UMEÅ UNIVERSITY

A Comparative Analysis of Metadata Tools for use on Unknown Operational Datasets

Leo–Ra Schwieder Juneblad

Leo–Ra Schwieder Juneblad
Spring 2024
Degree Project in Computing Science and Engineering, 30 credits
Supervisor: Alexandre Bartel
External Supervisor: Torgny Holmberg
External Partner: Ericsson
Examiner: Henrik Björklund
Master of Science Programme in Computing Science and Engineering, 300 credits

Abstract

When working with large datasets it is important that the right tools and methods are selected in order to effectively, it is important that the right tools and methods are selected in order to effectively analyze the data. This thesis presents a comparative evaluation of data management tools in the categories of validation, profiling, and feature extraction. The tools, Pandera, Ydata Profiling, SweetViz, and Tsfel, were selected and integrated into a data processing system for the WARA–Ops portal in order to validate, profile, and analyze new operational datasets uploaded to the portal. Finally, the system extracts statistical information from the dataset and uses a machine learning classification algorithm to apply a general label to the data based on the extracted information.

Acknowledgements

I want to extend my thanks to Alexandre Bartel, my supervisor, who provided discussion and feedback on this project. I also want to thank my external supervisor Torgny Holmberg, and the team at Ericsson Research, who gave me guidance and provided the inspiration and help I needed throughout the entire project. Finally, I want to thank Eva, Paul, and Viggo for giving me the moral support I needed to finish this thesis.

Contents

1 Introduction

As the amount of digital systems in the world grows, the amount of data generated grows as well. For a modern company to effectively manage its operations, it is important to manage the data that the operations generate. These types of large datasets are often referred to as big data, meaning datasets that are too large to process with standard data processing methods. Another aspect of big data that is important to consider is the velocity of the data, that is the speed at which data is generated. It means that any modern big data processing methods must be able to handle large amounts of data at a fast enough rate so as to not become overwhelmed.

When operating on large datasets it is important that data is both machine and human readable. Because of the amount of data it is impossible for processing to be done manually by a human, therefore it is important that the data be machine readable. On the other hand any results produced by machines should be verified by humans, meaning it is best for the data to be understandable for humans as well. The primary way of doing this is by adding metadata.

Metadata is most commonly defined as "data about data". It commonly contains information such as descriptions or keywords that describe the contents and structure of the data. However, it can also consist of statistical and summarizing information about a dataset. Metadata can be created automatically or manually. Automatic generation tends to give more rudimentary metadata compared to doing it manually. It can give information such as the time the data was created and who created it. On the other hand, manual metadata creation can be much more detailed but is much more time–consuming. If a user wishes to host their data on a certain platform but they are required to create metadata first, it may cause them to turn to a different hosting service where such tasks are outsourced. The challenge, therefore, is creating high–quality, statistical metadata with as little user interaction as possible. This comes down to choosing the correct tools for the job.

This thesis investigates various data processing, statistical metadata creation, and data visualization tools. The focus is on their ability to accomplish tasks with minimal user involvement, particularly in identifying and presenting intriguing sections of large operational datasets. This approach involves a comparative evaluation and implementation of tools for metadata creation and data visualization, with the outcome of a bespoke system for metadata generation. This will be done together with Ericsson research, and the final result will be implemented into the WARA–Ops web portal. WARA–Ops is a research project that seeks to develop new methods for big data management. The WARA–Ops portal is a web–based interface that provides access to large operational datasets.

Research Question

In order to evaluate data processing tools for use on a modern operational database, this thesis aims to answer the following question, which is split into three aims, each with its own objectives.

What are the best methods to automatically extract and visualize statistical metadata from large unexplored datasets?

Aims:

Aim 1 Perform a literature analysis and evaluation of currently existing tools.

- Identify appropriate metrics for analyzing the effectiveness of tools.
- Literature study to identify, categorize, and rank existing tools.

Aim 2 Design a data processing system to automatically extract and display metadata from an unknown dataset.

- Identify important statistics of a large dataset to extract and visualize in order to analyze the data.
- Identify the most appropriate tools for the system using previously identified metrics and rankings.
- Construct a data processing system using the proposed design and selected tools.

Aim 3 Validate the data processing system against the existing tools.

- Evaluate the data processing system against previously identified metrics.
- Comparatively evaluate the system against previously analyzed tools to determine what value it provides.

2 Related Work

Metadata extraction and generation has been a popular research topic for many years and has become even more relevant with the rapid spread of big data. A lot of research has been done on generating and extracting metadata from large and small datasets. Such as [9], from 2004, that comparatively analyses tools for extracting website metadata. However, most research on the automatic extraction of metadata is relegated to specific fields where specific types of metadata creation are required.

In [20], different tools for automatic metadata generation are evaluated in the context of the needs of a library. They find many interesting points of discussion on automatically generating or extracting metadata using different tools. However, they are limited to discussing only from the perspective of the library community. When it comes to more statistical metadata, there is less common research.

Both [14] and [28] analyze automatic data visualization tools for selecting appropriate visualizations of data statistics. Both present valuable and effective tools that facilitate quick visualization for exploratory data analysis. Further, [16] provides an evaluation of data visualization tools for analyzing data from space sensors. [25] evaluates multiple data visualization tools for use in analyzing biological data. While this research presents effective tools for automatic data visualization, it is still done with respect to the specific domain the tools are used within and not an entirely general setting.

3 Background

Statistical metadata generation and data analysis is a complicated topic that covers a lot of ground. Therefore, it is important to define the scope of this thesis and lay down the groundwork for understanding the results presented.

3.1 Big Data

Oracle describes big data in [18] as large amounts of complex data generated quickly. Managing big data today is more important than ever, especially in large digital systems where daily operations can generate massive amounts of data. A more rigorous definition of big data is the three Vs: volume, velocity, and variety. Volume implies a large amount of data generated. Velocity means that the data is being generated at a high rate of speed, and variety describes the high variance in what type of data is being generated. Further Vs have been added, such as value and veracity, but the first three are the main components that define big data. To gain any valuable information at all from big datasets, performing some form of big data analysis is necessary. Big data analysis or big data analytics simply means using tools to summarize and analyze very large datasets.

3.2 Metadata

Metadata is most simply defined as ”data about data”. For humans and machines to be able to handle the enormous amounts of data they come across every day, metadata is a vital resource. There are many types of metadata, categorizing different types of information for different purposes [24, p. 6,7]. For example, technical metadata describes technical aspects of some data, such as file size and type. Structural metadata, on the other hand, describes how data is structured. It describes information such as page numbers or the sequence order in the data. For this thesis, however, generating descriptive metadata is the main focus. Descriptive metadata describes the information needed to understand the data, such as the name of the data, where it came from, or statistical information on the data. While much of metadata creation today is created to enable machine readings of the data, good, high–quality, human–readable metadata is important for understanding the large amounts of data being generated.

While metadata is generated all the time, for almost every piece of data created, it can only be useful if it accurately represents the data it is derived from. So–called high–quality metadata. High–quality metadata does not have a strict definition and so can be hard to qualify. However, one generally agreed upon important aspect of creating high–quality metadata is taking into consideration the context within which it is created. Xavier Ochoa and Erik Duval in [17] evaluate metadata quality measurements for digital learning repositories. They

state that “The tasks metadata should enable in a digital repository are to help the user to find, identify, select and obtain resources. The quality of the metadata will be directly proportional to how much it facilitates those tasks.” They make it clear that the quality of the metadata is evaluated on how useful it is to the context in which it was created. For descriptive metadata in an analytical context, high–quality metadata should provide an accurate view of the data in a smaller size that can be easily analyzed.

3.3 Data Mining

Data mining is the practice of using computers to semi–automatically search through large datasets, as described by Gillis, Stedman, and Hughes in [6]. Data mining primarily aims to find hidden patterns and valuable information that can be extracted from large, difficult–to–explore data. One key aspect of data mining is that information can be extracted from unknown or unexplored data. That is data that the person or system performing the data mining is not immediately familiar with.

3.4 Time Series Data

As described in [11], a time series dataset is a sequence of events recorded over time. Thus, a time series entry usually contains the resulting values of the event paired with a timestamp of when those values were recorded. An operational dataset is a record of systems operations, most often over some timespan, and therefore, it is often stored as a time series. Time series data can also show certain unique correlations and patterns that are not present in a normal dataset.

In contrast to time series analysis, concerned with predicting future events given a series or analyzing past events of a known series. The objective of this thesis is to create a generalized system that extracts analytical information and presents it to a person who can then perform their own analysis.

3.4.1 Time Series Motifs

A somewhat new aspect of time series analysis is time series motif detection. In [15], Mueen et al. describe a time series motif as a sub–sequence of a time series repeated one or more times in the series. Depending on what the data represents, a motif can provide valuable information by showing time–based, repeating patterns in the data.

Finding time series motifs entails repeatedly comparing subsections of a series. This can be a very computationally intensive process for large time series. A method called the matrix profile has been developed to improve it. Calculating the matrix profile for a time series entails calculating the distance between a subsequence and the rest of the data using a sliding window. This is done for all subsequences in the series, and the shortest distance for each subsequence is recorded. The shortest distances are then compiled in a vector called the matrix profile. By observing the lowest values in the matrix profile, one can see the subsequences that are most alike.

3.5 Summarizing Data

Data summarization is the process of identifying key features in a dataset to provide an overview of it, as explained in [31]. It is a key step in exploratory data analysis (EDA), as it gives a more digestible view of a dataset that can serve as a good basis for further exploration and analysis. One of the most used statistics for summarizing data is central tendency. Central tendency refers to the center of a dataset. It is often calculated using the mean or average. Other useful summarizing statistics are the dispersion and spread of the data. This can include variables such as variance, skewness, and kurtosis to determine how spread the data is and its general distribution.

3.6 Correlation Analysis

N. Gogtay and U. Tatse [8] describe correlation or correlation analysis as a way of defining how much two variables affect each other. There are different forms of correlation analysis. However, a popular one, and the one used in this thesis, is Pearson correlation. Pearson correlation distills the linear relationship between variables down to a number between -1 and 1, where -1 indicates a perfect negative correlation and 1 indicates a perfect positive correlation. A positive correlation means that the variables increase together, while a negative correlation means that one variable increases while the other decreases.

3.7 Data Clustering

As Aghabozorgi, Shirkhorshidi, and Wah describe in [1], clustering is a type of data analysis that attempts to group data points together into clusters, usually without in–depth knowledge of any preexisting categories. It is an unsupervised learning technique, meaning no verification of results takes place, and no outputs need to be known. Clustering algorithms look at a data point and compare it to other data points in the set. Points that are determined to be similar are placed in the same group while points that are less similar are placed into different groups. Two point's similarity are defined by the heuristic used by the algorithm.

