CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Edge Computing Targeted Data Profiling for Assessing the Quality of Sensor Data

Master's thesis in Computer science and engineering

Rasmus Jemth

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

Edge Computing Targeted Data Profiling for Assessing the Quality of Sensor Data

Rasmus Jemth

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

Edge Computing Targeted Data Profiling for Assessing the Quality of Sensor Data

Rasmus Jemth

Supervisor: Gregory Gay, Computer Science and Engineering (Software Engineering Division)
Advisor: Siddhant Gupta, Volvo Car Corporation
Examiner: Jan-Philipp Steghöfer, Computer Science and Engineering (Software Engineering Division)

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Edge Computing Targeted Data Profiling for Assessing the Quality of Sensor Data
Rasmus Jemth
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigated how the data quality of sensor data is determined and if it could be determined efficiently via a framework deployed on an edge computing device. The purpose of this was to enable data quality assessment to be done locally or close to the source of sensor data, i.e., utilize edge computing, in order to overcome the limitations of relying on cloud computing to assess large volumes of data. To determine how data quality should be defined, a literature study was conducted, which resulted in a selection of data quality dimensions deemed relevant for sensor data. The selection from the literature study contains the dimensions: Completeness, Accuracy, Timeliness, Consistency, Currency, Duplicates, and Precision. Based on this list, techniques based on data profiling was developed to assess the quality of sensor data, each technique assessed the quality regarding one specific data quality dimension. These techniques were implemented as part of an extendable framework. Each technique was evaluated by its time and space complexity, and the actual time it took for it to analyze datasets provided by Volvo Car Corporation, containing sensor data created by autonomous vehicles. Finally, approaches to improving the quality of sensor data based on the selected data quality dimensions were investigated.

Keywords: Data Profiling, Data Quality, Data Quality Assessment, Edge Computing, Sensor Data.

Acknowledgements

I want to thank my supervisor Dr. Gregory Gay from the Software Engineering division at Chalmers and the Univerisity of Gothenburg for guiding me throughout this thesis and for offering valuable feedback and help. I would also like to thank my advisor Siddhant Gupta at Volvo Cars for helping me with this thesis and for allowing me to do this thesis at Volvo Cars.

Rasmus Jemth, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1
Introduction

With the recent emergence of *IoT* (Internet of Things) products like voice assistants, smart sensors (e.g., thermometer, home security) and connected vehicles (e.g., cars, scooters), a need for quick computation (i.e., computation with low latency) that is not limited by restricted bandwidth have been created. This need is because many of these products generate a significant amount of critical data with their sensors. The amount of data generated depends highly on the product, for instance, connected autonomously driving (AD) cars are said to generate terabytes of data each hour of driving [2]. During recent years, a substantial amount of computational tasks are being done in centralized compute clusters (data centers), this is a part of the *Cloud Computing* paradigm [3]. When applying cloud computing, the data needs to be sent to a compute cluster before any computation can be performed. To send the data requires sufficient bandwidth to support the transmission, and it also entails extra latency due to the transmission time between the device and the compute cluster. The bandwidth requirement and latency of cloud computing limit many of the previously mentioned products that rely on fast responses or create large amounts of data.

A new computational paradigm has emerged to solve the problems previously mentioned about cloud computing, called *Edge Computing*. Edge computing means the computation is performed on the *network edge*, which is somewhere between the source of data and the cloud, for instance close to, or within, an IoT-device [4]. The latency and bandwidth problem that cloud computing has will be avoided by performing the computations closer to the source of data. These problems are avoided since there is no need to send the generated data anywhere. Hence no bandwidth is required and no latency is added to the process. The disadvantage of edge computing is that the resources of an edge device are limited [5], especially compared to cloud computing.

This thesis investigated how to apply *Data Profiling* for edge computing to analyze sensor data. Data profiling is a set of analysis techniques to collect statistics, determine metadata, and assess the quality of data [6] [7]. The primary purpose of data profiling in this thesis is to determine the quality of sensor data. By applying data profiling techniques on the edge, a summary can be done in real-time based on the collected data, which could include important and informative statistics and insights about the data. The benefit of this approach is that while all the generated data might not be possible to send to a server/cloud due to an overwhelming amount of generated data and limited bandwidth, a summary would be smaller and require minimal bandwidth to stream to a server.

The approach to investigating data profiling for edge computing in this thesis was

to develop a data profiling framework. Various data profiling-based techniques were tested and used to create this framework. To determine if a technique was appropriate for edge computing, and therefore should be used in the framework, an evaluation was performed on the technique to determine its time complexity (Big O and Big Omega), space complexity (Big O and Big Omega), and actual speed. If a technique could handle big datasets with limited computational resources, it was deemed appropriate for edge computing.
The thesis project was done together with *Volvo Car Corporation* (VCC). The techniques were evaluated on datasets provided by VCC that had been generated by test AD cars, which were equipped with various sensors.
This thesis resulted in a list of data quality dimensions deemed relevant and applicable for determining the quality of any kind of sensor data. The selection is presented in Chapter 6. This list contains the data quality dimensions: Completeness, Accuracy, Timeliness, Consistency, Currency, Duplicates, and Precision. Given these, at least one data quality assessment technique based on data profiling was developed for each of the selected data quality dimensions. These techniques were evaluated by their time and space complexity, and how long time they take to process datasets, see Chapter 8 for the techniques and their evaluations. Furthermore, a list of data enhancement techniques used to improve the quality of data was presented, see Chapter 9.

1.1 Statement of the Problem

Due to the amount of sensor data some connected devices (e.g., IoT-devices) generate, it is not always feasible to send it to a server for further processing or examination since the bandwidth might be limited. In cases where the amount of generated data is too much for the available bandwidth, either sampling methods could be used that send only a subset of the generated data. Alternatively, the device would need to stop collecting data and synchronize with the server in given intervals. In both these methods, we need to sacrifice something. In the first method, we sacrifice valuable data, thus losing precision, accuracy, and potentially important abnormalities in the data. In the second method, there will be a long delay between when the sensor data is collected to the time we receive the sensor data.
Even though it might not be possible to send the data to a cluster as it is being generated, it might be interesting and necessary for the owner or manufacturer of a device to get a real-time summary of the generated data, which will give statistics and insights of the data generated from their device. A summary of a dataset would be much smaller (in size) compared to the dataset itself, so it would be possible to send it to a server with limited bandwidth. While a summary would not give the full picture of a dataset, it would give critical insights that could be important in determining if the sensors are working correctly or highlighting potential abnormalities. This thesis is aimed to create a framework containing various data profiling-based techniques that can create such a summary that shows the quality of the data generated by the sensors.
Many established data profiling techniques are computationally difficult [8], therefore, difficult to deploy on edge devices, due to the limited computational resources

of edge devices [5]. Hence, the data profiling-based techniques the developed framework uses needs to be efficient to enable deployment on an edge computing device.

1.2 Purpose

The purpose of this study is to create an efficient data profiling framework suitable for edge computing, which can be used to get insights about the data and determine the quality of the data (based on a set of data quality dimensions). The requirement of the framework was that it should be *agnostic*, i.e., the framework should support any sensor data. Furthermore, the framework should be possible to execute on the edge. To determine how this framework should be developed and which techniques and algorithms should be developed, an investigation was done on data quality dimensions and how data profiling could be used to assess the quality of sensor data.

1.3 Scope

The scope of this thesis was narrowed to sensor data. Thus all the techniques considered will be applicable to sensor data. The kind of data profiling which this thesis focused on was data quality assessment. Therefore only data profiling techniques with the purpose of data quality assessment were considered in this thesis. The work during this thesis was limited only to handle pre-generated data. Thus, no kind of testing or evaluations were done with actual sensors, even though the use-case for this research is to enable data quality assessment of sensor data in real-time. The only actual pre-generated datasets used in this thesis were datasets supplied by VCC that had been generated by an AD car.

1.4 Research Questions

This thesis aims to investigate data profiling for quality assessment on edge computing and develop a data profiling framework for edge computing. To enable this, the purpose was broken down into these research questions:

- Which data quality dimensions are relevant to determine the quality of sensor data?
- Can data profiling efficiently be used in edge computing to assess the quality of sensor data, and in that case, how?
- Could sensor data of bad quality be efficiently enhanced?

These research questions were chosen because to create relevant and informative insights, relevant and informative data quality dimensions are needed. Additionally, to calculate the metrics based on the selected data quality dimensions, efficient data profiling-based techniques and algorithms are needed so the framework can be deployed to an edge device. Finally, to modify or remove incorrect and unnecessary

data and to improve the overall quality of data, efficient enhancement techniques are needed.

2
Extended Problem Statement

This thesis investigated which data quality dimensions should be used to determine the quality of sensor data, and which data profiling and enhancements techniques should be used to determine and improve the data quality in regard to the determined data quality dimensions. The techniques used in this thesis are required to be efficient because this research is focused on developing these kinds of methods and techniques for edge computing (e.g., computing on IoT-devices and similar connected devices), which have limited computational resources.
This chapter will explain these subjects and present some common problems with or within these subjects. These explanations will be brief and give general information about these topics. Some of these concepts are explained more thoroughly in Chapter 3. Furthermore, this chapter presents the problems this thesis tries to solve and present some use cases for the result of this thesis.

2.1 Edge Computing

Computing has gone through different paradigms during the last century, from the first considered modern computers *Z3* and *ENIAC* developed in the 1940s, to the introduction of the *PC* in the 1970s, and more recently the emergence of cloud computing [9]. Cloud computing is a deployment model where resources are deployed over the Internet, this provides services like *SaaS* (Software-as-a-service), *PaaS* (Platform-as-a-service) and *IaaS* (Infrastructure-as-a-Service), where SaaS delivers software over the Internet (e.g., email service), PaaS delivers a deployment platform (like *Amazon Web Services* [10]), and IaaS provides actual resources like storage and processing units [9]. When referring to "resources deployed over Internet", it means that the resources being used are located somewhere else (not locally), often in a centralized data center. By consolidating computational resources and offering access over the Internet, it makes the service accessible virtually anywhere, scalable, and it improves the utilization of the hardware [9] [11]. However, cloud computing adds latency to the process and is limited by the bandwidth [4]. Cloud computing is not effective when there are strict latency requirements since, in the cloud computing model, the data needs to be sent to a compute cluster, and then the compute cluster will eventually need to send back the result of the computation. The process of transmitting data over the Internet takes time, and thus, sending and receiving data from a compute cluster will add a transmission delay. Also, if the available bandwidth is limited or the quantity of data is huge, it will take additional time to transmit all the data to the cloud platform, since the limited bandwidth will

throttle the transmission.
A new paradigm of computing has recently emerged that does not have the same problems as cloud computing with latency and data volume/bandwidth. This new computing paradigm is called edge computing. The main idea of edge computing is that we bring the computation and data processing to the *network edge*, which is some resource between the cloud and the source of the data [12]. In practice, this could be a mobile phone which generates data in the form of, for example, video, audio, or GPS location, where a sensor (like the camera) can be seen as the source of data, and the processing unit of the phone could be used as a means for edge computing. The goal of edge computing is to bring the computation close to the source of data, which results in quicker computations and no bandwidth requirements.
There are some disagreements and differences between how edge computing is defined and how it is performed. In this thesis, edge computing refers to the edge computation definition given in [12], i.e., any computation done between the source of the data and the cloud. The main focus and use case for edge computing in this thesis is computations done on or close to sensors. There are some other variants of computational approaches that could be seen as a subset of edge computing, or as edge computing implementations [13]. For example fog computing [14], cloudlet computing [15], and mobile edge computing (MEC) [16]. Any further investigation of these will not be done. However, there are short explanations of them in Chapter 4.
An example that highlights the benefits of an edge computing device could be a surveillance camera that detects, for instance, intruders. Object/intruder detection is needed to detect an intruder. If cloud computing were used to perform the intruder detection, the video feed would need to be sent continuously to a compute cluster. This would delay the detection of an ongoing robbery, which could result in a successful theft. Instead, if edge computing would be used in this scenario, we would detect the intruders much faster if a robbery was taking place since the video data would be analyzed at or near the surveillance camera.
The reason for the recent emergence of edge computing might be due to the rise of IoT-devices (Internet of Things). Many IoT-devices perceive the world around them with the help of sensors. Examples of such devices are voice-assistants that perceive sound through microphones, smart thermostats that use a temperature sensor. It is predicted that by 2025, real-time data generated by connected devices (like IoT-devices) will generate almost 50 ZB (1 ZB = 1 trillion GB) [17]. Many of the IoT-devices benefit by using edge computing instead of cloud computing due to no latency or bandwidth requirement.
To utilize edge computing in connected devices, the devices need to be equipped or connected to an edge computing device, which is a processing unit handling computations. Edge computing devices are defined by their location, i.e., they are located close to the source of data, this could technically mean that an ordinary desktop computer with a regular processor can be used as an edge computing device by connecting it to a, e.g., IoT-device. However, a desktop is not particularly suitable for this purpose due to the impracticality of equipping a probably quite heavy, expensive, and high-energy consuming desktop computer to a small device like, for instance, a camera. There are computing devices that are marketed or especially

suitable as edge computing devices due to their size, efficiency, cost, and mobility. An example of this is the *Nvidia Jetson Nano* [18], which is a small computer equipped with 128 Cuda-cores, which is especially suitable for deep learning, e.g., image recognition. Another example is the *Intel Neural Compute Stick*, which looks like a USB memory stick, and is easily attached through a USB connection. It is aimed towards enabling low-cost devices like *Raspberry Pi* [19] to perform various AI applications (e.g., computer vision).
Due to the requirement of size and mobility, the computational capabilities of edge computing devices are rather limited, especially compared to cloud computing, where there are almost unlimited computational resources. Hence, the tasks that are possible to be done on the edge are limited. To some degree, this could be combated by optimizing the software within the edge computing device to make it run faster. Due to the limited capabilities of many edge devices, this thesis needed to focus on developing techniques that were efficient, i.e., techniques that are possible to be deployed on an edge device.

