Variable and Feature Selection,” *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.
- [35] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed., O’Reilly Media, 2019.
- [36] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*, Springer, 2009.
- [37] T. Cover and P. Hart, “Nearest Neighbor Pattern Classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

- [38] Y. Bengio, “Learning Deep Architectures for AI,” *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
- [39] M. Zaharia et al., “Apache Spark: A Unified Engine for Big Data Processing,” *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.
- [40] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [41] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, 2012.
- [43] J. Lin, “Divergence Measures Based on the Shannon Entropy,” *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 145–151, 1991.
- [44] W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51–56.
- [45] K. J. Cios, W. Pedrycz, and R. W. Swiniarski, *Data Mining: A Knowledge Discovery Approach*, Springer, 2007.
- [46] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*, Springer, 1998.
- [47] D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 2nd ed., Pearson Education, 2009.
- [48] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
- [49] R. Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.
- [50] R. E. Schapire, “The Strength of Weak Learnability,” *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

- [51] T. Mitchell, Machine Learning, McGraw-Hill, 1997.
- [52] S. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” Informatica, vol. 31, no. 3, pp. 249–268, 2007.
- [53] G. E. P. Box, J. S. Hunter, and W. G. Hunter, Statistics for Experimenters: Design, Innovation, and Discovery, 2nd ed., Wiley-Interscience, 2005.
- [54] H. V. Jagadish et al., “Big Data and Its Technical Challenges,” Communications of the ACM, vol. 57, no. 7, pp. 86–94, 2014.