Clustering can be very useful in the analysis of time series data as it can provide insights into underlying patterns and groupings that the data presents over time. However, in order to properly analyze a dataset through clustering, a clustering algorithm must be chosen. Two popular types of clustering algorithms are hierarchical and partitioning clustering. Hierarchical clustering can be done agglomeratively or divisively [1]. Agglomerative clustering is done by placing every data point into its own cluster and then merging the closest pair of clusters. This is then repeated until only one cluster remains, or a suitable number of clusters is reached. Divisive hierarchical clustering is the opposite. All points start as one cluster and are divided until they are all in separate clusters, or a suitable number of clusters is reached. Partitioning clustering, on the other hand, works by placing a set of data points into a set number of clusters such that each cluster contains at least one data point.

One of the most popular partitioning clustering algorithms is k–means clustering. It works by grouping points around k cluster centers [12]. It begins by randomly selecting k points that serve as cluster centers. It then calculates the distance from each data point to the cluster

centers and groups each data point to the closest center. It continues by recalculating the centers to be the actual centers of each cluster. It then repeats by assigning each point to its nearest cluster center and recalculating the centers. This is done until the centers converge, stop changing, or a set number of iterations have passed.

One problem with k–means clustering is choosing an appropriate value for k. There exist a few methods for this, such as the elbow method, and the silhouette score. The silhouette score is a value between -1 and 1 and measures how similar a point is to its own cluster compared to other clusters [10]. A high silhouette score can indicate well defined clusters. By testing different k values and recording the silhouette score one can find a potentially optimal k value by selecting the one with the highest silhouette score.

3.8 Tools

The following tools are evaluated for use on datasets in the WARA–Ops portal.

3.8.1 Pandas

Pandas is an open source data analysis tool for Python [19]. The primary feature it provides to this project is the dataframe object. A dataframe is a two dimensional array–like object that contains data where each column may contain a different type. Visualizing a dataframe shows it is similar to a spreadsheet. All the datasets in the WARA–Ops portal are in dataframe format, which is good as Pandas provides a large number of features for analyzing dataframes.

3.8.2 SciKit–Learn

Scikit–learn (Sklearn) is a popular, open–source machine learning library for Python. It provides a multitude of tools for constructing machine learning models, as well as datamining and data analysis. These include tools for classification, clustering, and dimensionality reduction.

3.9 Tools for Evaluation

There are many challenges to generating high quality metadata, and in order to do it effectively, the right tools are required. Without domain knowledge, some data validation and exploration is required, meaning no single tool can serve as a complete solution. In the case of generating metadata for unknown datasets multiple tools are needed for different purposes. The following sections divide the tools into validation, profiling, and feature extraction. A restriction placed on the tools used in the project is that they be compatible with Python as that is the language used in the WARA–Ops portal.

3.9.1 Validation

In data management, validation can entail a lot of things, but mainly it is about defining how the data is expected to look. This includes the shape of the dataset and the datatypes it consists of. Validation is important in many different cases, specifically for the purpose of this thesis it limits the need to guess what a dataset should look like. By having a user provide the shape of a dataset it is much easier to process effectively.

Pydantic

Pydantic is an open source, Python–based data validation tool [21]. It is focused on providing type safety in Python code by allowing users to add data validation points in their programs. It functions by defining a schema class that contains the expected names and datatypes of the data and then passing the data to the schema class. If the validation fails an error is raised specifying where the data does not match the schema. Data can be passed through the schema one piece at a time or put together as a dictionary. Pydantic also provides features for defining expected values, and so called data coercion, meaning if a datatype of a column does not match the schema, Pydantic can try to convert the column to that datatype. Beyond that, Pydantic is a widely used tool and has a large pool of resources available to potential users.

Great Expectations

Great Expectations is a data management tool for validating and documenting large amounts of data. It applies the philosophy of unit testing to data management by letting user efficiently build comprehensive validation suites for their data [5]. It specifically facilitates the processing of live data by allowing the creation of validation checkpoints. The checkpoints can be connected to data sources and when new data is produced by the source, it can be automatically passed through the checkpoint that can validate it. The focus of the validation is slightly different from most of the other tools mentioned in this report. Great Expectations lets users place expectations on the data that verify its characteristics. There is a large number of predefined expectations and it is possible for a user to create their own as well. With expectations, the focus is on verifying specific characteristics of the data instead of the focus being on verifying the datatypes, as with Pydantic, for example. The design of Great Expectations makes it ideal for data pipelines that handle large amounts of data, as long as the expected shape of the data is known beforehand.

TypedFrame

TypedFrame is an open source data validation tool for Pandas dataframes developed by Alexander Reshytko [23]. It is different from all other validation tools mentioned in this report as it only does type validation. It is built as a wrapper for the Pandas dataframe. Like Pydantic, the user defines a schema class that contains the expected columns of a dataframe as well as the expected type of each column. The dataframe can then be passed to the schema class, and any data not conforming to the schema raises an error. The only auxiliary feature it provides is for converting nonconforming columns to the expected datatype. TypedFrame is designed so that functions can take a schema class as input, thus providing a

clear template for how the input data should look. Thus the focus is on providing clear data entry for programmers.

Marshmallow

Marshmallow is a data validation, serialization, and deserialization tool developed and maintained by Steven Loria, Jérôme Lafrèchoux, and Jared Deckard [13]. The validation works by creating a schema class that defines the expected names and datatypes of each column of the validated data. Extra features are provided to define expected values, and value ranges as well as full validation of nested datatypes. Beyond validation, Marshmallows main features are data serialization and deserialization. Validation schemas can be applied easily before a data object is serialized or deserialized. This makes it very good when the integrity of data must be verified before it is stored or sent to some other system.

Pandera

Pandera is an in–depth data validation tool for use on dataframes in Python [27]. It supports validation for different types of dataframes. However, only Pandas dataframes are relevant to this project. Just as with the previously mentioned tools, it lets users define a validation schema class that contains the expected data columns and their respective datatypes. It is somewhat similar to Pydantic, both in function and focus, as both tools aim to create clearer typing for Python data processing. One of the biggest differences between them is the features they provide beyond type validation. Pandera gives a range of options for validating data. It supports deeper validation such as expected values and conversion of nonconforming datatypes. But it also provides function decorators so validation can easily be integrated into data processing pipelines. One of its most distinct features is the fact that it lets users define validation schemas using the syntax of Pydantic. This can be very useful depending on if users are more comfortable with the Pydantic syntax.

JSON Schema

JSON Schema is a format specification for validating JSON files described in [26]. This thesis discusses the Python implementation of the specification, developed by Julian Berman [3]. A JSON schema specifies how a JSON document should be formatted, this includes the expected names and datatypes of fields. A JSON object and the schema can then be passed to the validator and either pass the validation if it aligns with the schema or, if it does not, a validation error is raised. It is quite in–depth as it provides many more features for validation, such as optional fields, value ranges and expected values of fields. Descriptions can also be added to properties in the schema to describe the fields of the JSON document. Because it uses a JSON format is very practical for creating readable definitions for JSON data.

3.9.2 Data Profiling

Data profiling is a step in data analysis where a rough picture of the data is painted. It can include things such as amounts of missing values, sections of constant values, correlations,

etc. The following are tools for automatic data profiling.

Ydata Profiling

Ydata Profiling is a data analysis tool for the initial stages of data processing [29]. It automatically generates a profile of the data, containing statistics and categories in the dataset, in order to ascertain the quality of the data before it can be processed further. It also provides a special profiling mode for analyzing time series data, which is relevant to this report. This mode adds deeper analysis of the timestamp values and a few new labels that are time series specific.

The tool is designed to facilitate easy exploratory data analysis. It is a good out–of–the–box solution as it compiles the stats it calculates into a full report on the data presented in an HTML document with included graphs and charts. And if a dataset is very large, it allows certain computations to be left out of the report to lower resource consumption. Because of the visual means in which the profile is presented, the results from Ydata Profiling are both easily understood and informative.

Sweetviz

Sweetviz is a data profiling tool developed by Francois Bertrand [4]. It functions similarly to Ydata Profiling by analyzing a dataset and visualizing the results in a report. Although some of the content of the analysis differs from that in Ydata Profiling. Sweetviz also provides some features related to comparing two datasets or comparing different columns in a single dataset. Just as with Ydata Profiling, it is a good out–of–the–box solution for EDA.

Dython

Dython is a small, lightweight data analysis tool developed by Shaked Zychlinski [30]. While it is not specifically focused on data profiling, it still provides a good interface for generating statistical data on a dataset. The main feature of Dython is the association graph that is very easy to generate for a dataset to see the extent to which the numerical variables in the dataset are associated. As stated by Zychlinski, it is focused on ease of use and so efficiency was deprioritized. So it may be less suitable for use on large datasets.

3.9.3 Feature Extraction

Feature extraction tools analyze a dataset and pick out predetermined variables, called features, that may be valuable. Many of these tools are designed to extract features relevant to training machine learning models. However, some of these features can still be useful for understanding the dataset.

Tsfresh

Tsfresh is a time series feature extraction tool [7]. It calculates a large number of characteristics or features of a dataset. The intended purpose of the tool is to extract features that may be relevant to training or running the data through machine learning models. Because of this, it tries to extract a lot of features but also provides methods for filtering out values that are deemed insignificant by a significance test. Because it is designed specifically for time series data it requires the timestamp variable when it is run. Tsfresh also requires the variable that identifies the time series. Should a dataset contain multiple time series, the features for each one are calculated separately. This provides more precise features, however, its usefulness depends on the information available about the dataset.

Tsfel

Tsfel, or Time series feature extraction library, is exactly that, a Python–based tool for time series feature extraction [22]. It functions very similarly to Tsfresh, it even calculates very similar features. However, some features differ between the tools. Tsfel provides a configuration variable that can be modified to extract certain types of features making it very easy to customize what is extracted. Tsfel also provides the option of leaving out the time series id variable, meaning that even if the dataset contains multiple time series, features are calculated assuming they are one series. This is useful when analyzing unknown datasets, but provides less clarity in the features.

Stumpy

Stumpy is a Python–based time series analysis tool [2]. Specifically Stumpy provides algorithms for calculating the matrix profile for a given time series. Using the matrix profile features such as motifs or outliers can be extracted from the dataset. One of the main goals of Stumpy is performance, it is designed to efficiently compute the matrix profile. Calculating the matrix profile does require some domain knowledge, so it may be less effective on unknown datasets.

4 Solution Design

In order to answer the research question proposed in Section 1, the tools in Section 3.9 must be examined and evaluated. Then, a system is to be designed together with Ericsson, where the tools are implemented and tested. This also provides some evaluation metrics for the tools as they must conform to the needs of the system. When a user uploads a dataset to the portal, it enters the system, which functions as a pipeline and extracts potentially relevant metadata and statistical data, and presents any interesting features to the user. Before testing, the tools are ranked against some general metrics to determine how easy they are to work with and what value they provide. Once ranked, they are tested against some general metrics derived from the review in Section 3, and should the design of the pipeline create any specific metrics, the tools are tested against them as well in order to find which are most suitable for use, which can be discarded, and if any further development of custom tooling should be done.