2.2 Data Profiling

Data profiling is a set of techniques used to collect statistics of a given dataset or data source [8] [20]. The purpose of these statistics could be, for instance, to assess data quality or perform metadata discovery. The data that is being analyzed is often assumed to be in the format of a table, i.e., each data point (or datum) has multiple values, where each value represents the current data point's value for one axis (or column). To visualize how a data point might look like, assume a people registry. There is one column for forename, one for surname, and one for age. A data point would include all these values for one person (i.e., a tuple). An example of data profiling is, for instance, to determine the number/ratio of empty or faulty formatted values in a particular column or to analyze the relationship between two columns (e.g., find if there are any correlations).
Data profiling can be done manually, for example, by executing SQL-queries or by just eyeballing spreadsheets or SQL tables [7]. These approaches are time inefficient and error-prone when performed on a large amount of data. Instead of doing data profiling manually, there are different software tools, both free and commercial, which can be used for data profiling. In these tools, a dataset is imported, and then the data profiling functions are selected. Some examples of free software are *Aggregate Profiler Tool*, which uses data from relational databases or a Hadoop system, and *DataCleaner*, which has, for instance, data profiling functions like completeness analysis, and unique key check [21]. There are various commercial data profiling tools. The profiling functions are often packaged in a data processing suite. Some examples of such software are *IBM InfoSphere Information Analyzer*, *Oracle Enterprise Data Quality*, and *SAP Information Steward* [21].
Data profiling is traditionally done on relational data [8], for instance, SQL-databases. However, due to the recent trend of connected devices (like IoT-devices), a substantial amount of data that is being created and processed today are not relational data from databases, but sensor data created by connected devices. This opens up the opportunity to research how data profiling should be used on such type of data.

2.3 Data Quality

The quality of data is, in some sense, arbitrary since there are no determined universal rules to decide how data quality is determined. Instead, the quality of data can be measured in different dimensions, i.e., it is multi-dimensional [22]. Which dimensions should be used depends entirely on the context since the various dimensions are not relevant or applicable in all scenarios.
The reason why assessing the quality of data is important is to ensure that the data is correct, which could be the basis of various decisions. If the data is incorrect, it could lead to bad decisions or decrease the quality of a service or a product.
An example of bad quality data is, for instance, consider an Internet shop with a database containing a table of customers. In the table, there is a column for the email addresses of the customers. The data would be considered bad if many of the email addresses are formatted incorrectly, e.g., missing "@" or missing the email domain. If we wanted to send out an email to all our customers to advertise some products, many of our customers would not receive the email, since their address is incorrect. This would probably result in fewer orders. Another example where the quality of sensor data is important is a fridge with a temperature sensor. The fridge attempts to cool itself if the temperature is too high. If the data from the sensor is inaccurate or imprecise, it could lead to the food spoiling or freezing (depending on how the data is incorrect). The data, in this case, could be incorrect due to faulty or badly calibrated sensors. If this would be noticed early, we could repair or change the sensors to avoid any damage to, in this case, the food.
To determine data quality, one would have to establish a set of dimensions that the data will be evaluated on. Afterward, the quality could be evaluated by humans (e.g., collectors and consumers of data) [22], this is suitable when a computer cannot determine the data quality dimensions (e.g., dimensions that indicate how readable or understandable the data is). However, this is not particularly time or cost-efficient. Thus it is preferable to create an automated software framework to determine data quality.
Since, in most cases, multiple data quality dimensions are used to describe the quality of data, it might be challenging to compare the overall quality between two datasets. One of the datasets can have a higher metric on one of the dimensions, and the other dataset might have a higher metric on another dimension. To be able to differentiate easily between datasets and to easier understand the overall quality of a dataset, all metrics could be aggregated into one combined metric or score. Dai et al. [21] present two metrics for this, one of which is the linear *Data Quality Metric* (DQM), where you assign each dimension a weight w_i between 0 and 100 (the sum of all weights must be 100), where i indicates the dimension. Then the metric q_i of each dimension is multiplied with its assigned weight. Lastly, all the products are summed. This formula assumes that the used data quality dimensions are expressed as a value between 0 and 1, i.e., a normalized value. The formula for DQM is:

$$DQM(t) = \sum_{i=1}^{n} q_i^t \cdot w_i \quad where\ 0 \leq q_i^t \leq 1\ \ \forall i \in [0, n]\ \ and\ \sum_{i=1}^{n} w_i = 100$$

Where t indicates the current time.
The other combined data quality metric presented was *Exponentially Weighted Data Quality Score* (EWDQS) [21]. EWDQS is related to time, the EWDQS at time t is dependant on DQM at time t and the EDWQS at time $t-1$, and an arbitrary set λ-value between 0 and 1. The formula for EWDQS is:

$$EWDQS(1) = DQM(1)$$

$$EWDQS(t) = \lambda \cdot DQM(t) + (1-\lambda) \cdot EWDQS(t-1) \quad 0 \leq \lambda \leq 1$$

Due to the lack of a standard on evaluating the quality of data, it might be difficult to determine how data quality should be objectively evaluated. By establishing a set of data quality dimensions or a framework for determining data quality aimed for a specific kind of data like sensor data, it would be easier and more straight forward for engineers and researchers working with, in this instance, sensor data, to determine the quality of their data.

2.4 Data Enhancement

Data enhancement can be performed on a dataset to improve the quality of the data. Like data quality dimensions, different enhancement techniques are suitable in different contexts. It could be used to remove bad data or to try to correct faulty data points. Data enhancement techniques can be applied to improve the quality of a specific quality dimension or multiple quality dimensions.
Assume the example of the table of customers given earlier in Section 2.3. A table contains a column with the email addresses of the customers. If some email addresses are formatted incorrectly, we might be able to correct the faulty email addresses. For instance, we know that all email addresses must contain one "@". Some of the addresses in the table might contain two of them or none at all, due to human error. In this example, it would be possible to implement a function based on this heuristic that would detect these kinds of formatting errors and correct them.
Data enhancement can be beneficial in an IoT-device or a similar environment since it could remove or correct erroneous or redundant data. This is beneficial since sending erroneous or redundant data to the cloud will take up bandwidth and storage without providing any valuable information.

2.5 Environment of Data Collection

As explained previously in Section 2.2, data profiling is usually done on an already existing dataset or database. In this report, data profiling is instead done on sensor data, which generally is generated continuously as sensors collect data. However, this thesis's research is limited to performing evaluations on pre-generated datasets and not data that are generated in real-time. Nonetheless, the aspect of the data being generated continuously was still emulated by letting the techniques process one data point at the time, in an incremental fashion.
The set of data points received thus far (in time) is referred to as a "continuous

dataset" in this report. This term is introduced to avoid confusion between datasets, and the kind of data (environment) a sensor data framework would analyze. However, continuous datasets are a subset of datasets (i.e., a continuous dataset is a dataset, but a dataset does not necessarily need to be a continuous dataset).

2.6 Aggregated Problem Statement

In this thesis, we will develop a data profiling framework for determining and enhancing the quality of sensor data whose purpose is to be deployed to some edge computing device. The various problems of creating such a framework are:

- Creating a framework that can be deployed on an edge device.
- Determining which kinds of data quality dimensions should be used to evaluate the quality of sensor data, and how those dimensions should be presented.
- Finding which data profiling techniques that are useful to calculate the quality of data, and are efficient enough to be suitable for edge computing.
- Finding data enhancement techniques that theoretically can enhance sensor data with respect to the set of chosen data quality dimensions.

2.7 General Use Case

With the use of IoT-devices and other connected devices, it is sometimes not possible to utilize cloud computing in real-time due to time and bandwidth constraints. An example of this could be a surveillance camera recording video since video files are generally quite big. Another example of this is AD cars, which generate multiple terabytes of data for each hour of driving [2]. In these scenarios, it can be important to know that the sensors are working correctly. By deploying a framework to these kinds of devices to determine the quality of data, it is possible to get insights about the sensor data in real-time, since sending information about data quality does not require high bandwidth. While the sensor data itself will not be accessible online in real-time, this approach will provide insights into the quality of the generated data. The assessment of data quality ensures that whenever there is any problem with a sensor, it can be detected quickly over the Internet in real-time by looking at the quality of the sensor data.
Furthermore, in some cases, cloud computing is possible but is unnecessary and a convoluted approach for a task. An example of this is, for instance, a smart fridge manufacturer. The manufacturer may not have any reason to know the temperature of their customers' fridges in real-time, and thus designing the fridges to send data to a cluster would be unnecessary. Additionally, dedicating a cloud for handling all the temperature data for possibly millions of customers will be expensive. While the sensor data itself may be unnecessary for the smart fridge manufacturer, detecting bad sensor data, and thus detecting broken or damaged sensors in the fridge, could be important to improve the quality of the fridges. By deploying efficient techniques to such fridges, it is possible to either receive regular (e.g., daily) updates from the fridges, or get a warning whenever a fridge seems to be malfunctioning (due to bad sensor data). In either case, the amount of data that is sent and needs to be handled

is lower. It is thus more efficient and easier to handle this than to have possibly millions of data sources (in this case, fridges) that are constantly sending data to the fridge manufacturer (e.g., their cloud or server).

2.8 The Volvo Car Corporation Case

This thesis was done together with Volvo Car Corporation (VCC). The data profiling case for VCC is to try to use data profiling on sensor data generated by their AD test cars.
Companies that develop AD cars collect sensor data by driving the car in a limited area. The sensors used are, for instance, radar, LiDAR, and cameras. AD cars require many sensors to perceive the world around them, otherwise, they would not be able to operate themselves safely. The data generated from the test AD cars can be used to verify sensor quality, train machine learning models, or other functions related to AD. These test cars are equipped with multiple sets of sensors to improve reliability. After the collection is done, the data from those sensors are usually sent to a centralized data cluster for storing, validation, and processing.
Since data is sent only after the collection is done, no information about the collected data is known during the time it is being collected. By equipping the AD cars with some edge computing device, the data can be processed and profiled during data collection, which could be used to create a summary that is feasible to send in real-time to a server, even with limited bandwidth. A summary would give live insights about the data generated/collected, which could be beneficial since it would enable the possibility of spotting faulty sensors or other abnormalities.
Data profiling can not be done in real-time via cloud computing because AD cars generate immense amounts of data, as much as multiple terabytes for of data each hour of driving [2]. Due to bandwidth limitations, it is not possible to send the data during the collection to a centralized data cluster. However, by performing the computations on the edge (i.e., at the car), you can send a summary created in real-time by a data profiling framework that contains the most critical insights. This is possible since a summary will only be a small fraction of the original continuous dataset's size, and thus the bandwidth requirements are much lower.

3
Background

This chapter explains the background of the concepts that are brought up and used in this thesis. It explains the theory of the most vital parts of this thesis and explains different data quality dimensions, data profiling, and data enhancement techniques.

3.1 Data Quality

This section describes the categories and types of data quality, and lists dimensions that can be used to determine the quality of data. The data quality dimensions presented in this section are the basis of the final dimension selection (see Section 6.3), which answers the first research question: "Which data quality dimensions are relevant to determine the quality of sensor data?".