In order to begin selecting tools it is necessary to answer the first aim of the research question presented in Section 1. Of the tools that are evaluated favorably the best ones of each category are selected for implementation into the pipeline. To answer the second aim, the pipeline itself is designed, and the statistical metrics explained in Section 3 are examined to determine which are most relevant to the final stage of the pipeline. Finally, to achieve the third aim, the pipeline itself is evaluated against the same general metrics as the tools, to determine its usefulness as a data processing system.

4.1 Selecting Tools

Selecting the appropriate tools for the pipeline is the first step in its construction. To do this, the first step is to rank the tools using common metrics. They are then evaluated on the results of the testing, and the highest ranked tools are selected. Because the pipeline is split into three distinct stages, the tools are split into three categories and the rankings are done within each category. However the evaluation metrics are very similar within each category.

4.1.1 Initial Evaluation

The initial ranking must be done first as provides insights into what the tools can do and how they fit into a general metadata management process disconnected from the specific context of the pipeline. Once the initial ranking is done, each tool is tested and ranked again based on the initial ranking and the results of the tests. While the tool's second ranking determines the selected tools, ultimately, the tools are only restricted by the requirements of the pipeline, meaning the highest–ranked tools may not always be the ones chosen.

The initial ranking is done per category, as they are in Section 3.9. General metrics when

ranking all tools are the community around the tool, as well as the advertised focus of each tool. Some special metrics are used in each category. In the validation category, the types of available validation are considered. In the data profiling category, the information extracted is considered. For the feature extraction category, the customizability of the tool, is considered.

4.1.2 Secondary Evaluation

As explained in Section 3.9, there are several practical metrics for determining a good metadata management tool. However, because the pipeline is constructed of multiple, more specialized tools, the tools are not all similar to many of the metadata management tools evaluated there. Several relevant metrics exist while independent of tool type. There are specifically four main metrics that are examined for all tools:

- Accessibility
- Running time
- Time complexity
- Flexibility

Accessibility encompasses a few features, and it is important as tools with good accessibility interact with them quickly and effectively. Accessibility mainly refers to a small learning curve for new users. It is evaluated by analyzing the interface of the tools. Specifically, this means the tools syntax and potential input requirements and output information. In order to be useful the system must be understandable. Users are not necessarily very familiar with Python or any programming language. Thus, any tools used must be readable and understandable to someone not familiar with the interface. Users may be somewhat unfamiliar with the data they are processing as well which means the tools must provide a comprehensive view of what they do and how in order to facilitate exploration of the data.

Running time is simply the time it takes to run the tools on some input data. It can be an important aspect of a tool depending on its use case. If a fast output is required in a system, faster tools are required. This is evaluated by running the tools with some input data and recording the time to process it.

Time complexity is an important metric as these tools must be able to handle very large amounts of data. A tool with high time complexity would drastically slow down any system in which it is integrated. Time complexity can be evaluated by running the tools with increasingly large datasets and recording the running time. The results can then be analyzed to determine matching curves and thus find the complexity.

Finally, flexibility is important because it is not unusual for data to change, and thus, any metadata schemas must adapt to the new data format. The tools must allow for changes to the input in order to be used effectively. Flexibility is evaluated by manually analyzing the interfaces of the tools to determine when and where the tools allow the expected input data to change.

Some restrictions are also placed on the tools by the pipeline design. Mainly, the tools must be robust so as to be able to handle missing values or values of the wrong type. This is

related to the flexibility aspect of the metrics specified. Because the purpose of the pipeline is to process any operational dataset that users wish to upload to it, it is expected that data formats for the datasets are not set in stone. The format of the datasets may change, both in small and large ways, such as a column changing types from integer to a floating point number, to several new columns being added and several old ones being removed. This means that the tools used must be at least somewhat changeable to accommodate new data.

Running time is not a critical metric for the pipeline. It is expected that processing the data in the pipeline takes some time, so speed is not a necessary focus for the tools. However, it still provides a good foundational metric for ranking the effectiveness of the tools. Time complexity is also a concern. As stated, the datasets are very large, so tools must not eat up time and resources whenever they are run.

4.2 Pipeline Design

The system for data analysis, a collaborative effort between the engineers at Ericsson Research and me, is structured as a pipeline that takes a Pandas dataframe as input. When a user uploads a new dataset to the portal, the dataset enters the pipeline, undergoes processing, and the results are presented to the user. The design of the pipeline prioritizes explainability, ensuring that users can comprehend each decision made.

The pipeline consists of three stages. The first stage validates the dataset and lets the user modify any assumptions the system makes. If a column is not of the expected type, the user has the ability to change it, and the pipeline attempts to convert the column to the appropriate type. Apart from the shape of the data and general datatypes of each column of the dataframe, it does not extract any in–depth statistical metadata from the dataset.

The second stage is more in–depth and provides more information about the data. The main purpose of the second stage is to build a general dataset profile. It should include simple statistical information on the data, such as correlation charts, missing values, and means and averages of numerical columns, as well as common values of categorical columns. While this is stage two of the pipeline, it must not necessarily finish before stage three as both stages extract independent information that is presented to the user when the data has been processed.

Finally, stage three provides more in–depth information on the dataset. This stage cleans the data and extracts more complex features such as autocorrelation, outliers, and clusters. Some of the features extracted are then used to ascribe labels to the dataset. These labels include information such as whether the dataset is scattered or tight, or strongly or weakly correlated. Just as with stage two, the focus is on extracting valuable information, whereas speed and memory usage are a lower priority. A high level overview of the pipeline can be seen in Figure 1

As stated, the main requirement of the pipeline is explainability. It must also be robust to ingest and process any file that a user uploads, provided it is a time series in a dataframe format. The user must be given an explanation of how every feature was extracted and why. This must also be true if the pipeline fails. If a problem occurs, the user must know why.

In order to properly analyze the datasets, methods such as dimensionality reduction and clustering are used. All steps must also be presented to future users. This means the third

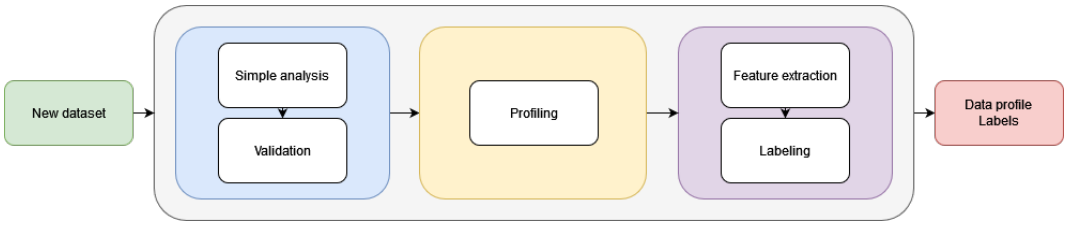

Figure 1: Overview of pipeline

step of the pipeline should be transparent but robust enough that users don't need to be an expert in data science in order to effectively use it.

The features calculated in the third step of the pipeline are used to train a classification model to label the dataset. Implementing such a model requires the extraction of information that can contribute to classifying the data. Gathering the classification metrics beforehand is important, as the description must be transparent enough that the user can understand what it is based on. Training such a model also requires data to train it on. Creating training data entails extracting the selected statistical features from a variety of datasets, in order to get a comprehensive spread of values for different types of datasets. The training data may be extracted either from datasets that exist in the portal or be synthetic.

4.3 Metrics for Analysis

The ultimate goal of the pipeline is to classify the data depending on the metrics extracted, but before that, it must be decided which analytical metrics should be extracted to contribute to the classification.

4.3.1 Data Summary

To begin summarizing a dataset, as described in Section 3.5, the size and shape provide a good starting point. However most users may be at least somewhat familiar with this information. So a good way to provide a better view of a dataset is to find the center. This can be done using the mean or median of the data. Then finding the spread of the data provides a clearer picture of the form of the data. These metrics give a broad overview of the shape of the data. However, more features can be added, such as skewness and kurtosis, to give even further insight into the distribution of the data. In the third stage of the pipeline, the variance of the dataset is calculated as a representation of the spread and used to label the dataset.

4.3.2 Underlying Patterns

Because the datasets that pass through the pipeline are all time series, it is assumed that some underlying patterns may be present in the data over time. These can be analyzed through time series motifs as explained in Section 3.4.1. Autocorrelation of variables in the dataset can also help with finding repeating patterns in the data. Finally, groupings may be present in the data that may be visualized and discovered using clustering algorithms discussed in Section 3.7. As there is a high chance that the data is high dimensional, some reduction must

be done in order to visualize any potential clusters. This is done using Principal Component Analysis (PCA), and t–distributed stochastic neighbor embedding (tSNE). PCA provides dimensionality reduction while keeping as much variance as possible from the original data, and tSNE is used to visualize the data in two and three dimensions.

Just as with the data summary, some of these metrics are fed to a classification model to produce a label for the dataset describing the spread and groupings in a simple manner. All of these metrics are also visualized and presented to the user as they are.

4.4 Evaluation of Completed System

Once the data processing pipeline is completed it is tested against the same general metrics as the individual tools were. That is: accessibility, running time, time complexity, and flexibility. Running time and time complexity are tested in the same manner as the tools. By running the system on different datasets, and recording the timings. However, the fact that the system incorporates multiple tools is considered as that makes it slower than any individual tool.

Accessibility and flexibility are tested slightly differently from the other tools. Because the pipeline is made up of multiple different tools it is only as flexible as the least flexible tool. Any part of the pipeline not made up of premade tools is tested to see how well it can adapt to new data. Accessibility is evaluated by analyzing the outputs from the pipeline to see how well it explains any decisions taken regarding the data. Using these metrics the pipeline as a tool is evaluated.

5 Implementation

The first step in implementing the system, detailed in Section 4, was evaluating the tools that were gathered. Once tools had been evaluated and selected, the pipeline was constructed, and the tools were integrated into the system.

5.1 Tool Evaluation

As stated previously, the tools were first ranked according to the features collected in Section 3.9. They were then tested and evaluated on the metrics described in Section 4. Finally, the best–evaluated tools in each category were chosen for implementation into the pipeline.

The initial evaluation of all the tools was done using three metrics: the community around the tool, the features provided beyond type checking, and the stated aim of the tool. The tool's aim is the primary metric for the first ranking. The community and features are valuable but less important than what the tool's main purpose is.

5.1.1 Data Validation

The secondary evaluation was done on the four metrics described in Section 4.1.2: running time, time complexity, accessibility, and flexibility. These metrics were tested slightly differently within each tool category.

To begin with, testing the running times of the validation tools was a straightforward process of running all the tools on a large dataset and timing the performance. Timing was done using the Python time package to get the time before and after validation and subtracting the start time from the finish time. The validation was done on an example dataset from the WARA–Ops portal of metric data from the Ericsson Research data center. The dataset contained 8,095,419 entries corresponding to about 5 hours of data center operations. The code for the TypedFrame test can be seen in Listing 5.1 and the schema for validating the data can be seen in Listing 5.2

```
start = time.time()
# Validating the dataframe stored in df
data = DataSchema(df)
end = time.time()

# Display validation time for TypedFrame
print(f'TypedFrame: {end - start}')

```

Listing 5.1: Code for determining validation speed of TypedFrame

```
class MetricSchema(TypedDataFrame):
    schema = {
```

```
        "hostid": int,
        "host": str,
        "status": int,
        "itemid": int,
        "name": str,
        "units": str,
        "item_status": int,
        "value_type": int,
        "clock": int,
        "ns": int,
        "value": str
    }

```

Listing 5.2: TypedFrame schema for validating example data

The time complexity was measured similarly to the running time, but with increasing input sizes. The data was of the same format as the set used for testing running times, and sampled to select subsets of different sizes. The running times could then be plotted against the input sizes and the graph can be compared to the theoretical time complexities of the tools. The code for testing the time complexity of TypedFrame is shown in Listing 5.3. Note that in the example code, more data was added to the dataset, as TypedFrame was very fast and lower input sizes gave sporadic output times. The schema is the same as seen in Listing 5.2.

```
for i in range(1, 41):

    # Select a subset of entries
    entries = i*1000000
    df_sample = df.sample(n=entries)

    # Validate the subset
    start = time.time()
    data = MetricSchema(df_sample)
    end = time.time()

    # Display the running time and number of entries in the dataset
    print(f'{entries} {end - start}')

```

Listing 5.3: Code for testing time complexity of TypedFrame

Measuring accessibility was done less objectively than the previous two measurements. A rough measure was determined to be the amount of input required to validate a dataset. To begin with, a simple data format was created for a small dataset, containing a host ID, a name, a timestamp, and a value. Then, the number of characters, words, and lines required to create a simple validation schema for the dataset were counted and combined to give a general accessibility score to the validation interface. Examples of the validation schema used for each tool can be seen in Listing 5.4 to 5.9. Further, the output of the tools after a failed validation was also considered. The failed validation messages were checked for specificity, if the entries that caused validation to fail were specified or not.

Pydantic

```
class ValidationSchema(BaseModel):
    hostid: int
```

```
    name: str
    timestamp: datetime
    value: float

```

Listing 5.4: Pydantic schema

Great Expectations

```
data.expect_column_values_to_be_of_type('hostid', 'int')
data.expect_column_values_to_be_of_type('name', 'str')
data.expect_column_values_to_be_of_type('timestamp', 'datetime')
data.expect_column_values_to_be_of_type('value', 'float')

```

Listing 5.5: Great Expectations expectation suite

TypedFrame

```
class ValidationSchema(TypedDataFrame):
    schema = {
        "hostid": int,
        "name": str,
        "timestamp": datetime64,
        "value": float
    }

```

Listing 5.6: TypedFrame schema

Marshmallow

```
class ValidationSchema(Schema):
    hostid = fields.Integer()
    name = fields.String()
    timestamp = fields.DateTime()
    value = fields.Number()

```

Listing 5.7: Marshmallow schema

Pandera

```
schema = pa.DataFrameSchema({
    "hostid": pa.Column(int),
    "name": pa.Column(str),
    "timestamp": pa.Column(np.datetime64),
    "value": pa.Column(float),
})

```

Listing 5.8: Pandera schema

Table 1 Example entry of testing dataset

hostid	host	status	itemid	name	units	item_status	value_type	clock	ns	value
10700	eselda13u11s05	0	432299	interface bond0.4036: Operational status	NaN	0	3	1691020800	76226	0

JSON Schema

```
schema = {
    "title": "Validation Schema",
    "type": "array",
    "items": {
        "properties": {
            "hostid": {
                "type": "integer",
            },
            "name": {
                "type": "string"
            },
            "timestamp": {
                "type": "date-time"
            },
            "value": {
                "type": "number"
            }
        }
    }
}

```

Listing 5.9: JSON schema

Finally the tools were evaluated on flexibility. This was done by evaluating the input required to accommodate a dataset different from the schema. Similar to the accessibility measurement, the number of words required to validate a new dataset was counted and compared.

5.1.2 Data Profiling

Running time was once again measured by simply running the tools on a large dataset, recording the time right before and after and subtracting the start time from the end time. The profiling was done on a sample of the dataset in Table 1. The sample contained $300,000$ entries.

A problem was discovered with SweetViz during the evaluation. No column in the dataframe can have the name “value”. This was fixed simply by checking the names of the dataframe columns and changing any with that name. A problem also arose when profiling datasets of about $100,000,000$x11 entries, with Ydata Profiling. The tool would crash when it was run, presumably because of high memory usage. No simple fix for this was found, so the eventual solution was to sample the dataset if it was too large. This would not give as accurate of a representation of the data as using the whole dataset, and so should it be deemed a waste of resources, the tool may be removed in the future.

The time complexity was analyzed by running the tools 100 times, increasing the size of the input each time. The running times for each input were plotted against the number of

entries each run and the graph could then be examined to determine the time complexity of the tool. The code for the Ydata Profiling evaluation can be seen in Listing 5.10.

```
1
2  for i in range(1, 101):
3      entries = i*3000
4      df_sample = df.sample(n=entries)
5
6      start = time.time()
7      # Time series mode and mininal mode are turned on
8      profile = ProfileReport(df_sample, tsmode=True, sortby="clock",
       minimal=True)
9      # The profiling report must be displayed in order for the majority of
        calculations to be done
10     profile.to_notebook_iframe()
11     end = time.time()
12
13     print(f'{entries} {end - start}')
14
```

Listing 5.10: Code for testing time complexity of Ydata Profiling

Accessibility was measured differently from the validation tools. The tools were compared on what input each required to generate a profile of a dataset. These were compared by running the tools on a sample of the same dataset as presented in Table 1 and comparing which tools required the least amount of input. Finally, the outputs were evaluated to determine how easily the information in the reports could be reached and if it was explained.

Finally flexibility was measured by running different datasets through the tools. First, the tools were run with the dataset seen in Table 1, then data was removed, leaving missing values in multiple columns, and then the datatypes of columns were changed. Secondly, the options for customizing the output of tools were examined. This was done through study of the documentation and testing of features.

5.1.3 Feature Extraction

The secondary evaluation was done similarly to how it was done for the data profiling tools. Running time was tested on a sample of the data shown in Table 1, consisting of $100,000$ entries. And time complexity was evaluated the same way, by timing when the tools were run with different inputs.

Accessibility was evaluated in the same way as it was for the profiling tools, by analyzing the amount of input required to extract features from a dataset. And flexibility was also evaluated similarly. First, data of different types, with different amounts of missing values was tested on the tools to see how flexible they are regarding input. Then, the tools were tested to determine how much the output could be customized.

5.2 Building the Pipeline

Some assumptions were made when building the system. First, in order to limit unnecessary work, files uploaded must be in a dataframe format. This could change in the future but that is outside the scope of this project. Second, all datasets uploaded must be time series data,

similar methods could be used on other types of datasets, but it is outside the scope of this project. Third, the timestamp of the time series data is provided by the user.

Building the pipeline started by considering generality and robustness. Testing was done often with different data files containing data of different shapes and sizes. A lot of safeguards were added to catch as many potential errors as possible. Should anything go wrong it must be explained to the user so that they can alter their input or report the error accurately.

The tools chosen after the evaluation were integrated into the system. Real integration into the portal backend is handled by the Ericsson engineers. The implementation part of this project handles building and testing the pipeline in the portal through the Jupyter Notebook interface that it provides, as this lets the sample datasets already in the portal be tested on the pipeline.

5.2.1 Pipeline Stage One

The first stage of the pipeline performs a simple analysis to determine the size of the new datasets along with the datatypes it contains. The user is able to view these features and confirm that they are as expected. If they are not, the user should be able to modify whichever aspect is incorrect, and the system should do its best to adapt to the modifications.

This stage of the pipeline was built on the validation tool selected after the secondary ranking of the tools. It begins by loading the dataset into the system as a dataframe along with letting the user specify the timestamp column in the dataset. The dataframe is then displayed to the user before it is passed through a for loop that checks the types of each column and generates a file containing a validation schema of the data. The user may then modify the file if needed before the dataset is finally validated using the generated schema in the file. Tryexcept statements were added to stop empty datasets, and datasets of the wrong type from being uploaded.

5.2.2 Pipeline Stage Two

The second pipeline stage was very straightforward to construct as it consists entirely of the data profiling tools that were chosen. The dataframe that was validated in the previous step was passed to the data profiling tools that created and displayed profiles of the data. The profiles were then saved as HTML files that the user could view.

Depending on the size of the dataset, the profiling may take some time, so the third stage of the pipeline can start before the second has finished, as it is entirely independent. Once the second stage was finished, the data profiles that were generated were displayed for the user, and the pipeline moved to the final stage.

5.2.3 Pipeline Stage Three

The third stage of the pipeline provides a more in–depth analysis of the data and puts a label on the dataset based on the automatic analysis it performs. In this stage, the analytical metadata is created.

Before the analysis was done, the data was cleaned, by removing missing values. First, if any column contained no values, it was removed. This threshold could be changed by the

user if they, for example, would want to remove columns that are missing 80% of values. If any columns were still missing values, they were interpolated using the Pandas interpolation feature. Missing values were filled in using surrounding values. After removing missing values, the data was encoded to enable analysis by the chosen tools. Encoding was done using the Sklearn label encoder in order to convert any non–numerical values to numbers. Label encoding was chosen over one hot encoding, as, if the data is large, with many unique categories, it may take a lot of extra space to store the data and extra time to process.

Once the data was filled in and encoded, depending on the size of the dataset, it was sampled. Because some of the analysis can be very slow and memory intensive, if a dataset had more than $20,000$ entries, a random sample of that many entries was selected for analysis. The data, either the sample or entire dataset depending on size, was then passed to the tools that calculated a multitude of features of the dataset.

For further analysis, after the feature extraction tools were run, the data was reduced. Because the pipeline processes large datasets, it can be assumed that much of the data passing through the pipeline is high dimensional. Reducing the dimensionality of the data allows for easier visualization and improved performance. First, the data was reduced with principal component analysis (PCA) using Sklearns PCA function. Principal components explaining 80% of the variance in the data were kept as that would explain most of the variance while hopefully removing the least important values. The remaining data was then embedded into three dimensions using t–distributed stochastic neighbor embedding (tSNE) so it could be visualized. Both PCA and tSNE were selected to reduce the dataset as much as possible while trying to preserve the underlying patterns in the original data and making the data viewable in two or three dimensions.