3.1.1 Categories and Types

Data quality dimensions can be categorized by their context, underlying meaning, and by how quality is determined. The main four categories of data quality dimensions are *Intrinsic*, *Contextual*, *Representational*, and *Accessibility* [23].
How data quality dimensions are defined is either domain-agnostic or domain-specific. Domain agnostic means that the specific dimension can be applied and used on any dataset, i.e., the dimension is not dependant on the external context of the data. Domain-agnostic dimensions could be based on things like, for instance, spelling errors, formatting inconsistencies, and the number of empty/null values. Domain-specific dimensions are determined by a set of rules created/derived by a specific company, governmental rules, or other rules/policies specific to some context. An example of this could, for instance, be data displaying the temperature, if the data is displayed in Kelvin, a data point with a negative temperature value is invalid, meanwhile, if Celsius is used, a negative value is possible and acceptable.
The metric of a dimension can be developed and calculated with one of three functional forms, *Simple Ratio*, *Min (or Max)*, or *Weighted Average* [22]. The simple ratio is used to calculate the frequency of a specific occurrence. The occurrences calculated with a simple ratio metric is usually measuring the occurrence of something good (i.e., the higher value/ratio, the better), like the ratio of valid values to the total number of values [22]. In the case of calculating the occurrences of valid values, it is calculated with the simple ratio like this:

$$\frac{number\ of\ valid\ values}{number\ of\ invalid\ values + number\ of\ valid\ values}$$

A min or max form is to take the min or max value among some data quality metrics with normalized values (between 0 and 1) [22]. Using min is a conservative metric since it usually is used to determine the worst-case among some parameters, meanwhile, max is a liberal (optimistic) metric.
A weighted average form is similar to min/max form, but instead of taking the min/max value of a set of variables or metrics, it takes the average of the variables/metrics [22]. The quality metrics used can either be equally weighted or differently weighted if it is known that some of the dimensions are more important than others.

3.1.1.1 Intrinsic Data Quality

An intrinsic data quality dimension describes and is given by the data in a dataset [24]. An example of a typical intrinsic data quality dimension is *Accuracy*. Accuracy describes how well the dataset reflects the real world, it is calculated by the biggest error or biggest uncertainty in the dataset (this could be the biggest difference between two values from different sensors with the same time/timestamp).

3.1.1.2 Contextual Data Quality

A contextual data quality dimension is determined from information about the dataset, rather than using specific data in the dataset, as in an intrinsic dimension. Contextual dimensions indicate how suitable the data is for a task or the current context [25]. One commonly used contextual dimension is *Completeness*. The completeness dimension is given on the simple ratio functional form and calculates the frequency of missing data. This could be that a specific number of data points are expected and the number of available data points is less than the expected, or that some values in the dataset are empty, set to a default value or null. The actual values within the dataset do not affect the quality, only the absence and presence of data points/values affect the metric of this dimension.

3.1.1.3 Representational Data Quality

Representational data quality dimensions describe how understandable the data and data formats are. Examples of representational dimensions are *Interpretability* and *Ease of understanding* [23]. This category of data quality will not be further investigated or considered since it is not relevant to the thesis. There is no major advantage of having sensor data that is easy to understand and interpret, since humans rarely need to inspect raw sensor data.

3.1.1.4 Accessibility Data Quality

Accessibility data quality dimensions show how accessible the data is to, e.g., consumers of the data and how secure it is (i.e., how difficult it is for intruders to access the data) [24]. This category will not be considered since in the context of this thesis, it is assumed that the data is accessible and ready for use. Additionally, any security aspect is not in the scope of this thesis.

3.1.2 Data Quality Dimensions

This section describes different intrinsic and contextual data quality dimensions, some of which has already been briefly explained in the previous section.

3.1.2.1 Completeness

Completeness is a contextual and domain-agnostic dimension that indicates how much data is missing, this could be the number of empty or null values in a column of a dataset [7], or in the context of sensors, the number of missing values [26], which could be due to faulty sensors. Since it is not possible to quantify the number of missing records (you can not count something that you do not receive), an approximation can be made instead. For instance, if a sensor is expected to send 10 values each second on average, and in a specific dataset or time frame, only 2 values are received each second, approximately 8 values/second is probably lost.
Completeness is calculated by computing the ratio of the number of existing values to the total number of values:

$$\frac{number\ of\ existing\ values}{total\ number\ of\ values} = \frac{number\ of\ existing\ values}{number\ of\ existing\ values + number\ of\ missing/empty\ values}$$

3.1.2.2 Data Volume

Data volume is a contextual, domain-agnostic dimension that determines how much raw data is available for a specific task [25] [26]. Data volume can be calculated using the completeness metric. For instance, if for a particular task, only data that is 1 minute old is useful, then the volume of data is calculated by $f \cdot 60 \cdot c$, where f is the rate which data is generated and c is the completeness metric.

3.1.2.3 Appropriate Amount of Data

The appropriate amount of data dimension is a contextual, domain-specific dimension that indicates if the volume of data is enough for a specified task [22]. This dimension is derived by the "Data Volume" dimension and by how much data a specific task requires. For example, if we need at least 10.000 data points for a specific task, we would have a threshold of 10.000, and if the number of data points we have/collected is above this threshold, the data is appropriate, otherwise, it is not, thus, this could be seen as a binary dimension. Since the threshold varies from different tasks, it is domain-specific.

3.1.2.4 Timeliness

Timeliness is defined differently in the literature, but it generally is based on the time/age of the data. Some of the definitions are:

- Difference between the current time in the system and the timestamp of a data point [26].
- Age of a data point [26].
- The "currency" of data, i.e., how recently the data got updated [27].

In this thesis, timeliness will refer to the first definition in the list above. This was decided since the two other descriptions fit the next data quality dimension described in this section; "Currency".
Timeliness will show the delay between when the data is initially recorded on the sensors to when the data is received by (in the case of this thesis) the data profiling framework.

3.1.2.5 Currency

Currency is a domain-agnostic, contextual data quality dimension. It describes how up-to-date the data is [28]. As with timeliness, currency is also defined differently in the literature. Some define it as a metric given to a data point based on when it was received by a system/database [29]. Others define it as how recently data was updated [24].
In this thesis, currency will refer to how recently the dataset was updated. This will indicate if sensors have stopped sending data, and are thus faulty. When the currency of a continuous dataset starts to drop, it might be due to faults in the sensors.

3.1.2.6 Accuracy

Accuracy is an intrinsic, domain-agnostic dimension, which is given by the biggest measurement error [25], which in the context of sensor data is the biggest difference at a specific time between different sensors that measure the same thing (this assumes two or more sensors measuring the same thing). It is calculated this way on sensor data because we can never truly know the true values (ground truth) when measuring in the real world, the only information we have is from the sensors, we are thus limited to comparing the available sensors to each other.
Accuracy is calculated like this: Assume we have a set of sensors that measures the temperature. The time is indicated by t, and there are in total n time steps. A temperature value from sensor i at time step t is written like this. v_t^i. The accuracy a is calculated by taking the biggest difference between two sensors at the same time step, i.e.:

$$a \;=\; max_{i,j,t}\; v_t^i - v_t^t \; such\; that\; i \neq j$$

An alternative to calculating accuracy this way is to instead aggregate data values that are close to each other (in time) and creating a larger set of values to calculate the accuracy. An example of this, consider a single sensor that generates 10 values each second. The values can then be aggregated to groups of, for example, 10 values, and then the accuracy is calculated by taking the biggest difference between those 10 values. The advantage of this is that accuracy can be calculated in cases where there is only one sensor available.
It is important to note that the accuracy dimension can refer to a slightly different meaning. In [21], accuracy is defined as the percent of data that is correct. As previously explained, in the context of sensors, it is impossible to really know the true values, and we can thus not determine if a data point or value from a sensor is correct.

3.1.2.7 Confidence

Confidence is an intrinsic, domain-agnostic dimension that shows the maximal statistical error of, in this case, sensor data [26]. An example of this is that we measure temperature with a set of sensors, then with the statistical error $p = 99\%$ we might have an expected error of ϵ =3 degrees, this implies that if the average value received from the sensors is 20, then with a probability of 99%, the actual value is between 17 and 23. The smaller the error ϵ, the better the quality of the data since that means we are more confident about the actual value. To calculate confidence, two or more sensors are required, since two or more values are required to calculate a confidence interval. In sensor data, confidence describes the measurement error from random noise that might affect a sensor [26]. The confidence interval is calculated by the Z-value (given by the desired confidence interval), the mean, and the standard deviation of the measurement errors given from the sensors.

$$interval = [\overline{x} - Z \cdot \frac{\sigma}{\sqrt{n}}, \overline{x} + Z \cdot \frac{\sigma}{\sqrt{n}}]$$

$\overline{x}$ is the mean of the values, σ is the standard deviation of the values, n is the number of values.

The margin of error decreases as the number of values (population size) increases. To create a lower (and thus better) margin of error, one could aggregate values as described in Section 3.1.2.6. By doing this, confidence can also be calculated in scenarios when only one sensor is available.

3.1.2.8 Believeability

Believability is a contextual dimension which is calculated on the min functional form as explained in Section 3.1.1, i.e., is computed by finding the minimum value among a set of values. It shows how credible the data is. It is calculated by taking the metrics of the other data quality dimensions and normalizing them (if the metrics are not already normalized) and selecting the lowest metric [22].

3.1.2.9 Validity (Free-of-Error)

Validity indicates how much of the data is correct/valid concerning its format or value [21]. Validity is given by the simple ratio form and shows the ratio of the number of correct values (or correctly formatted values) to the total number of values. How to determine if a data point is incorrect/invalid or not, depends on the task and the established rules for that task. For example, if measuring the velocity of a car, then every value that is not between 0 and 35 m/s can be seen as incorrect. The validity dimension is dependant on the task, and thus, the validity dimension is domain-specific.
In some papers, this dimension is referred to as *Free-of-Error* [22], there are some slight variations to what it means, but in most papers, the dimensions Validity and Free-of-Error are almost identical, thus it would be redundant to use both.

3.1.2.10 Objectivity

Objectivity indicates to what extent the data is unbiased [22]. This dimension is domain-specific since what constitutes a bias depends on the situation (there is no "inherent" bias all data must have). In the literature, this dimension is not further explained. Thus, it is open for interpretation of how one would implement this dimension.

3.1.2.11 Relevancy

Relevancy indicates how useful or applicable the data is for a given task [22]. Since it is dependant on the task, it is domain-specific. As with the dimension "Objectivity", relevancy is not clearly defined, and there is thus no formal guideline on how this should be implemented.
An example of the use of the relevancy dimension is, for instance, if we wanted to train a machine learning model to classify images of cats and dogs. A dataset with horse images would have a low relevancy score in this context since, for the task of training such a model, it requires specifically cat and dog images.

3.1.2.12 Value-Added

Value-added is a data quality dimension that describes how beneficial and advantageous it is to use a particular dataset for a specific task [22]. It is domain-specific since depending on the situation, the value which the data adds might differ.
This dimension is similar to the previously explained "Relevancy" metric, but they are not equivalent. If some data is relevant for a task, it does not mean that it will be advantageous to use that data for the given task. For example, assume the example given in Section 3.1.2.11. We want to train a machine learning model to classify images of cats and dogs. Assume we have a dataset containing fives images with cats, this data is relevant. However, due to the lack of dog images, we would not be able to train the model properly, and thus the value that the dataset actually provides is low.

3.1.2.13 Consistency

Consistency measure how consistent and correct the data is. Consistency is measured differently depending on the context. It is mostly used on databases (i.e., relational data), e.g., measure how equal the data is on two (or more) separate storages, which should be synchronized and equal [30].
In [28], consistency is divided into three categories: *format consistency*, *logical consistency*, and *strong consistency*. Format consistency measures the degree to which the data follows formatting rules (e.g., the values must be integers). Logical consistency measures to what degree the dataset follows a set of logical rules. Logical rules could be that a value must be bigger than, for instance, 0, or logical dependency rules like a data point's value in column A plus its value in columns B must equal its value of column C. Strong consistency measures how equal two datasets are, i.e., the measure given in the example above with the databases.
Both format and logical consistency are domain-specific since they are dependant on

a set of either formats or rules. Strong consistency is domain-agnostic since it does not require any set of pre-defined rules depending on the context. However, this can only be used for the specific use case of having two or more different storages that should be redundant (i.e., have the same data). Since this thesis will use data quality dimensions on sensor data, strong consistency will not be applicable.
However, in [31], consistency is determined by how consistent the other data quality dimensions are, this is a bit similar to strong consistency explained above, but instead of measuring how equal the data is between different storages of data, it measures how equal the data quality is between different data points within the same dataset. For example, assume we have a dataset where each tuple has ten values, thus the table within the dataset has ten columns. Seven of these columns have no missing values, thus a completeness of 100% for each of the seven columns, the remaining three columns have a completeness metric of $95\%, 90\%, 80\%$. Then the consistency of the completeness in the dataset is 70%, since 70% of the columns "agree" on a completeness value of 100%, and are thus consistent with each other. However, a high consistency does not necessarily mean that the data is good, if a dataset, in general, is bad, then the data will "agree" on a bad quality metric, and thus achieving a high consistency.