The t–SNE embedding of the data is then presented in a three–dimensional graph for the user to explore. In order to find groupings, the data is then clustered using Sklearns k–means clustering algorithm. The clustering is run 10 times with k values increasing from 1 to 10, and the inertia of each clustering is used to construct an elbow graph. The graph is presented to the user along with an explanation so that they may select an appropriate k value either based on the graph or on domain knowledge they possess. The data is then presented again in a three–dimensional graph, this time showing the clusters. Finally, the distances from the data points to their cluster centers are calculated, and the data is plotted again, with increasing size the further from the cluster centers, in order to visualize the spread of each cluster. During this process, the silhouette score and within–cluster sum of squares are calculated and saved.

In order to show the user what data may have contributed to each cluster, a random forest is used. It is trained on the sample data that was used to create the clustering, and the labels of the clusters to find similarities. The importance of each column to the clustering is plotted to give the user an idea of what columns contribute the most to the clustering. Finally, the correlation is calculated, and the correlation matrix for the data is shown using the Pandas correlation function. The absolute mean value of the correlation and the mean–variance of the dataset are both calculated and saved.

Once all the features have been calculated and visualized, the statistical metadata is passed to the labeling model, which provides labels that give a general overview of the dataset. Specifically, it describes the spread, correlation, and clustering of the data in simple terms. The spread is determined based on the variance of the dataset, and is expressed as “High variance”, “Medium variance”, or “Low variance”. The correlation is described by the ab-

solute mean of the correlation matrix of the data. By removing the trivial diagonal and summing the absolute values of the top half of the correlation matrix, a metric for the correlation across the dataset is determined. The correlation label can be: “Strong correlation”, “Reasonable correlation”, or “Weak correlation”. Clustering is determined by two values: the silhouette score and the within–cluster sum of squares (WCSS). The silhouette score determines the strength of the clustering. It can be “Strong clustering”, “Reasonable clustering”, “Weak clustering”, or “Bad clustering”. Finally, the quality of the clusters is determined by WCSS. The value is divided by the size of the dataset, to scale it down and make the differences clearer. The cluster quality can be “Tight clusters”, “Regular clusters”, or “Scattered clusters”.

Initially, the model was created using a Pytorch neural network, however, despite extensive testing and tuning, the model never achieved an accuracy above 60%. Instead, a random forest classifier from Sklearn was used. Training data was obtained by generating randomly clustered data, calculating the necessary features, and hand labeling it. Labeling was first done by observing, visualizing the data, and applying an appropriate label. When a number of labels were created, more data was generated and labels were applied based on the previous labels. The number of estimators and size of training data was tweaked until an accuracy of 99% was reached. Such high accuracy could imply overfitting. However, because the output data was relatively small, the results were satisfactory.

6 Results

As described in Section 4 the tools were divided into three categories, validation, profiling and feature extraction, and they were evaluated within these categories.

6.1 Evaluation of Validation Tools

The initial ranking was done with regards to the metrics listed in Section 4.1. They were available resources, the focus of the tools, as well as the validation–specific metric of features provided for validation beyond type checking.

The resources were judged on the size of the community around the tool. A rough measure of the size of the community was created by observing the number of people who have starred the tool, and watch the tool on GitHub. Using this to show the size of the community surrounding the tool, serves as an indicator of the size of the potential pool of resources behind the tool. The goals and features were determined from the descriptions of the tools, as presented in Section 3.9. The tools were ranked on each metric individually before each ranking was combined, giving the resulting initial ranking in Table 2.

TypedFrame has a very small community compared to the other tools. The stated goal of the tool is not entirely in line with the purpose it would serve in the pipeline either. Marshmallow has an explicitly different focus than the other tools, and JSON Schema serves a slightly different purpose as, at the moment, incoming data is not in a JSON format.

Pydantic and Great Expectations have large communities and a large number of resources to draw from when learning about them. However, while Great Expectations is mostly in line with the needs of the pipeline, Pydantic is not entirely. Both tools provide a multitude of extra features, specifically, Great Expectations provides methods for integrating validation into data pipelines, which is of interest.

Pandera does not have as much interest as most of the other tools, however its stated focus falls well in line with what it is needed for. Pandora also provides a large number of features and a large number of datatypes for type checking and type conversion. Specifically, it provides features for pipeline integration and deeper validation along the lines of Great Expectations.

For the secondary ranking, the running time, time complexity, accessibility, and flexibility were examined. To test the validation speed of the six validation tools, they were run on a dataset containing $8,095,419$x11 entries of operational data from the Ericsson Research data center. An example entry can be seen in Table 1. An example of a TypedFrame schema for the dataset is displayed in Listing 5.2. The example shows the expected datatypes for each column of the dataset.

To get a fairer measurement of running time, each tool was run 10 times, and the average

Table 2 Initial ranking of validation tools

Rank	Tool	Community	Focus	Extra features
1	Great Expectations	9582	Data processing pipeline validation for known datasets	Many
2	Pydantic	19014	Low level data validation for Python programming	Many
3	Pandera	3048	Data processing pipeline validation	Many
4	JSON Schema	4511	Validation of JSON objects	Many
5	Marshmallow	6980	Serialization and deserialization of data	Many
6	TypedFrame	94	Validation of Dataframe objects	Few

Table 3 Speed and accessibility test results

	Time (s)	Accessibility
Pydantic	68.1	148
Marshmallow	242.3	306
Pandera	8.3	310
GX	5.3	647
TypedFrame	0.00057	173
JSON Schema	442.5	401

time was calculated. To gain a somewhat concrete measure of accessibility, the number of characters, words, and lines in a tool schema were counted and summed to give an accessibility score. This was done by taking only the names of the columns and the datatypes to remove the noise from class names and descriptions. This means that the accessibility score for the schema in Listing 5.2 was calculated without line 1. Both the time and accessibility results are displayed in Table 3.

Testing of the time complexity of the tools was done similarly to the running time tests. Each tool was run with increasingly larger datasets, and the times were recorded. The resulting graphs are shown in Figures 2.

To evaluate the flexibility of the tools, the syntax for adding additional columns was examined and the number of additional words required for making a column optional was counted. Results for each tool can be seen in Table 4. Any additional columns must be optional so as not to invalidate the previous data format.

Based on the initial ranking and the results of the tests, the tools were ranked a second time. The results of the second ranking can be seen in Table 5.

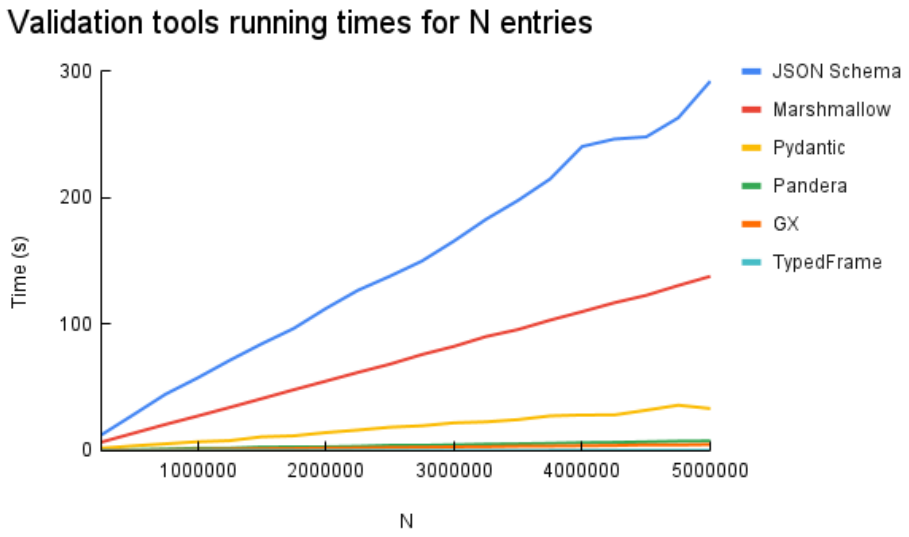

Figure 2: The running times of the data validation tools with increasing input size.

Table 4 The amount of extra input required to create an optional column to a schema

Tool	Input required to make column optional
Pydantic	1
Marshmallow	2
Pandera	2
GX	0
TypedFrame	1
JSON Schema	0

Table 5 Second ranking of validation tools

Rank	Tool
1	Pandera
2	Pydantic
3	Great Expectations
4	TypedFrame
5	JSON Schema
6	Marshmallow

Table 6 Initial ranking of profiling tools

Rank	Tool	Community	Focus	Features
1	Ydata Profiling	12150	Easy data profiling	Data summarization values
2	SweetViz	2892	Easy data profiling	Data summarization values
3	Dython	405	Easy data analysis	Categorical and numerical associations

Table 7 Running times of data profiling tools on a dataset with 300,000 entries.

	Time (s)
SweetViz	11.9
Dython	14.5
Ydata Profiling	99.5

6.2 Data Profiling Tools

The initial ranking of the data profiling tools was done partly on the general metrics of community, and stated focus of the tool, as well as the specific metric of extracted data. Table 6 shows the results of this ranking.

Dython is the smallest tool, and so predictably has the smallest community surrounding it. The focus of the tool is data analysis in an accessible manner which is an important part of the pipeline. Its primary function is calculating the associations between the variables in the dataset using, Pearson correlation, Correlation ratio, Cramérs V, and the Uncertainty coefficient, as described in Section 3.9. It also provides an interface for calculating a multitude of features for exploring machine learning models.

Ydata Profiling and SweetViz are relatively similar in function. The focus of both tools is specifically generating a comprehensive data profile that can be viewed in the form of a report. Ydata Profiling had by far the largest community of them, but functionally they provide very similar features.

For the secondary ranking, the running time and time complexity tests were performed the same as for the validation tools. The results can be seen in Figure 3. For the tests Ydata Profiling was run on datasets with up to 300,000x11 entries, over 100 runs. It was run in time series mode to represent its use in the system. It was also run in minimal mode, disabling the associations it calculates, as without it, profiling large datasets would sometimes cause memory issues. Time complexity analysis of SweetViz and Dython was also done using a set of 300,000x11 entries over 100 runs. The Dython tests were run on the Association function of the tool, as that would be the main one used if the tool were chosen. Running time tests were performed by running the tools on a dataset with 300,000x11 entries. The results are presented in Table 7.