3.1.2.14 Duplicates (Uniqueness)

The duplicate dimension indicates if there are any duplicates in the data [25] and if there are, to what extent are they present. The duplicate dimension is a contextual, domain-agnostic dimension that uses the simple ratio form. This dimension is calculated by dividing the number of unique data points (i.e., non-duplicate data points) with the total number of data points. Thus, a high duplicate quality value (i.e., close to 1) indicates a low frequency of duplicates. Likewise, a low duplicate quality value (i.e., close to 0) indicates a high frequency of duplicates.

3.1.2.15 Precision

Precision is a domain-agnostic, intrinsic data quality dimension that measures the standard deviations among values in a dataset [21]. In the case of sensor data, if the standard deviation is low, it indicates that the sensors have high precision, and vice versa.
To calculate standard deviation, two or more values are required, and thus precision can only be calculated in scenarios where there are two or more sensors. However, as with the accuracy and confidence dimensions, data from a single sensor can be aggregated to enable the calculation of the standard deviation, and thus calculate the precision dimension.

3.1.2.16 Reputation

Reputation gives a measure of how well regarded the data is, given its source or content [22]. An example of reputation that is determined by the source of the data is: if we had one accurate and expensive sensor and one cheap and inaccurate sensor, then data from the cheap sensor will have a lower reputation since we can assume

that the data coming from that sensor is worse, likewise the expensive sensor would have a higher reputation.
The determination of the reputation of data is dependant on the data source and the data sources' reputation. The source of data trivially differs in different situations and contexts. The reputation dimension is, therefore, domain-specific.

3.2 Data Profiling

There is no clear consensus on the definition of "Data Profiling" [21]. In [32], it is defined as creating informative summaries from data (or a database). According to [7], the definition of data profiling is a set of techniques that can determine metadata from a dataset. The paper [6] defines it as a method to ensure data quality.
These definitions both conflict and overlap. By assuming the definition given by [7] (i.e., determine metadata), we will encompass the other definitions by having a liberal view of metadata. The definition of metadata is data that gives information about data [33], basically data about data. Both a summary and data quality can be seen as metadata since they are both data given by other data. However, in this thesis, we define data profiling as the process of summarizing the data quality of a dataset.
The focus of data profiling in this thesis is determining data quality. This section will go through the three main classes of data profiling, and explain specific data profiling categories and approaches.

3.2.1 Data Profiling Classes

Data profiling techniques can be divided into three main classes, *Single Column*, *Multiple Columns*, and *Dependencies*.

3.2.1.1 Single Column

Single column data profiling techniques analyze only one column of data in a table [7]. This could range from basic counts to more advanced methods such as creating histograms.

3.2.1.1.1 Cardinalities

Cardinalities are specific values or counts of various specified occurrences. Some examples of cardinalities in a column are the total number of rows, the length of data in that column (e.g., max/min/average length of strings), maximal and minimal value, number of non-existing values, the number of distinct values. The value length can be used to perform outlier detection and format the data to make it more consistent. The number of non-existing values can be used to calculate completeness. The number of distinct values can be used to determine the duplicate quality dimension [7].

3.2.1.1.2 Patterns & Data Types

The category Patterns & Data types contain techniques to determine the most common data type and pattern the values in a specific column have, which can be used to determine poorly formatted or invalid data [7]. Patterns in this context refer to the format and structure of the data in a column. An example of this is if there is a column that contains phone numbers, then most of the phone numbers in that column should have the same (correct) format. By identifying the pattern of the most common format for (in this case) phone numbers, one can find outliers that do not conform to the common formats. These outliers are probably wrong, and by detecting these, they can be corrected or removed.

3.2.1.1.3 Value Distributions

Value distribution describes the cardinalities of various discretized groups that the values in a column are assigned to or belong to. An example is a column that indicates gender, where the possible values on this column are "male" and "female". The value distribution will reveal the number of occurrences of the value female and the number of occurrences of the value male. When handling data that is not discretized like gender, i.e., continuous data like sensor data, the values should not be put into groups, where each group represents a specific value. As an example to display why this should not be done, let us assume we have a column for velocity, where one data point has the velocity value $11.033 m/s$. It is unlikely that many other data points will have the exact same velocity. As a result, most of the groups will only have one member. Instead of this, we can discretize ranges of values, like one group for velocities between $0-5 m/s$, one group for velocities between $5-10 m/s$, and one group for $10-15 m/s$, this way, it is more clear and easy to understand the data and to get a grasp of the distribution of velocities in that column.
Value distributions can be used to create histograms, which are graphs showing the frequency of occurrence of the various groups [7]. In a histogram, each group will have a bar indicating the number of members of that group.
Furthermore, *constancy* is also part of this category, which is the ratio of the most frequent value to the total number of values, or the ratio of the number of values in the most common group to the total number of data points.

3.2.1.1.4 Domain Classification

Domain classification is used to classify the meaning (or title) of a column, i.e., if a column contains phone numbers, that column should be classified as "phone number" [7]. This is only useful when the column has no descriptive name.

3.2.1.2 Multiple Columns

Multi-Column profiling is similar to single column profiling, but it uses two or more columns.

3.2.1.2.1 Correlations & Association Rules

Correlation techniques reveal if different columns are related to each other [7], i.e., if there is some relationship between the values from two or more different columns. An example of this could be a table of employees. There is a column for age and a column for salary, one might suspect there is a correlation between how old an employee is, and how much an employee earns. Association rules are a bit similar to correlation techniques, but instead of finding numeric correlations or relations, they look for associations between specific values between the columns. Association rules can, for instance, be used to discover products frequently purchased together [7]. For instance, assume a table of purchases in a food market, where each data point contains the items one customer bought during one visit to the store. An association rule that could possibly be found in this scenario is, when hamburgers are purchased, then usually french fries are also purchased. This association rule would be written like this:
$\{hamburger\} \rightarrow \{french\ \ fries\}$.

3.2.1.2.2 Clustering & Outliers

By clustering data points into groups that are similar based on a subset of their values, data points that do not fit into any of these groups can be detected as outliers and then be removed [7]. Clustering is an unsupervised machine learning technique used to find patterns in data [34]. Some common clustering algorithms are k-NN (k-nearest neighbors) and K-means clustering. These are defined in Section 9.1.3.

3.2.1.2.3 Summaries & sketches

Summaries and sketches are used to make easily accessible insights to the user about the underlying data. These are created by sampling or downsizing, so the result is smaller in size than the original dataset [7].
Some of the basic techniques that could be used to create summaries are to calculate the mean and standard deviation of one or multiple columns. Another approach is to use clustering [35]. Summaries and sketches encompass earlier explained data profiling techniques. One could argue that summaries and sketches are a set of other data profiling techniques, where the techniques somehow reduce the volume of data to create a compressed version of the data that still provides meaningful information and insights. A valuable summary would have as much “compaction” (small data volume) while minimizing the information loss [34].

3.2.1.3 Dependencies

Dependency data profiling techniques are similar to multiple column profiling in that both use two or more columns, however, dependency techniques are used to find and describe dependencies between different columns. Some examples of dependency techniques are finding functional dependencies and finding unique row combinations [7]. To find dependencies can be computationally difficult due to that

the number of potential dependencies is exponential, which means that all dependency algorithms have a worst-case complexity of exponential time. Furthermore, many of the use-cases of dependency discovery focus on finding better and more efficient ways to create and optimize a schema of a database, e.g. finding keys and removing redundant columns.
Due to dependency discovery being computationally difficult, it is not suitable for the computationally limited edge computing devices. Furthermore, many use cases for dependency discovery is for databases (database management) and not sensor data. Due to these two facts, this thesis did not investigate any dependency techniques or algorithms.

3.3 Data Enhancement

When data has predictable errors, one can mitigate these errors by applying data enhancement techniques that correct these errors. Data enhancement could, for instance, be: adding missing data, correcting/editing erroneous data, or simply the removal of incorrect or unnecessary/redundant data.
This section lists and explains different data enhancement techniques.

3.3.1 Outlier Detection

The purpose of outlier detection is to find data points that are significantly different from other data points. Outliers may indicate incorrect data, rare events, or other anomalies [25]. These outliers can be removed from the data to make the data more accurate and homogeneous, or to highlight rare and important events.
To find outliers, one could use clustering algorithms. A data point that does not conform to any of the cluster groups could be seen as an outlier.

3.3.2 Interpolation

Interpolation is that the values of a missing data point are inferred. This can be used to combat the problem of having a dataset with missing data points (low completeness).
Given a missing data point d_i at timestep i, we can utilize the data points close to d_i in time, e.g. d_{i-1} and d_{i+1}, to infer the value of d_i. The method of inference varies, some examples of interpolation methods are linear interpolation and polynomial interpolation [25].

3.3.3 Deduplication

Deduplication methods find duplicates of data points and remove the redundant data points so that there are no data duplicates left in the dataset. This is beneficial since duplicates offer no further information, due to the data being redundant. Another benefit of deduplication is that it will make the dataset smaller, without losing any information.

3.3.4 Data Cleaning

Data cleaning consists of a set of steps to enhance data. The steps are:

1. Defining the error types, and what should be classified as an error.
2. Identify errors based on the definitions created in step 1.
3. Correcting or possibly removing the errors identified in step 2 [36].

Data cleaning can be seen as a superset of techniques to both outlier detection and interpolation since step 2 of data cleaning is error detection, which could be outlier detection. Furthermore, the correction part of step 3 can be an interpolation method.

When designing a data cleaning process, one could (and maybe has to) utilize different enhancement techniques, like outlier detection and interpolation.

4
Related Work

The subjects (data quality, data profiling, etc.) that are being addressed and researched in this thesis have previously been researched separately to a varying degree. However, no research discusses how all these subjects can be used together. This thesis aims to fill the gap of data profiling for quality assessment performed by edge computing.

4.1 Edge Computing and IoT-devices

Due to edge computing being a recent technology (or paradigm), the research and development within this subject are limited. There are some people that point out some use cases and applications for edge computing, like Satyanarayanan [37], who highlights the use case of applying edge computing for analytics and the privacy aspect of processing sensor data at the edge. However, with edge computing defined as any kind of computation between the source of data and the cloud, it encompasses a few different computational methods. Dolui and Datta [13] describe and primarily compares different implementations of edge computing. The different implementations that are brought up are fog computing, cloudlet computing, and mobile edge computing (MEC). Fog computing is defined as computations performed in devices like gateways and routers that are placed somewhere between the end devices and the cloud. These kinds of devices are called fog computing nodes. Mobile edge computing is defined as nodes deployed in RANs (radio area networks) to provide mobile-devices with cloud-like computing capabilities. Cloudlet computing is defined as dedicated devices that emulate the capabilities of data centers, but at a smaller scale and placed closer in proximity to the end-users.
While the research on specifically edge computing as a computing paradigm has not been that extensive, there has been more extensive research within some specific edge computing implementations, e.g., fog computing and mobile edge computing.
Within the area of mobile edge computing, there exists some research on how it could be used to improve IoT-devices. Sun and Ansari [38] present an architecture of how IoT-devices utilize mobile edge computing in base stations connected to fog computing nodes. Furthermore, Abbas et al. [16] present a survey on MEC which provide an overview of the subject, and give examples of practical use cases for MEC. Some of the use cases given are the use of MEC in mobile big data analytics, connected vehicles, and wireless sensor networks where the main benefits of using MEC in these use cases are primarily low latency and enabling the processing and analysis of data with high bandwidth.