Accessibility evaluation was done by analyzing what information was required to create a profile of the data. The comparison between the input of the tools can be seen in Listing 6.1. Example outputs of Ydata Profiling, SweetViz, and Dython can be seen in Figures 4 to 9. Both Ydata Profiling and SweetViz have similar presentations with simple layouts that are

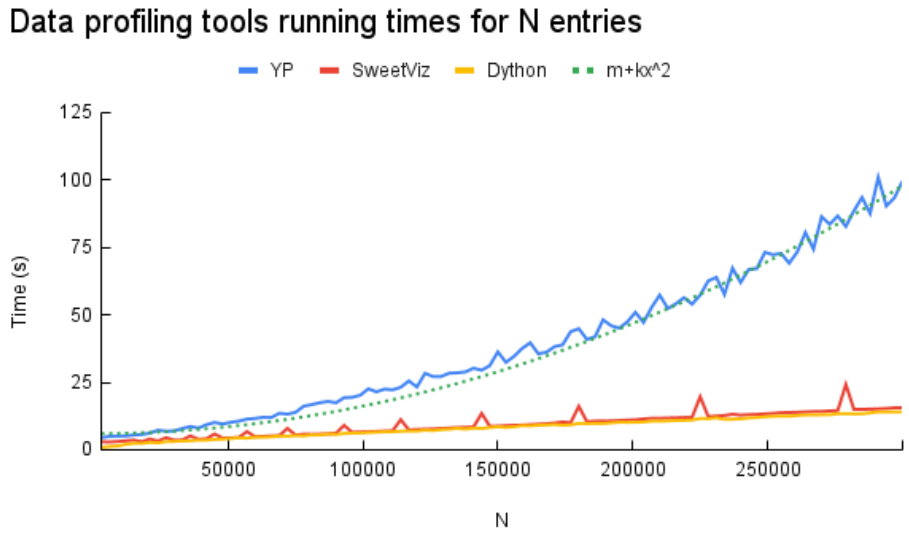

Figure 3: Running times of data profiling tools with increasing input size.

highly readable. Each column has its statistics presented individually where each calculated statistic is named. Ydata Profiling provides labels for certain patterns it can observe in a column. However, the labels are not explained in the profile and require the user to read the documentation for explanation. Dython has lower accessibility as it presents a correlation matrix that it does not explain. Different methods are used depending on if the column contains continuous or categorical values, but it is not explained in the interface, and also requires the documentation to understand.

```
# Create and display profile report using Ydata Profiling
profile = ProfileReport(df, tsmode=True, sortby="timestamp")
profile.to_notebook_iframe()

# Create and display profile report using SweetViz
report = sv.analyze(df_validated)
report.show_notebook()

# Create and display association graph using Dython
associations(df_sample)

```

Listing 6.1: Code for running and displaying profiles using Ydata Profiling, SweetViz, and Dython

Finally, the flexibility testing was done using the same datasets previously mentioned, with modifications. It shows that all tools could handle missing values. Ydata Profiling was the only tool not to crash when presented with unsupported datatypes. Instead, it marks the column as unsupported. Ydata Profiling and SweetViz both have highly customizable output. Ydata Profiling lets users add configurations to the input when creating the profile, while SweetViz lets users provide inputs to the config file it uses to create a profile. Both let users remove or keep whichever statistics they wish. Dython has the least customization, simply by having a less varied analysis. Users may choose which functions to use, however, for a good overview of the data, the association function is the only real option. Using the initial ranking and the results shown in this section, the second ranking was performed. The

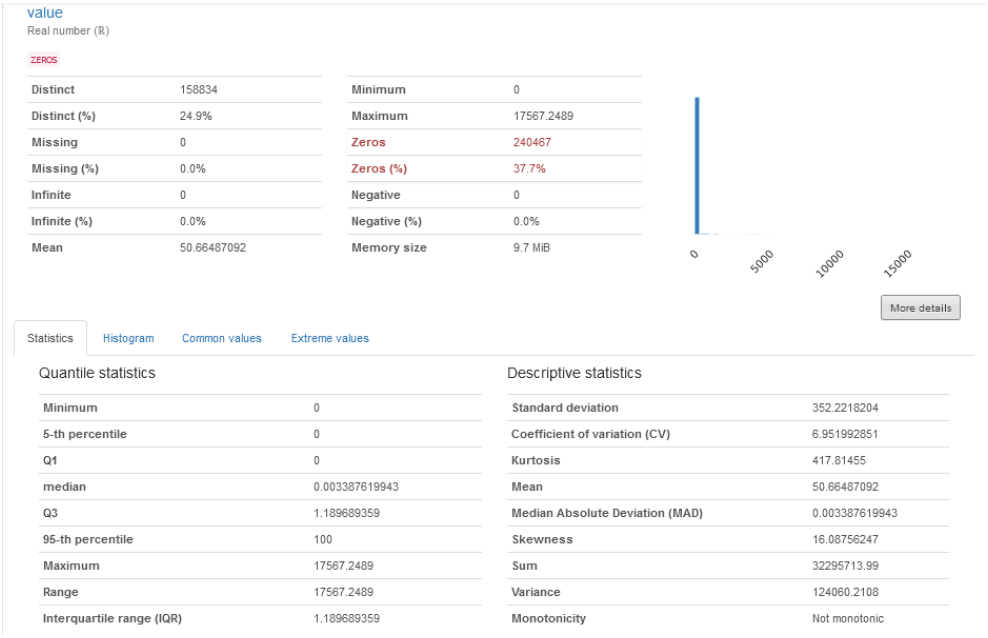

Figure 4: Example of Ydata Profiling profile for "value" column from data as shown in Table 1.

Figure 5: Example of data alerts given for each column by Ydata Profiling, for data shown in Table 1.

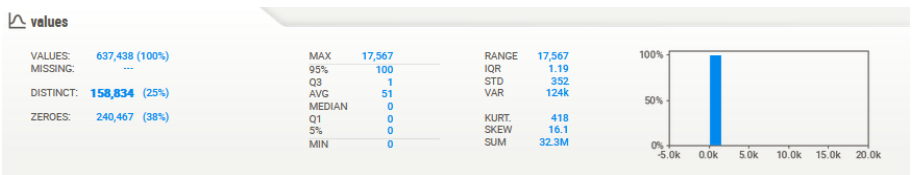

Figure 6: Example of collapsed profile of "value" column for data shown in Table 1, given by SweetViz.

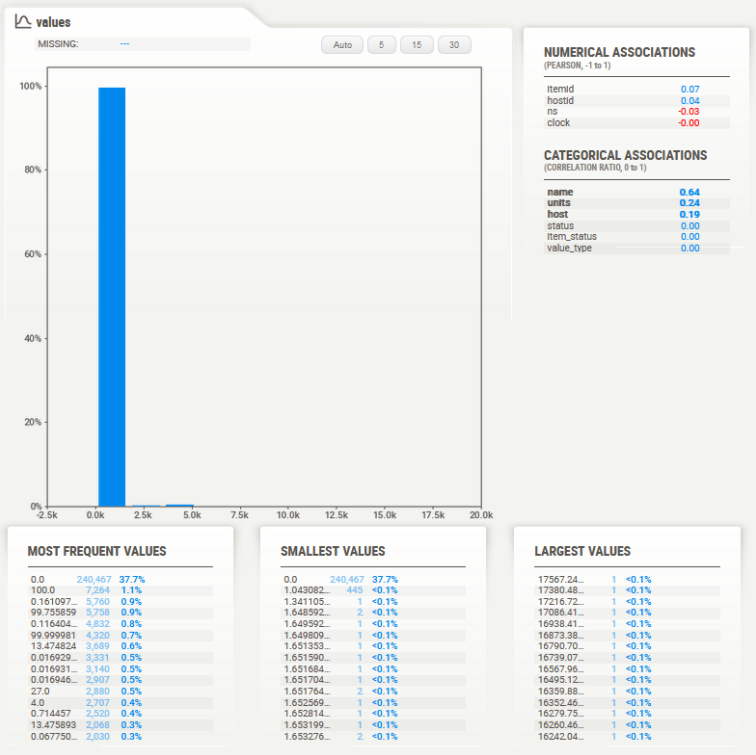

Figure 7: Example of extended profile of "value" column for data shown in Table 1, given by SweetViz.

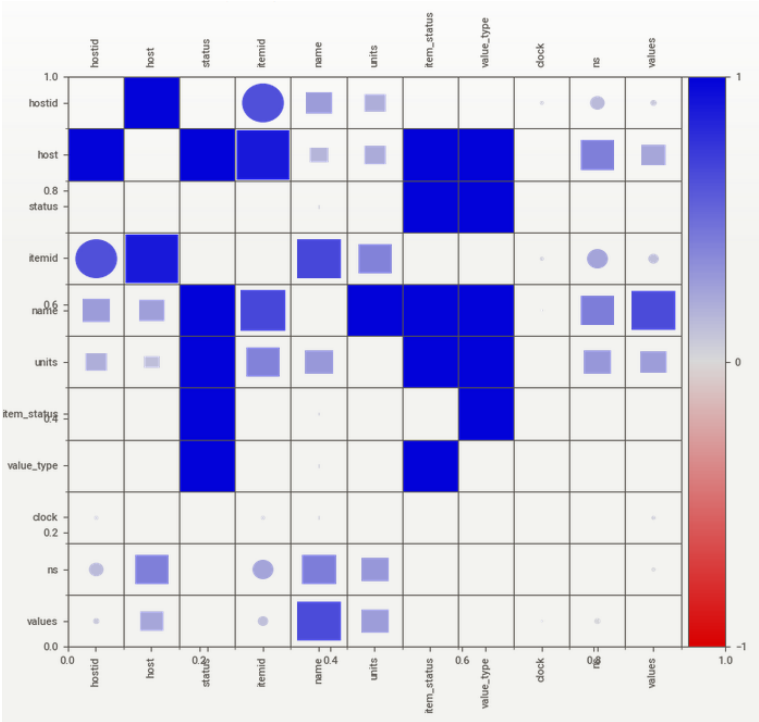

Figure 8: Example of association graph for data shown in Table 1, given by SweetViz.

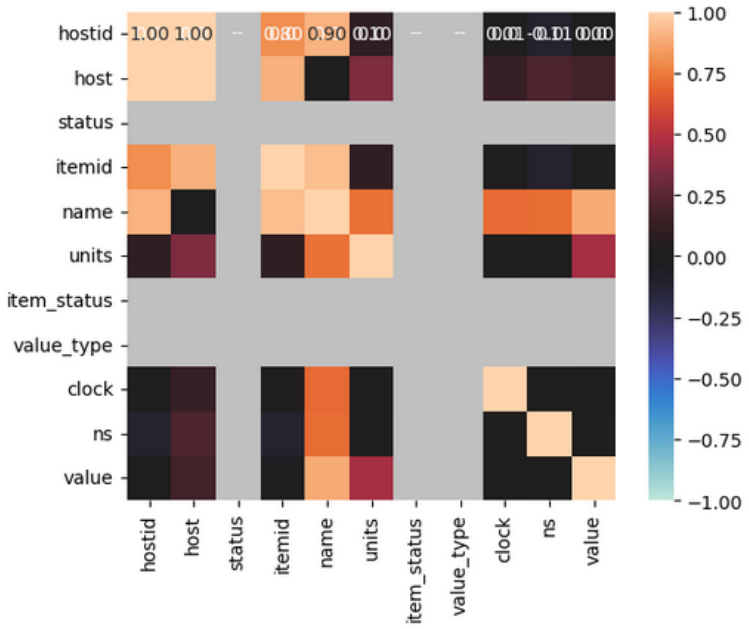

Figure 9: Example of association graph for data shown in Table 1, given by Dython.

Table 8 Second ranking of data profiling tools

Rank	Tool
1	SweetViz
2	Ydata Profiling
3	Dython

resulting ranking is shown in Table 8.