Most of the research within edge computing and its sub-fields are focused on the architecture and the communication between the computation device/node, the source of data, and the cloud. However, the focus of this thesis is mainly on the implementation of an application, i.e., data quality assessment. There is some research similar to this, for instance, Cheng et al. [39] developed the system *Geelytics* that utilizes edge computing for analytics from IoT-devices, which they refer to as *edge analytics*. Edge analytics is also researched by Xu et al. [40], where they present a service for edge analytics in real-time, called *Edge Analytics-as-a-Service* (EAaaS). The purpose of the developed EAaaS was to provide a more suitable alternative for IoT applications, instead of using some kind of cloud-based service. While [39] and [40] are similar to this thesis since both focus on utilizing edge computing to create some insight about sensor data/IoT-devices, [39] and [40] does not perform any data quality assessment or does not utilize any data profiling technique.
Another approach of utilizing edge computing for IoT-devices has been made by Casado-Vara et al. [41]. Their goal was to improve the quality of sensor data with an edge computing architecture based on blockchain technology. This paper is similar to this thesis, however, the quality of data is not explicitly determined, and no data quality dimensions are used. Instead, the algorithm proposed tries to, given a set of multiple temperature sensors, predict the actual "true" temperature, this algorithm is based on game theory and assumes numerous sensors.
Bonomi et al. [14] argue for the use of fog computing in IoT, based on the characteristics of fog computing. Some use cases proposed in [14] are the use of fog computing in connected vehicles and wireless sensor networks. However, in contrast to this thesis, [14] does not propose any specific techniques of how edge computing should be done, it is limited to propose use cases. He et al. [42] propose a new computing model based on fog computing for IoT analytics. They ran their experiments on Raspberry Pi computers, which partly inspired the selection of hardware in this thesis. However, it did not include any data quality assessment or data profiling.
Hromic et al. [43] present an approach to provide analytics on IoT-devices and services. Their approach is based on using a platform called OpenIoT and using clustering algorithms. While [43] presents an approach for IoT-device analytics, as the other papers included in this section, it included neither data quality assessment or data profiling.
Hence, there was almost no research available on either data profiling for edge computing or data quality assessment for edge computing, except for [41].

4.2 Data Quality

Pipino et al. [22] describe how data quality is presented with the help of different functional forms. They list a set of data quality dimensions and explain the importance of maintaining a high quality of the data within an organization. That paper gives an overview of data quality but does not specify a particular domain or use case. [22] provided this thesis with a list of data quality dimensions and how various dimensions are expressed, however, the focus of that paper was not sensor data. Meanwhile, Karkouch et al. [25] describe the problem of maintaining and determining the quality of data created by IoT-devices. They conduct a survey ag-

gregating the challenges and most relevant data quality dimensions for IoT-devices. Additionally, they list and describe a set of data enhancement techniques that can be used to improve the quality of data created by IoT-devices. That paper provided this thesis with data quality dimensions suitable for IoT-devices and enhancement techniques. However, the number of presented data quality dimensions were a bit limited, and its sources for the selected data quality dimensions were limited to only a few papers. Furthermore, the paper did not provide any techniques or implementations for data quality assessment. Klein and Lehner [26] investigated how a set of data quality dimensions should be used to present the quality of data in data streams that are generated by sensors (of smart-items), and how different processing techniques affect the quality of the data. They did however not focus on edge computing or the efficiency of assessing data quality.
Hence, there were no papers on data quality that presented implementations of data quality assessment techniques, or how data quality should be assessed on the edge.

4.3 Data Profiling

Naumann et al. present data profiling categories and discuss challenges that must be solved in data profiling [7] [8] [44]. To some degree, their papers on data profiling are redundant to each other, but there are some differences between them. The papers [7] and [44] are quite similar, with the differences that [7] is a survey paper that contains more in-depth information about specifically "traditional" data profiling for relational data, and [44] is part of a tutorial given by the authors, it presents the classes of data profiling and highlights directions for future research within the subject. [7] lists the classes and categories of data profiling techniques, and what information can be determined by those. [8] briefly explains new areas within data profiling that differs from the traditional data profiling on relational databases. It does not provide any concrete solutions, it mostly brings up problems and challenges that need to be solved, such as incremental data profiling and performing data profiling on non-relational data. The papers from Naumann provided a basis for the technique development of this thesis. However, the papers focused mainly on data profiling on relational data, which sensor data is not, furthermore they did not focus on data quality assessment.
Kusumasari et al. [6] use data profiling with the software Openrefine to determine the quality of a dataset, where the goal was to suggest how to improve the quality of data. The data they used was an already existing dataset on "medicine permits", however, the researcher did not investigate the efficiency and speed of determining the quality of data.
Dai et al. [21] examine different data profiling tasks, data profiling tools (software), and explains several data quality dimensions and how these dimensions can be aggregated into one data quality score/metric. While it explained and presented data profiling and data quality, it focused on using already existing data profiling tools. Thus no specific techniques were discussed in that paper.
Hence, not much research has been done on data profiling on sensor data and data profiling for data quality assessment.

4.4 Data Enhancement

There is some research on enhancement of sensor data, for instance, Zhang et al. [45] conducted a survey on outlier detection techniques that can be applied on wireless sensor networks. It categorizes the outlier detection techniques into five main classes: Statistical-based, Nearest Neighbor-based, Clustering-based, Classification-based, and Spectral Decomposition-based. Some of these classes have multiple subclasses. For each class and subclass of outlier detection techniques, [45] presents a paper describing that particular outlier detection technique and briefly explains it. [45] provided this thesis with outlier detection techniques.
Mandagere et al. describe data deduplication enhancement techniques in the paper [46]. It provides a taxonomy of data deduplication and presents multiple deduplication algorithms that are aimed towards performing data deduplication on data files.
Maletic and Marcus [47] gives an overview of data cleansing (or cleaning) and gives a list of methods of how data cleansing is done. The methods proposed by that paper are: Statistical, clustering, pattern-based, and association rules methods.
Berka [48] brings up a data cleansing algorithm based on clustering that tackles the data quality issue of having contradictory data. Which, in this case, refers to data points belonging to different categories (classes) but having the same or similar values. This is a problem for machine learning classification tasks where a classifier tries to predict which class a data point belongs to. If multiple data points are very similar (or are the same) and belonging to different classes, it will be impossible for the classifier to distinguish the data points from each other and to classify them correctly.
Hence, there is no one conclusive research on enhancement techniques for sensor data, even though some papers cover subsets of this subject, like [45], which covers outlier detection on wireless sensor networks well, however, outlier detection techniques are just a subset of all possible enhancement techniques.

4.5 Remarks of the Related Work in Relation to this Thesis

Due to the lack of research and information regarding choosing data quality dimensions and determining and enhancing the metrics of these data quality dimensions in a manner that makes it suitable for an edge computing device, this thesis aims to connect the research on the individual subjects to create a better understanding of data quality assessment in edge computing. Furthermore, the goal was to find which data profiling techniques can be utilized to enable data quality assessment efficiently, and how these data profiling techniques are implemented or utilized.

5
Methodology

This thesis aims to determine which data quality dimensions are relevant and can be applied universally to sensor data. This addresses the research question:

- Which data quality dimensions are relevant to determine the quality of sensor data?

To answer this question, a literature study was conducted. Given the answers to this first research question, we answer the remaining research questions, which are:

- Can data profiling efficiently be used in edge computing to assess the quality of sensor data, and in that case, how?
- Could sensor data of bad quality be efficiently enhanced?

It is necessary to establish which dimensions are relevant since otherwise, it would not be possible to determine which data profiling techniques would be used and how the data should be enhanced.
To answer the data profiling question, different techniques have been implemented and evaluated based on how efficient they are. To determine this, the evaluations are done by a set of criteria that are defined in this chapter.
To answer the research question about data enhancement, a list of possible techniques is presented that theoretically could improve the quality of sensor data. Furthermore, two basic enhancement techniques were implemented as a proof of concept.

5.1 Design Science Methodology

During the research in this thesis, the *Design Science* methodology was used. Design science is a set of analytical techniques. It is usually used when creating an innovative artifact whose purpose is to improve a process [49]. This methodology was chosen since the purpose of this thesis is to develop a framework (i.e., an artifact) to improve the collection and processing of sensor data (i.e., a process).
During the process of design science, there are two main parts, the collection of knowledge through developing the artifact, and the analysis of that artifact. The artifact needs to be innovative to generate novel knowledge, otherwise, the creation step is called a routine step. Design science can also use a routine creation, but in that case, only the analysis step is part of the research [49].
The process of design science is iterative and performed in cycles, given a problem, a suggestion on how to solve this problem is made. This suggestion is then designed, developed, and evaluated, based on this, a conclusion is made. If more problems exist or there is room for improvement, propose a new suggestion and continue. This

is done until there is no room for improvement, if the current solutions are sufficient, or there is no time left.
The design science process consists of three types of cycles, the relevance cycle, design cycle, and rigor cycle [50]. Relevance cycles are used to derive the requirements of the artifact given the environment and perform field testing to assess the relevance of the performed research. A design cycle could be seen as the core of design science research. It consists of the "two main parts", designing and developing an artifact and evaluating the current state of the artifact. A rigor cycle consists of either grounding of the research or making additions to the knowledge base using the results of the performed research. This process makes use of scientific theories and methods, experience and expertise in the researched subjects, and meta-artifacts [50].
In this thesis, the development of the framework was done by using the three cycles of design science research. The requirements given by the relevance cycles was that the framework must be able to assess all the data quality dimensions selected from the literature study, the proof of concept enhancement techniques must be able to improve the quality of data, and the framework's techniques would need to be done in a reasonable amount of time. "A reasonable amount of time" is determined by the update frequency of the sensors producing data that is evaluated by the framework and its implemented techniques. If the framework process a data point more slowly than it receives newly generated data, it will gradually fall behind new data, which will result in out-of-date advice from the framework. In the design cycle of this thesis, the framework and the various techniques were designed and created. Furthermore, the techniques were evaluated by a set of criteria. The rigor cycles of this thesis provided the research with information about the relevant topics, i.e., the grounding. Furthermore, rigor cycles were used to present the additions that this thesis added to the knowledge base. The addition of this thesis includes the lessons learning during the evaluation of the framework, and the architecture of the framework.
Some additions this thesis provided to the knowledge base:
The implementation of the data profiling-based quality assessment techniques is an addition to the knowledge base since as shown in Section 4.2 and Section 4.3, there are no papers that present implementations of data quality assessment techniques and how data profiling should be applied to process sensor data for the purpose of data quality assessment. The existing knowledge this thesis used was the information regarding how various data quality dimensions were defined and computed, and which data profiling categories are available and how those are applied.
Since it does not exist any research which presents concrete implementations of data profiling-based quality assessment techniques, it does not exist any evaluation of such implementations. Thus, the evaluations of the various techniques are an addition to the knowledge base.
The architecture of the framework is explained and presented in this thesis. The architecture can be seen as a meta-artifact of this research and is thus an addition to the knowledge base. Someone wishing to create a similar framework for the purpose of processing sensor data can base their framework of the architecture provided by this thesis.

The implementation of the design science research in this thesis:

1. Relevance Cycle:
   (a) Determining the requirements of the framework and the various techniques.
2. Design Cycle:
   (a) Implementation of the framework and the techniques.
   (b) Evaluation of the implemented techniques based on a set of criteria.
3. Rigor Cycle:
   (a) Grounding: Research the current state of data quality dimensions, data profiling, and data enhancement.
   (b) Additions to the knowledge base: Observations made during evaluations, conclusions to the research questions, and detailed implementations of data profiling-based quality assessment and enhancement techniques.

The selection of data quality dimensions that were in the requirements of the relevance cycle is explained in Chapter 6. How each technique is evaluated is presented in Section 5.3.

The iterations during the research and the development of the framework were as follows:

1. Rigor Cycle 1: Grounding: Performed a literature study on data quality dimensions.
2. Rigor Cycle 2: Addition to the knowledge base: Synthesized a list of data quality dimensions relevant for sensor data, thus answering the research question "Which data quality dimensions are relevant to determine the quality of sensor data?".
3. Relevance Cycle 1: Requirements: Established the requirement that the framework needs to be able to assess data on the selected data quality dimensions from rigor cycle 2.
4. Rigor Cycle 3: Grounding: Researched data profiling and which categories of data profiling exist.
5. Design Cycle 1: Implemented the initial framework with two techniques: completeness and accuracy assessment.
6. Design Cycle 2: Implemented the remaining data profiling-based quality assessment techniques.
7. Rigor Cycle 4: Addition to the knowledge base: Presented the framework architecture and the implementations of the techniques (including pseudocode).
8. Design Cycle 3: Designed the test environment in which the techniques are evaluated.
9. Relevance Cycle 2: Requirements: Derived the requirement that the techniques must be able to process data faster than 10 data points per second.
10. Design Cycle 4: Performed evaluations on the implemented data profiling-based quality assessment techniques.
11. Rigor Cycle 5: Addition to the knowledge base: Presented the result of the evaluations on the data profiling-based quality assessment techniques in design cycle 4, and thus together with rigor cycle 4 answers the research question "Can data profiling efficiently be used in edge computing to assess the quality

of sensor data, and in that case, how?"
12. Rigor Cycle 6: Investigated and found relevant data enhancement techniques.
13. Relevance Cycle 3: Requirements: Established the requirements of the data enhancement techniques, which were: the enhancement techniques should be able to improve the quality of sensor data and should be able to process data faster than 10 data points per second.
14. Design Cycle 5: Implemented a few proof of concept enhancement techniques.
15. Design Cycle 6: Performed evaluations on the proof of concept enhancement techniques.
16. Rigor Cycle 7: Addition to the knowledge base: Presented the result of the evaluations of the implemented (proof of concept) enhancement techniques, and thus answering the research question "Could sensor data of bad quality be efficiently enhanced?".