6.3 Feature Extraction Tools

Just as with validation and profiling, the initial ranking was done on available resources and focus of the tool. The ranking also took into account the specific features extracted. The results of the ranking are presented in Table 9.

Stumpy has a somewhat large community and is well–supported. However, the tool's focus

Table 9 Initial ranking of feature extraction tools

Rank	Tool	Community	Focus	Features
1	Tsfresh	8268	Easy extraction of time series features	Calculates 77 general statistical features
2	Tsfel	880	Easy extraction of time series features	Calculates 68 general statistical features
3	Stumpy	3058	Calculation of the matrix profile	Calculates the matrix profile

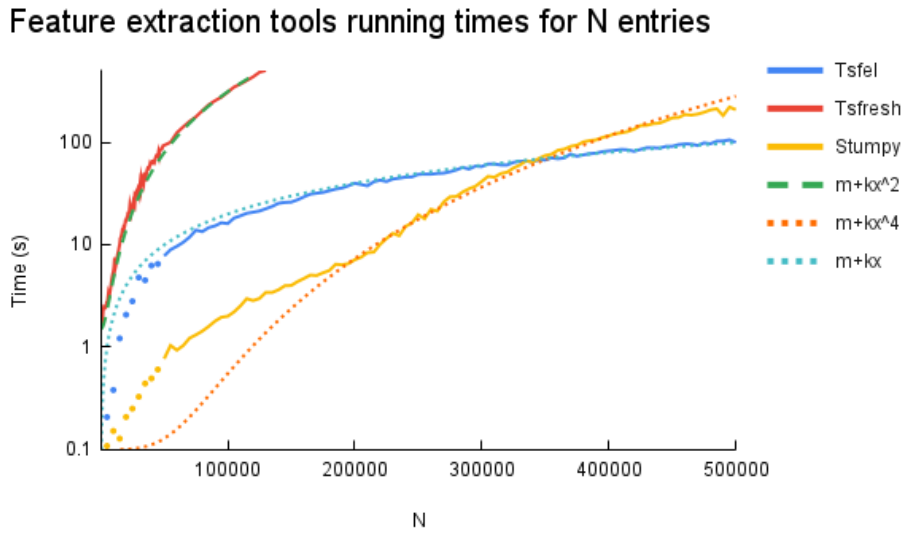

Figure 10: Running times of feature extraction tools with increasing input size plotted on a logarithmic scale.

Table 10 Running times of feature extraction tools on a dataset with 100,000 entries.

	Time (s)
Stumpy	2.0
Tsfel	16.1
Tsfresh	306.0

and the specific features it calculates are very specific. Compared to Tsfresh and Tsfel, it does not provide a wide range of features, thus placing it low in the initial ranking.

Tsfresh and Tsfel are very similar in terms of features extracted and focus. However, Tsfresh has a much larger community and more resources available. Both focus on analyzing time series data and calculating features that may be relevant for machine learning models to understand the data.

Time complexity and running time were analyzed similarly to the validation and profiling tools. The results can be seen in Figure 10, presented in a logarithmic plot for easier comparisons. Testing of Stumpy was done using a dataset of 300,000x11 entries, on one column of numerical data to represent a data column in a dataset. It was done with a window size of 300. Tsfresh and Tsfel were evaluated with datasets containing 50,000 and 500,000 entries, respectively. The running time tests were done on a dataset of size 300,000x11.

Accessibility was evaluated similarly to the data profiling tools by analyzing the amount of input beyond the dataset required to run the tool. Examples of this can be seen in Listing 6.2. A rough metric for ease of use can be observed by evaluating the amount of required input.

```
# Extract features from df using Tsfresh
extract_features(df, column_id=id, column_sort=timestamp)

# Extract features from df using Tsfel
cfg = tsfel.get_features_by_domain()
```

Table 11 Results of flexibility testing on Tsfresh, Tsfel, and Stumpy

Handles	Tsfresh	Tsfel	Stumpy
Missing values	No	No	Yes
Non–numeric values	No	No	No

Table 12 Second ranking of feature extraction tools

Rank	Tool
1	Tsfel
2	Stumpy
3	Tsfresh

```
tsfel.time_series_features_extractor(cfg, df)

# Calculate matrix profile from df "values" column using Stumpy
windowsize = 24
stumpy.stump(df['value'].values, windowsize)

```

Listing 6.2: Code for calculating features using Tsfresh, Tsfel and Stumpy

The flexibility evaluation was done by testing input data with missing values and non–numerical data. It revealed that none of the tools are very flexible regarding input. They all require the input data to be numeric without missing values. Any unsupported datatypes also cause the tools to give an error. Changing the data shape is not a problem. Table 11 shows the results of the tests. Tsfresh lets users customize precisely which features they want to extract by adding a dictionary, specifying the desired ones. Tsfresh does as well by letting users create a dictionary of features they wish to calculate.

All results contributed to the final ranking of the feature extraction tools, which can be seen in Table 12.

6.4 System Evaluation

Testing the system's running time was done the same way as the other tools. Input size started at $25,000$ entries, in increments of $25,000$ up to $950,000$ entries, as memory problems with Ydata Profiling prevented the size from increasing further. The results can be seen in Figure 11. The running time of the system, on input data with $900,000$x11 entries, was: 1029.5s

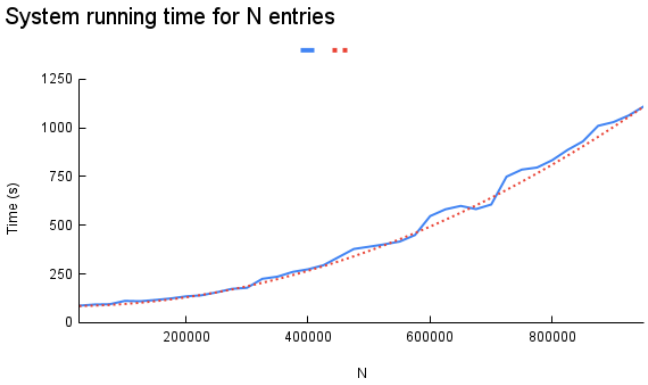

Figure 11: The running time of the system for N entries.

Accessibility was tested by examining what input is required and what explanations were provided for the output. Beyond the dataset, the system requires the dataset's timestamp column as input. Further on, the system also provides the option of selecting a k value for clustering. All outputs are explained by the system. Finally, flexibility was tested using datasets with missing values, as well as empty datasets and datasets with columns added and removed, all of which the system can handle to different extents. Updating datasets required some further input from the user to add optional columns. This evaluation ensures that the system is robust and reliable. Together, these results provide points of comparison with the top ranked tools, shown in Tables 5, 8, and 12.

7 Discussion

For the first aim of the research question, the tools were studied and evaluated solely on the information provided before testing. This created the initial ranking based on what each tool could potentially provide to the system. Categorizing the tools was relatively simple. The validation tools were easy to identify and group together as they all serve a similar purpose. For the data profiling tools, Ydata Profiling and SweetViz were very obviously similar. However, Dython could have theoretically been placed in the feature extraction category, but because it functions by providing visualizations of a dataset, it was judged as falling toward the data profiling category. The feature extraction tools were also quite simple to categorize. Stumpy is quite different from Tsfresh and Tsfel, however it can still be categorized with them as it extracts a feature of the dataset, namely the matrix profile.

One potential problem for evaluating the tools with metrics for evaluating metadata extraction tools, is the fact that metadata tools vary a lot. The specific metrics referenced in Section 3.2 were specifically used for metadata generation tools, which are more comprehensive and usually more focused on a specific domain than any of the tools analyzed in this project. However, they provided a good basis for identifying evaluation metrics for the tools used. Running time and time complexity give a good idea of the speed of the tools. Accessibility is referenced in Section 3.2 and is important as it lets new and unfamiliar users access and use the tools effectively as quickly as possible. Finally, flexibility is an important metric as datasets continue to grow. Inevitably, data format, type, and shape change, and tools should be able to manage that in order to be useful.

7.1 Designing the Data Processing System

Regarding aim two of the research question, the design of the data processing system was based entirely on the needs of WARA–Ops. For the final labeling of the data, creativity is the only factor that limits what metrics can be selected. It was determined that a good start for the system should be a summary of the dataset. The basic summarization metrics selected were determined to be, variance and correlation based on the explanations in Section 3.5. To examine underlying patterns, clustering of the dataset using k–means clustering was determined to be appropriate for analysis. This provides information on the spread of the data, along with possibly identifying underlying patterns.

Training the labeling model was relatively simple. The input has only four values, and because the training data was labeled somewhat rigidly, not much tuning was needed to achieve a high degree of accuracy. Presumably, the model is somewhat over tuned. However, the immediate fix for this is better training data. The training data used was labeled by hand, so generating an appropriate amount of it required some amount of rigidity in accordance with the input values. Higher quality training data is likely more ambiguous about which labels are applied to which values.

7.2 Results of Tool Evaluations

Beginning with the validation results, they are as expected. Because validation tools run over the input data, checking the type of each one, they are expected to exhibit running times linearly increasing with input size. This behavior is present in the results. While it is good that they meet expectations, it does not provide good grounds for the secondary ranking, as all tools are the same. The running times, on the other hand, provide better grounds for ranking. The running time results are also expected. TypedFrame is simply a wrapper over a dataframe, that provides very few features, and so has very little overhead, making it very fast. Great Expectations is also designed to process large datasets so it can be expected to be partly optimized for speed. Pandora, while being somewhat focused on larger datasets, is also used for lower–level type–checking in programming, meaning it may be less focused on performance. Pydantic, is also a lower level tool and so is not focused on execution speed. Pydantic, along with Marshmallow and JSON Schema, does not validate dataframe objects directly either. Because the input for the speed tests was converted from a dataframe, it is possible that the structure of the converted dataframe may not be optimal for validation.

The accessibility and flexibility tests are rougher than the time measurements but provide a good basis for comparing the tools' interfaces. It becomes clearer when comparing the syntax of the tools directly. TypedFrame, Pandas, Pydantic, and Marshmallow all have quite clear syntax that show the name of the column along with the datatype. However, Pandera and Marshmallow require some more input that may be clarifying to someone familiar with the tool but may be confusing to a new user. JSON Schema has a quite clear syntax that is accessible and flexible but can grow very quickly. For large datasets, writing and checking a large JSON schema could become cumbersome. Great Expectations syntax shows more of what the focus of the tool is. Because it uses specific expectations, it is assumed that the data format is known, and simple type–checking is not a focus. Based on these results, together with the initial ranking, it is clear that Pandera is the most appropriate. Considering that it can be used with the syntax of Pydantic, it is more accessible than the other tools while still providing effective validation with a multitude of features that can be added, should the need to be.

The time complexity graphs of the profiling tools were somewhat unexpected. SweetViz calculates the association graph, that is, $O(n^2)$ at worst, however the SweetViz graph shows a relatively linear increase. This is most likely the result of the dataset used for testing. A dataset with larger dimensions would most likely incur an increase in running times. This is also visible in the Dython graph, as it shows a linear increase despite calculating the associations of the dataset. Ydata Profiling, on the other hand, was run in minimal mode, which stops it from calculating the associations between columns. However, its graph exhibits $O(n^2)$ performance, as can be seen by the dotted line in the graph. The increased running times may come from time series specific calculations, as time series mode was enabled. The running times of the tools were not entirely expected. SweetViz and Dython both create an association graph, but SweetViz also constructs a profile report. Despite this, it is faster than Dython.

All the tools were quite flexible, as part of their purpose was to analyze new data. They could all handle data with missing values, however, while Dython and SweetViz gave errors when encountering an unsupported datatype, Ydata Profiling could still construct the profile while marking the column as unsupported. The output of SweetViz and Ydata Profiling

was also customizable to a high degree. Specific features and columns can be blocked from analysis in both tools. The association graphs could also be enabled and disabled in both. Finally, the tools were very accessible. The only input required for all of them is the dataset itself, but when run in time series mode, Ydata Profiling also requires the timestamp of the dataset. As can be seen in Figure 8, SweetViz provides the association graph for the dataset, with an explanation of it. Figure 4 shows the labels put on the data by Ydata Profiling. While insightful, the labels are not explained in the report, which may cause problems for more ambiguous labels such as "Skewness" and "Imbalance" where the thresholds for determining the label are not given. Using these results, the secondary ranking was determined, however, because Ericsson has the resources, both SweetViz and Ydata Profiling can be run in parallel with the rest of the system, so both tools were chosen.

Finally, the feature extraction tool evaluation was interesting. Tsfel and Tsfresh extract very similar features. However, Tsfresh uses the Id columns of the dataset and calculates features for every time series in the dataset, which is reflected in its longer running times. This can partly explain the time complexity analysis, as the running time increases when more IDs are added to the dataset. Stumpy has a theoretical time complexity of $O(n^2)$. However, in Figure 10 it exhibits a much steeper curve, closer to $O(n^4)$. This may have been caused by a deprecated package in the portal that slows down the tool. No simple solution to this was found. However, even going by the normal complexity of $O(n^2)$, Stumpy still ranks below Tsfel.

The only feature extraction tool that is as accessible as the profiling tools is Tsfel. As shown in Listing 6.2, it requires the dataset and a config file as input. However the config file may left as is and do not need to be handled by the user, although should the user wish, it may be modified to extract specific features. Tsfresh requires the ID column of the dataset, requiring further input from the user, and Stumpy requires some domain knowledge and extra input to select an appropriate window size. Flexibility results were expected. Both Tsfel and Tsfresh require clean, numerical data to perform their tasks and Stumpy, while requiring numeric values, can still perform its tasks with missing values. Based on these results, the optimal tool for integration is Tsfel. However, because Stumpy has such a specific focus, it may be a valuable tool to use as well.

7.3 System Evaluation

Because the system consists of the tools running in sequence, the time complexity of the system can be assumed to be the sum of the complexities of the tools within the system. The lower order values can then be disregarded meaning it becomes the time complexity of the highest order of the tools. This is reflected in the results shown in Figure 11. The analysis of Ydata Profiling showed a complexity curve of n^2, which is also what is shown in the running times of the system itself. One difference in the finished system is that the data profiling tools can be run in parallel with the rest of the system. This speeds up the system as a whole. The time complexity of tSNE is also $O(n^2)$, however, it is run on a sample of the dataset to reduce running time and memory usage, which means the input size to the algorithm can be controlled, and thus has less of an impact on the running time.

The accessibility and flexibility evaluation results are also expected as the system is designed specifically to be accessible and flexible. While there are several opportunities for

users to provide input, the only required input is the dataset itself and the timestamp column of the dataset. This is in line with the rest of the tools analyzed. Any problematic datasets entered are processed and analyzed to the extent possible, and most errors that occur are explained by the system.

8 Conclusion

This thesis has examined different data management tools and evaluated them on the basis of extracting statistical data and metadata from an unknown dataset. The tools were categorized and ranked based on what value they could provide to a data analysis system. The most suitable tools were chosen and integrated into a data analysis pipeline for large operational datasets, and the whole system was compared to the initial sets of tools to examine what value it added. The metrics for evaluating data analysis tools were examined. The best metrics were determined to be accessibility, letting users easily use the tool effectively. Flexibility, to ensure that the tools do not break under heavy usage as much of the data may be very variable in size, shape, and type. Running time can give a rough measure of comparison between tools, and time complexity is important for determining if a tool can be used effectively on very large datasets.

The aims and objectives stated in the introduction have been achieved. The tools were studied and ranked according to aim 1. The rankings can be seen in Section 6. Then, a data processing system was designed with the help of Ericsson Research, the tools were evaluated again on practical metrics, and the most appropriate tools for the system were selected and implemented. The construction of the system using the selected tools achieved the objectives of Aim 2, and finally, the system was evaluated on the same practical metrics as the individual tools as stated in Aim 3. The tools selected in the end were Pandera for validation as it is very accessible and flexible while providing relatively good speed. SweetViz and Ydata Profiling were chosen for data profiling as they provide accessibility and flexibility. They are slow when running on large datasets, however, if they can be run in parallel with the rest of the system, they can provide informative reports on the data together with the rest of the analysis the system outputs.

8.1 Reflection

The project was challenging but interesting as it is a field I do not have much real–world experience in. I have studied artificial intelligence and cloud systems, but I have never worked with data processing tools and operational datasets of the scale present in this project. This project has reinforced the idea that a clear goal is the most important criterion for success in a large project such as this one. At the start, the goals felt somewhat flexible and unfocused which made it difficult to focus the work. They became clearer further on in the project, which then made the work much more effective.

8.2 Future Work

Data analysis and data mining are gigantic areas and will most likely always have room for future research. Firstly this thesis does not even come close to a comprehensive list of tools. All categories: validation, profiling, and feature extraction, have a large number of tools available that may be more appropriate than any of the tools analyzed here, depending on the use case. It is also possible to be much more specific in each category. The metrics used for evaluation in this thesis provide a good basis for implementing tools into a general system, however, much work can be put into examining the tools in more specific contexts.

There is also much more to be done with the dataset summarization metrics. The metrics extracted from the data by the pipeline provide a simple overview of the data and puts a broad label on it. But there is a lot of room, both for more metrics and a deeper exploration of the metrics used. The autocorrelation, using the Pandas method is presented as is for the dataset, however, it could be more useful if the data was further processed, for example, by testing for the optimal timestep to use, the graph may provide more useful for meaningful analysis.

Most of the datasets in the portal are quite similar in that they are very scattered, with no immediately recognizable patterns. It would be interesting to base the evaluations on a more heterogeneous collection of datasets. This could provide opportunities for further exploration of the data, as well as give better grounds for the final label put to the data. As it is now it is a good feature for a data analysis system, however as the labels are based on the values calculated from currently existing systems, they might become less accurate on different datasets. The labeling also has room for more. Text may be added to describe further the spread of the data, things such as how skewed it is, or if there are many outliers. This would provide a much deeper view of a given dataset.

References

[1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. Time-series clustering – a decade review. *Information Systems*, 53:16–38, 2015.

[2] TD Ameritrade. Stumpy. https://stumpy.readthedocs.io/en/latest/, August 2023.

[3] Julian Berman. Json schema. https://python-jsonschema.readthedocs.io/en/stable/, January 2024.

[4] Francois Bertrand. Sweetviz. https://github.com/fbdesignpro/sweetviz, November 2023.

[5] Great Expectations Core. Great expectations. https://greatexpectations.io/, March 2024.

[6] A.S. Gillis, C. Stedman, and A. Hughes. Data Mining. https://www.techtarget.com/searchbusinessanalytics/definition/data-mining, February 2024. Accessed 27-02-2024.

[7] Blue Yonder GmbH. tsfresh. https://tsfresh.readthedocs.io/en/latest/, January 2024.

[8] N Gogtay and U Thatte. Principles of correlation analysis. *The Journal of the Association of Physicians of India*, 65:78–81, 03 2017.

[9] Jane Greenberg. Metadata extraction and harvesting. *Journal of Internet Cataloging*, 6(4):59–82, 2004.

[10] Hazal Gültekin. What is silhouette score? https://medium.com/@hazallgultekin/what-is-silhouette-score-f428fb39bf9a, September 2023.

[11] Ajay Kulkarni, Ryan Booz, and Attila Toth. What is time-series data? definitions & examples. https://www.timescale.com/blog/time-series-data/, Jan 2024. Accessed: 2024-02-26.

[12] Education Ecosystem (LEDU). Understanding k-means clustering in machine learning. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1, September 2018.

[13] Steven Loria, Jérôme Lafréchoux, and Jared Deckard. Marshmallow. https://marshmallow.readthedocs.io/en/stable/index.html, March 2024.

[14] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. Deepeye: Towards automatic data visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 101–112, 2018.

[15] Abdullah Mueen, Eamonn Keogh, Qiang Zhu, Sydney Cash, and M Brandon Westover. Exact discovery of time series motifs. volume 2009, pages 473–484, 04 2009.

[16] Shaun M. Oborn and D. Garlick. Effective visualization tools for large data sets. 2178, 1994.

[17] Xavier Ochoa and Erik Duval. Automatic evaluation of metadata quality in digital libraries. *Int. J. on Digital Libraries*, 10:67–91, 08 2009.

[18] Oracle. What is big data?, 2024. `https://www.oracle.com/big-data/what-is-big-data/`.

[19] The pandas development team. pandas-dev/pandas: Pandas, 2024.

[20] Jung-ran Park and Andrew Brenza. Evaluation of semi-automatic metadata generation tools: A survey of the current state of the art. *Information Technology and Libraries*, 34:22–42, 09 2015.

[21] Pydantic. Pydantic. `https://docs.pydantic.dev/latest/`, April 2024.

[22] Associação Fraunhofer Portugal Research. Tsfel. `https://tsfel.readthedocs.io/en/latest/`, March 2024.

[23] Alexander Reshytko. Typedframe. `https://typedframe.readthedocs.io/en/latest/`, January 2023.

[24] Jenn Riley. *Understanding Metadata*. National Information Standards Organization (U.S.), 2017.

[25] Purvi Saraiya, Chris North, and K. Duca. An insight-based methodology for evaluating bioinformatics visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 11:443–456, 2005.

[26] JSON Schema. Json schema. `https://json-schema.org`, 2024.

[27] Union. Pandera. `https://pandera.readthedocs.io/en/stable/`, March 2024.

[28] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22, 09 2015.

[29] YData. Ydata profiling. `https://docs.profiling.ydata.ai/latest/`, March 2024.

[30] Shaked Zychlinski. Dython. `http://shakedzy.xyz/dython/`, February 2024.

[31] Slavko Žitnik and Erik Štrumbelj. Introduction to data science. `https://fri-datascience.github.io/course_ids/handbook/`, October 2022.