5.2 Selection of Data Quality Dimensions

To find data quality dimensions and decide which data quality dimensions are relevant for assessing sensor data, a literature study was conducted. The study was done by finding and collecting papers and books on the subject of data quality. The approach of collecting relevant sources for this was to search for books and papers on the search engine *Google Scholar* [51] with the search terms "Data Quality", "Data Profiling Data Quality", "Data Quality Sensor Data", "Data Quality IoT", and "Data Quality Big Data". The search engine Google Scholar gives the sources which are most relevant for the search, i.e., the sources which match the search term well and have a lot of citations. This is important since sources with a lot of citations are generally more credible. When a source was found, it was examined to determine whether or not it actually was relevant for the literature study, i.e., if it contained information concerning data quality dimensions. If it was relevant and contained information about data quality dimensions, it was put into the list of sources of the literature study. This was continued until all relevant sources with high citation counts were collected and some of the found dimensions could be distinguished from the rest due to the higher number of sources presenting them.
The collected sources were on the topic of data quality of sensor data or IoT-devices, data quality of big data, or data quality in general. The reason why the collection of sources were not limited to data quality on sensor data or IoT-devices was due to the limited number of available papers. By including papers on data quality of big data and data quality in general allowed the literature study to include more sources. It also helped to create a broader view of data quality. By collecting many sources and prioritizing sources with many citations, it ensured that no important data quality dimension for the context of this thesis was missed. The various data quality dimensions which were presented and explained in the selected papers were aggregated into a table. The table contained the found data quality dimensions and the number of papers each data quality dimension was presented in. The idea of this was that data quality dimensions which were presented in many papers are probably more important, as it shows that there exists a consensus that a particular data quality dimension should be used. Data quality dimensions that were only pre-

sented by one paper and dimensions which were vague, had no consistent definition, or were redundant were not put into the table.
Furthermore, given the scope of this thesis and the fact that this thesis is limited to sensor data, some requirements on data quality dimensions were derived to ensure that the selected dimensions are suitable and feasible for the scope of this thesis. The selection of data quality dimensions was based on the derived requirements and the literature study. The selection was split up into two subselections: a main selection and a secondary selection. The dimensions in both of the subselections need to satisfy the derived requirements. The importance of a data quality dimension decided if it were to be put into the main or secondary selection, the basis of this was the number of papers in which a dimension was presented. The requirements, the study, and the final selection of data quality dimensions are presented in Chapter 6.

5.3 Evaluation of Data Profiling-Based Quality Assessment Techniques

From the selected data quality dimensions, data profiling-based quality assessment techniques were developed. The implemented techniques were based on data profiling categories and developed to follow the definitions of the data quality dimensions. The techniques implemented in the framework were evaluated in terms of their performance, specifically, time and space complexity, and runtime on data. These factors are important to assess due to the computational limitations of edge devices. If a technique is too slow, it would not be able to keep up with newly generated data, and if it required too much memory space, it would not be possible to execute it on an edge computing device.
Each data profiling-based quality assessment technique was evaluated by:

- Time complexity of processing a single data point by Big O and Big Omega notation.
- Space complexity by Big O and Big Omega notation.
- Time it takes to process a dataset, this is explained in Section 8.1.

The time and space complexities of the techniques were derived by analyzing their source code manually. The Big O and Big Omega notations of the time and space complexities show which variables the required time and space of an algorithm are dependant on, and how it grows as the variables increase. Big O shows the worst-case scenario, i.e., the slowest case, or the case that takes as much space as possible. Big Omega shows the expected scenario, i.e., the time it will take on average, and the space that is required on average.
The techniques were evaluated by their time and space complexity since they reveal which variables the runtime and required space of a technique is dependant on. Techniques that have a polynomial (or worse) time complexity can get very slow as the size of the dependant variables increases. If the space complexity is bad, the space required of the device will be high, which might make the technique infeasible. The techniques were evaluated by how long time it takes for them to process a dataset since it will reveal how fast each technique is. By knowing how fast a technique is, it is possible to infer whether or not it is feasible to use a technique

in real-time. The runtime of each technique was derived by running them on five different scenarios. The five scenarios were created based on a dataset containing sensor data provided by Volvo Car Corporation. The various scenarios had different sensor configurations and dataset sizes, as explained in Section 8.1. Each technique was run in each scenario 30 times to control for non-determinism. The runtimes were measured by computing the difference in time between when the framework started the analysis (assessment) and the time when all the data had been analyzed (assessed).

5.4 Selection of Data Enhancement Techniques

We selected data enhancement techniques by investigating enhancement techniques and algorithms presented in various papers on data enhancement. The approach of finding enhancement techniques was based on the paper [25]. [25] presents data quality dimensions relevant to IoT-devices and various categories of techniques to improve the data of IoT-devices. That paper was relevant for this thesis since the data that is created by IoT-devices is sensor data. The presented enhancement techniques in [25] were further researched by finding papers about those techniques. The approach of finding relevant sources (other than [25]) was to collect papers found by Google Scholar. The search terms used in Google Scholar were the various enhancement categories relevant for this thesis, more specifically: "Data Deduplication", "Interpolation", "Clustering And Outliers", and "Outlier Detection". The search terms were used in conjunction with "on sensor data", "sensor data", "sensors", "IoT", or just by itself. To ensure that the techniques were suitable for this thesis, the presented techniques in the sources would have to be based on one of the selected enhancement categories, i.e., data deduplication, interpolation, or clustering and/or outliers. Furthermore, the presented techniques need to be aimed toward sensor data or data with similar characteristics to sensor data (i.e., continuous data).
Additionally, two enhancement techniques were implemented as a proof of concept to display that such techniques can actually improve the quality of the data. Like the data profiling techniques, the proof of concept enhancement techniques are evaluated based on their time complexity, space complexity, and runtime, see explanation in Section 5.3. Furthermore, the result of the enhancement techniques was presented (i.e., enhanced data), along with the metrics of improvement in data quality, to make sure that the techniques actually improved the quality, and thus answering the third research question.

6
Data Quality Dimension Selection

This chapter presents the selection of the data quality dimensions which were deemed to be relevant for sensor data, and thus answering the first research question: "Which data quality dimensions are relevant to determine the quality of sensor data?". The selection was based on a literature study and rules derived from the characteristics of sensor data. The data quality dimensions used and mentioned in this chapter are described in Chapter 3.1. The chosen data quality dimensions are the dimensions used by the developed data profiling framework to determine the quality of a dataset.

6.1 Evaluation of Data Quality Dimensions

Since there is no definite standardized method for assessing data quality, there is no definite rule on how to define data quality. However, by focusing on sensor data, one can derive criteria to filter out irrelevant dimensions. Relevant data quality dimensions must be:

- Intrinsic or contextual
- Domain-agnostic
- Quantifiable
- Indicate whether the data suffer from any actual problems.

As explained in section 3.1.1, the data quality dimension categories representational and accessibility are irrelevant for this thesis. Representational dimensions are irrelevant since sensor data does not need to be easy to read or understand since there is no need for humans to inspect raw sensor data regularly. Accessibility is not relevant in this thesis since the data being used and analyzed are assumed to always be available. The remaining categories of data quality dimensions are intrinsic and contextual. Thus all the dimensions that were selected should be either an intrinsic or a contextual data quality dimension.

Since the purpose is to develop an agnostic framework that should be compatible with any sensor data, it follows that the dimensions should be domain-agnostic. This is trivial since it is not possible to infer the metric of a domain-specific dimension on any arbitrary data since, by its nature, it is specific to one specific company/purpose/sensor.

The dimensions should be quantifiable since otherwise, it would not be possible for a framework to determine and differentiate the quality metrics of different datasets numerically. In the context of data quality, quantifiable means that it is possible to assign a numeric value representing how good the data is regarding a data quality dimension. For example, to assess completeness, we assign a dataset the ratio of the

number of present values to the number of expected values, thus a value between 0 and 1. When a dimension is quantifiable and given by a value within a continuous range, the quality of various datasets can be compared, and potential flaws can be highlighted. Also, by having a dimension on a continuous range, one can easily see the improvement made by enhancement techniques.
The dimensions should give a good indication of whether there are any fundamental problems with the data. There might exist many dimensions that are possible to calculate on sensor data, but they do not highlight or indicate any underlying problems in the data, and is hence not of interest. For example, it is possible to sum all the sensor readings' values and present the sum as a metric. Trivially this would (in most cases) not give any relevant information about the quality of the data. To determine if a dimension is a good indication or not, it must be possible to argue that the dimension is directly or indirectly correlated to some problem that affects the dataset.

6.2 Literature Study

To determine which data quality dimensions should be applied when determining the quality of sensor data, a literature study was conducted that investigated which data quality dimensions were most commonly used in the literature. The literature study was conducted by collecting research papers and books that contained information or research on data quality dimensions. Since the purpose is to find data quality dimensions that are suitable for sensor data, the most relevant approach would be only to include papers that were directly or indirectly related to sensor data and IoT-devices. However, the number of available papers that mentions data quality concerning sensors and IoT-devices was limited. In 2019, Ahmed et al. [52] analyzed and mapped papers on IoT to categories. Only three papers analyzed were about data quality.
Due to the limited number of papers on data quality on sensor data and data from IoT-devices, papers explaining data quality in general (i.e., not concerning sensor data) and papers explaining data quality in the field of big data were also included. The reason for including big data papers is because sensor data can be viewed as belonging to "big data" due to its size and being non-relational. The papers are split up into three tables, one table with papers related to sensor data and IoT, one with papers related to big data, and one table with papers describing data quality in general (the last table is split up to two tables, due to those papers containing more dimensions that would not fit in just one table).
The number of occurrences of a data quality dimension indicates how important it might be since if many researchers use the same data quality dimension, it is probably useful.
The tables will only contain intrinsic and contextual dimensions since, as explained in Section 2.3, representational and accessibility dimensions were deemed irrelevant for sensor data (in the context of this thesis). Dimensions that were mentioned by only one paper are not present in the tables. Some dimensions were also avoided due to the dimensions being vague, no consensus of meaning, or entirely dependant on other dimensions (which makes it redundant).

Table 6.1: Sensor and IoT related papers.

	Acc	Conf	Comp	D Vol	Tim	Dupl	Prec	Vali	Cons
[26]	x	x	x	x	x				
[25]	x	x	x	x	x	x			
[53]	x	x	x						
[54]					x				
[55]	x		x		x		x		
[56]	x		x						
[57]					x			x	
[58]	x*		x*			x**	x*		x
#	6	3	6	2	5	2	2	1	1

Table 6.2: Big data papers.

	Acc	Conf	Comp	D Vol	Tim	Dupl	Prec	Vali	Cons
[21]	x		x		x	x***		x	x
[30]	x		x		x				x
[31]	x		x						x
#	3	0	3	0	2	1	0	1	3

* = Uncertainty, ** = Redundancy, *** = Uniqueness

Acc = Accuracy, Conf = Confidence, Comp = Completeness, D Vol = Data Volume, Tim = Timeliness, Dupl = Duplicates, Prec = Precision, Vali = Validity, Cons = Consistency

Table 6.3: General data quality papers A.

	Acc	Conf	Comp	D Vol	Tim	Dupl	Prec	Vali	Cons
[22]			x		x			x****	
[59]	x		x		x				
[60]	x		x		x				x
[61]	x		x				x		x
[28]	x		x						x
[62]	x		x		x	x	x		x
[29]	x		x		x	x		x****	x
[24]	x		x		x				x
[63]	x		x		x		x		x
[23]	x		x		x				
#	9	0	10	0	8	2	3	2	7

Table 6.4: General data quality papers B.

	AAD	Beli	Obj	Rele	Repu	V-A	Cur
[22]	x	x	x	x	x	x	
[59]	x	x	x	x	x	x	
[60]							
[61]							
[28]							x
[62]							
[29]	x	x	x	x	x	x	x
[24]							x
[63]							x
[23]	x	x	x	x	x	x	
#	4	4	4	4	4	4	4

**** = Free of error

Acc = Accuracy, Conf = Confidence, Comp = Completeness, D Vol = Data Volume, Tim = Timeliness, Dupl = Duplicates, Prec = Precision, Vali = Validity, Cons = Consistency, AAD = Appropriate Amount of Data, Beli = Believeability, Obj = Objectivity, Rele = Relevancy, Repu = Reputation, V-A = Value-Added, Cur = Currency

Table 6.5: Sorted list of the aggregated number of papers each dimension is presented in.

	Number of papers
Completeness	19
Accuracy	18
Timeliness	15
Consistency	11
Duplicates	5
Precision	5
Currency	4
Validity	4

	Number of papers
App. Amount of Data	4
Believeability	4
Objectivity	4
Relevancy	4
Reputation	4
Value-Added	4
Confidence	3
Data Volume	2

From these tables, we can see that a majority agreed that the dimensions completeness, accuracy, and timeliness should be used or at least considered when determining the quality of data. This agreement can be seen in all of the three tables.
The dimension with the fourth most papers mentioning it is consistency, with 11 out of the total 21 papers mentioning it. However, among the eight papers in the sensor and IoT table, only [58] mentions consistency. That paper defines this dimension as an approach to "ensure that multiple copies of the same data are identical since server failures and parallel storage may cause inconsistency" [58]. That approach of computing consistency is not relevant for this thesis since this thesis focuses on assessing data quality on the edge, close in proximity to the source. In that context, there is only one copy of the data. However, [31] defines consistency as the consistency in quality among data points in a dataset, which can be relevant and applicable for this thesis due to it would entail computing consistency on factors (i.e., data quality dimensions) that are available.
The remaining dimensions only have at most five papers mentioning them, out of 21. Nonetheless, some of them might provide informative insights. The dimensions duplicates and precision are both mentioned five times, two of which by sensor and IoT papers. The problem of duplicates should not be a problem among sensors since sensors will continuously create new unique readings. However, theoretically, it might be possible for a sensor to somehow malfunction and send duplicates of a reading. Regarding the precision dimension, it might provide some insights, but it is somewhat similar in purpose to accuracy, it might thus be redundant or unnecessary to use a precision metric in conjunction with accuracy.
The currency dimension is neither mentioned by any sensor/IoT papers or any big data papers. Nonetheless, it might be informative to use since it would indicate if a sensor has stopped working, which would be a critical problem.
The dimensions appropriate amount of data, believability, objectivity, relevancy, reputation, and value-added are all mentioned, and only mentioned in the four papers [22], [23], [29], and [59]. The similarity between those papers might be due to that *Richard Wang* wrote [22], [23], and [59], and the fourth paper [29] is referring to [22]. This creates an obvious bias, making the six dimensions less crucial.
The dimension validity is mentioned by four papers, one of which from the sensor and IoT table.

Confidence is only mentioned by the three papers [26], [53], and [25], all of these papers are from the sensor and IoT table. Among these three papers, the papers [26] and [53] are both written by *Anja Klein*, and [25] are referring to [26]. This means that basically, only one researcher describes/uses this dimension, which entails that this dimension is less important. Additionally, it is quite similar to the accuracy and precision dimensions, and might thus be redundant or offer little to no added information about the data.
The dimension with the lowest number of papers mentioning it is data volume, with only two papers, both of which are from the sensor and IoT table.

6.3 Selection Results

From the literature study, there is a clear divide between the various dimensions. Some of the dimensions are referred to in almost all papers, and some of them are referred to only a few times. The final dimension selection is divided into two lists, one list of dimensions that should be applied when determining the quality of sensor data. The other list contains dimensions that could be applied to sensor data, but are only relevant in some contexts. Furthermore, some of the dimensions were excluded from both lists because they could either not be calculated by a computer (i.e., not possible to objectively quantify), are domain-specific, or do not provide any relevant information about the underlying data and thus are unnecessary.
Techniques that determine and enhance data quality were developed/presented for both the main and secondary data quality dimension lists.

6.3.1 Main Selection

The main selection of data quality dimensions are:

- Completeness
- Accuracy
- Timeliness

These were selected because they were mentioned by a majority of the papers and books from the literature study. Furthermore, they were selected because:

6.3.1.1 Completeness

Completeness is contextual, domain-agnostic, and easily quantifiable. Completeness shows to what extent data is missing. If many data points are missing, there might be something wrong with the source of the data, i.e., the sensors are faulty. Sensors being faulty and not sending the required amount of data could be a significant problem.

6.3.1.2 Accuracy

Accuracy is intrinsic, domain-agnostic, and easily quantifiable. Accuracy indicates to what degree the data in a dataset is accurate. Data that are inaccurate could

be a result of poorly calibrated sensors, making the sensor data less reliable and credible.

6.3.1.3 Timeliness

Timeliness is contextual, domain-agnostic, and easily quantifiable. Timeliness describes the delay between when a sensor reading is recorded and when the (in this case) framework receives the data from the sensor. If the delay is high (bad timeliness), there might be some problems in the system or the communication between the sensor and the framework. In time-critical systems, the timeliness dimension is important to assess since it measures the delay between the sensor and the destination. A time-critical system requires the timeliness to be good (i.e., a low delay), since otherwise the information and computations in that system would be delayed, which might not be acceptable in such a system.

6.3.2 Secondary Selection

The data quality dimensions in the secondary selection are:

- Consistency
- Currency
- Duplicates
- Precision

These were selected for the secondary list because:

6.3.2.1 Consistency

Consistency was referenced in many papers. However, most of the descriptions about consistency were irrelevant for the task of collecting sensor data. Thus, it was not put into the main selection. However, Taleb et al. [31] defined consistency in a way such that it is relevant for the scope of this thesis. They defined it as the consistency in quality among data points in a dataset.
By the definition of consistency from [31], consistency is contextual, domain-agnostic, and quantifiable. While it is basically an extension of other data quality dimensions, it offers additional information about the data. Consistency will show if there have been any changes to the set of sensors or the system during the collection of sensor data. A dataset could have a fairly high quality concerning other dimensions, however, at some point (in time), something in the system could have been damaged, and other quality dimensions might not spot this change if it is small.
An example of this is if a sensor gets damaged halfway through the collection of data. The sensor might still be working, but be slightly less accurate. This will cause the, for instance, accuracy metric to get slightly worse. A slight change might not be recognized on its own. In this instance, the consistency metric on the accuracy would be bad and indicate that something might have happened. This could be important since slight declines in the quality can be detrimental in some cases.

6.3.2.2 Currency

Currency is contextual, domain-agnostic, easily quantifiable, and displays the delay between the current time and the last update, which can indicate if the sensors have stopped working. The reason for picking this dimension to the secondary list is that it can be seen as a complementary dimension to timeliness. Whenever the currency is bad, the timeliness will most likely also be bad. However, assume the situation of, for a given period, the timeliness has been good, and then abruptly, the sensors stop working. No more sensor data will be sent or received, this will result in the timeliness staying constant (until the next sensor reading is received), and the currency will decrease for each second of this scenario. In that scenario, currency would be beneficial to use. However, timeliness can be extended so that it covers currency, and due to timeliness being more common than currency (over three times according to literature study), the currency dimension was put into the secondary list.

6.3.2.3 Duplicates

The duplicate dimension is contextual, domain-agnostic, easily quantifiable, and shows if there are any duplicates in the dataset, which could be a problem since it is redundant to have duplicates. To measure the number of duplicates could be unnecessary since, in some contexts, the presence of duplicates is impossible. Duplicates might be non-existent because the sensors are designed to send data only once, or the data is already pre-processed and all duplicates are already removed. As the number of papers referring to duplicates is low, it may indicate that duplicates are rare in sensor data. It was thus not put in the main selection. However, in some cases, duplicates are a problem, and measuring it could be important. For instance, RFID data may contain duplicates since, according to [58], the same RFID tags can be used multiple times at the same location, and thus creating duplicates in the dataset. Furthermore, according to the sensor company *Monnit*, duplicates in sensor data can occur from sensors that require an acknowledgment from the receiver [64]. The acknowledgment is a response indicating that the receiver has received the sensor data. Duplicates occur when a sensor transmits data, and there is a significant delay before the receiver receives the data. In that case, the acknowledgment will also be delayed, causing the sensor to think that the receiver has not received the data, therefore, thinking the data is lost. Since the receiver has not sent an acknowledgment in time, the sensor sends to same data again. This will result in that the receiver receives the same data twice [64]. The same situation can also occur if the acknowledgment from the receiver is delayed.
Since there exist some instances where duplicates are a problem, it was selected as a relevant dimension.

6.3.2.4 Precision

Precision is intrinsic, domain-agnostic, and quantifiable (by computing the standard deviation). It is used to determine the precision of, for instance, sensors, which is important since then we know how precise the data and the sensors are. If, for

instance, the data analyzed fluctuates widely, it will be noticed by calculating the precision. However, the purpose and result of determining precision are very similar to accuracy since accuracy also is calculated by fluctuations in data. Thus, precision may offer little additional insights when used jointly with accuracy. Since accuracy has many more papers referring to it, accuracy was chosen to the main selection, and precision was put into the secondary selection. Even though precision is similar to accuracy, it might be relevant in some cases.

6.3.3 Remaining Dimensions

The data quality dimension that was deemed unnecessary, infeasible, or irrelevant for sensor data in general were:

- Confidence
- Validity
- Appropriate Amount of Data
- Believeability
- Objectivity
- Relevancy
- Reputation
- Value-Added
- Data Volume

6.3.3.1 Confidence

Confidence is intrinsic, domain-agnostic, quantifiable, and describes the statistical error, which could highlight uncertainties in a dataset. However, it is almost entirely redundant to precision. Precision is determined by the standard deviation. The size of the range in a confidence interval, i.e., the metric of the confidence dimension, is determined by the Z-value, the number of values, and the standard deviation. The Z-value is determined beforehand, and the number of values is known (usually the number of sensors), the only varying variable is the standard deviation, see Section 3.1.2.7. The value of confidence is determined entirely by the range of the confidence interval. Since the only varying variable of the computation of this interval is the standard deviation (i.e., the metric that gives precision), confidence is just a derivative of the precision dimension, and is thus redundant. The reason for choosing precision over the confidence dimension is because precision was referred to more often in the literature. Furthermore, precision is slightly easier to compute, and it does not require an additional argument (when using confidence, an arbitrary Z-value is required).

6.3.3.2 Validity

To determine the validity of a dataset, a set of rules to determine which data points are incorrect needs to be established. The selection of rules depends on the specific task. The validity is thus a domain-specific data quality dimension and was thus deemed irrelevant.

6.3.3.3 Appropriate Amount of Data

To determine whether or not the amount of data is appropriate, it is ultimately dependant on the specific task which is supposed to use the data. This dimension is thus domain-specific. Since this dimension is domain-specific, it was not selected.

6.3.3.4 Believability

Believability is contextual, domain-agnostic, and quantifiable, given that the other data quality dimensions can be normalized. However, it is redundant since it is determined completely based on the other data quality metrics. Believability was thus deemed to be unnecessary.

6.3.3.5 Objectivity

What constitutes a bias and how to detect biases is both difficult and specific to the context, since knowledge about the background of the data is required to be able to detect any obvious biases. Objectivity was thus deemed to be infeasible to quantify in the context of this thesis.

6.3.3.6 Relevancy

Relevancy is based on whether there is a specified application for the data, from which the relevancy metric will be decided based on how relevant the data is for the specific application. This dimension is thus domain-specific, and due to this, relevancy was determined to be irrelevant.

6.3.3.7 Reputation

Reputation is based on knowledge of the sources of data, which in this context is the sensors. Since it requires knowledge of the given environment, it is domain-specific. Furthermore, it is difficult to quantify this, and the metric for this dimension would theoretically stay the same during the entire data collection since the set of sensors will be fixed. Thus, the reputation dimension was deemed infeasible and irrelevant.

6.3.3.8 Value-Added

The value-added dimension is determined by how beneficial the dataset is given a specific task. Since this is specific for a particular task, this dimension is domain-specific. Furthermore, it is difficult to quantify this since the "value-added" is, in some regard, subjective. Due to these facts, the value-added dimension was deemed infeasible and irrelevant.

6.3.3.9 Data Volume

Data volume as a data quality dimension was presented in two of the papers and was the dimension with the fewest references. Furthermore, it does not describe any inherent problem with a dataset. The volume of data in the context of sensors can be calculated by the total time, the completeness, and the sensor frequency.

The sensor frequency is constant and can be assumed to be information that is known beforehand. Completeness is already a data quality dimension. Thus, data volume would only indicate the total time of a dataset, which does not highlight any problems with the sensors. Data volume as a dimension was thus deemed unnecessary.

7
Data Profiling Framework

The second research question of this thesis is:

- Can data profiling efficiently be used in edge computing to assess the quality of sensor data, and in that case, how?

To answer this question, we must ensure that the techniques used are efficient and reasonable to use with edge computing. To realistically determine these aspects, it is necessary to implement and run different techniques to see how good they are, and to compare them to each other. For this, a data profiling framework was developed.

7.1 Framework

To fit the scope of this thesis, the data profiling framework developed had to be suitable for edge computing, i.e., be able to receive and process continuous data and assessing data quality.

7.1.1 Framework Design

The data profiling framework is built such that the techniques are completely independent of each other. This enables easy addition of new techniques. This was important since multiple techniques were developed, some of which should be able to run at the same time. If the extension of new techniques was difficult, it would result in slower implementation and integration of new techniques to the framework. The independence of techniques enabled an easy configuration of a new sensor system. The framework is supposed to be used by attaching (connecting) the desired techniques to each sensor (or a set of sensors) that receives the data coming from the connected sensors.
The purpose of the framework is to ultimately be deployed on an edge device. As explained in Section 2.1, edge computing devices are located between the source of data and the cloud. An edge computing device receives and processes data from the source of data and sends the result to the cloud. Thus, the framework needs to be able to receive data, more specifically sensor data, and somehow be able to send the data quality metrics (i.e., the result) to a server. The framework was thus developed to be able to receive sensor data, see Figure 7.1. Furthermore, the framework is built such that it is possible to easily build data strings containing all the quality metrics, which then could be sent to the cloud or a server.

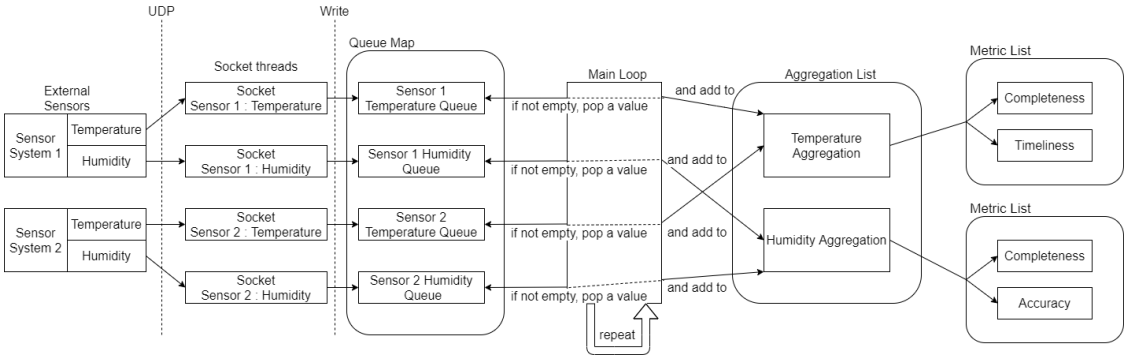


Figure 7.1: Overview of the framework used in a specific configuration.

7.1.2 Tools and Libraries

The programming language used to implement the framework was *Python 3* [65]. Python was chosen since there are many libraries available for Python that are suitable for this framework (mathematical and data processing libraries). However, by the end of the implementation, only one library not part of the standard Python library was used for the framework implementation. The only additionally used library was *Numpy* [66], which was used for the precision implementation to calculate standard deviation.

7.1.3 Architecture

The framework contains multiple parts, from receiving data from sensors to calculating the metrics. The following sections will explain each part of the framework. The overall architecture of the framework can be seen in Figure 7.1.

7.1.3.1 Receiving Sensor Data

In the context of a system with sensors, multiple sources might exist, and each source might measure multiple things. To clarify this, a sensor source might consist of a temperature and a humidity sensor at a specific geographical location. This sensor source measures two things; temperature and humidity. There might exist multiple such sensor devices located in different locations. Each sensor source will be referred to as a sensor system.
To allow communication and receive data from a set of sensor systems, a configuration file is given as an argument to the framework. This file contains the sensor systems and each sensor in every system, which provides the names for these (names are arbitrary), connection type (e.g., UDP), and which IP address and port data from each sensor are expected to be sent to, see Figure 7.2. Given a configuration file, the framework initializes sockets that are started in separate threads (i.e., in parallel), one socket for each sensor. Each of these sockets has a queue which they write to whenever they receive data, see Figure 7.3. These queues are thread-safe to avoid problems between the main thread and the socket threads.

```
{
    "sensors": {
        "Sensor1" : {
            "signals": {
                "Temperature": {
                    "connection_type": "udp",
                    "ip": "127.0.0.1",
                    "port": 9801
                },
                "Humidity": {
                    "connection_type": "udp",
                    "ip": "127.0.0.1",
                    "port": 9802
                }
            }
        },
        "Sensor2": {
            "signals": {
                "Temperature": {
                    "connection_type": "udp",
                    "ip": "127.0.0.1",
                    "port": 9811
                },
                "Humidity": {
                    "connection_type": "udp",
                    "ip": "127.0.0.1",
                    "port": 9812
                }
            }
        }
    }
}
```

Figure 7.2: Configuration file for receiving sensor data.

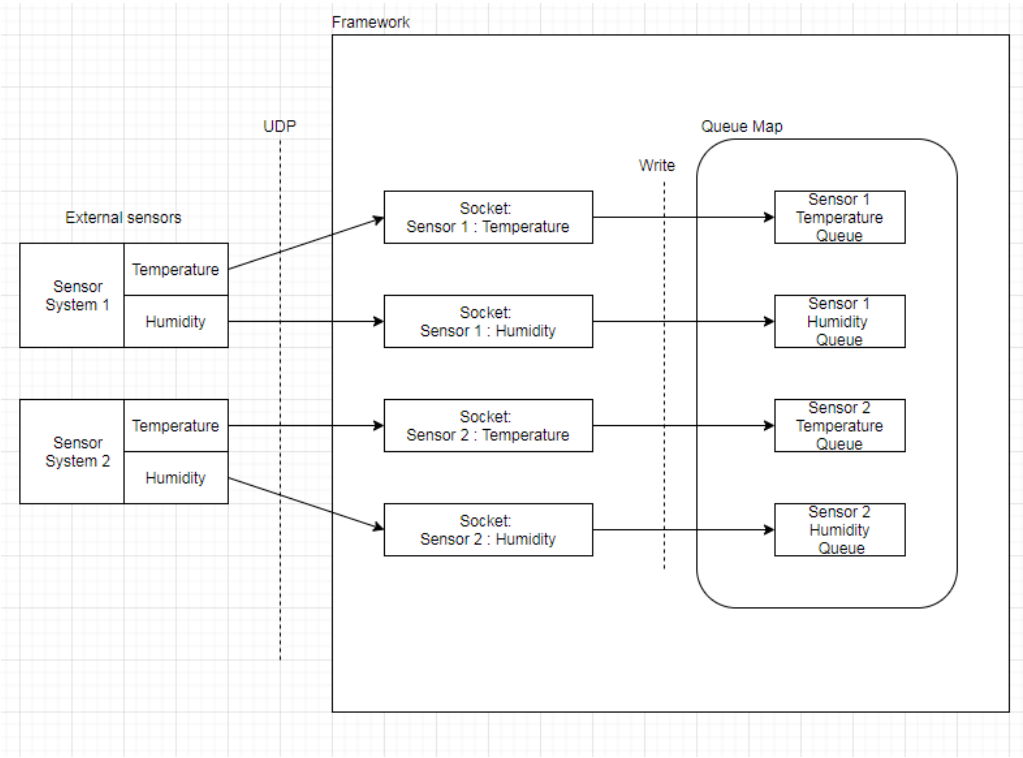


Figure 7.3: Overview of how the framework receives data from sensors.

7.1.3.2 Sensor Aggregation

When analyzing data, in some cases, it is preferable or necessary that multiple sensors are aggregated. For instance, consider the example presented above with multiple sensor systems, each with one sensor for temperature and one for humidity. In that scenario, it is logical to aggregate the temperature sensor values with each other, and the humidity sensor values with each other. Aggregating these sensors is relevant when, for instance, calculating accuracy or precision.
The things that are analyzed and assessed in the framework are so-called "sensor aggregates". A sensor aggregate is a mapping from a title (i.e., a sensor aggregate name) to a set of sensors. A sensor aggregate could be a single sensor or multiple sensors.
It is on sensor aggregates whom the framework is performing data profiling and data quality assessment.

7.1.3.3 Data Profiling Implementation

The implementations of the assessment techniques are called "Metrics", which are added to sensor aggregates. The metrics in the framework are independent of each other. However, they share some characteristics.
Every technique has a title indicating the quality dimension, a value indicating the metric of the data quality dimension, and an update function. The update function takes a timestamp and a list of sensor values as input. The sensor values come from the sensors which are in the sensor aggregate that the current metric is applied to, see Figure 7.4. The update function is called whenever the sensor aggregate has received values from each of its sensors at the same timestamp, i.e., the aggregate waits until it gets all data for a specific timestamp before it sends the values to its metric assessment techniques.
The update function is where the analysis (data quality assessment) is done. Each time the update function is called, the value of the metric is updated to reflect the current data quality of the continuous dataset.

7.1.3.4 Main Loop

When the sockets are initialized, sensor aggregates are created, and the desired metrics have been added to the aggregates, the framework can start with its analysis phase, i.e., the main loop. This phase will continue until the program is interrupted. This phase is a loop that continuously checks if any of the sensor queues have any data. Whenever data is found in a sensor's queue, it is removed from the queue (popped) and then added to all of the aggregates which contain that sensor, see Figure 7.5.
If any of the aggregates received new data, one metric could have been updated by the end of the loop. Thus, whenever an aggregate was updated, the framework will (in practice) send the metrics to some specified output, e.g., a server/cloud where the data quality metrics can be viewed. Alternatively, the framework could send the metrics within given intervals.

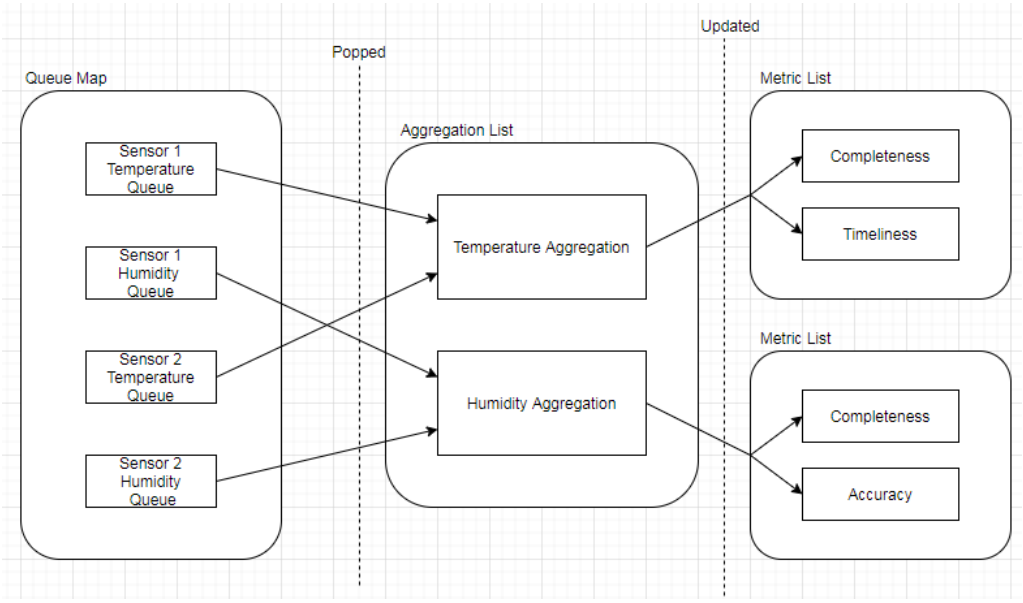


Figure 7.4: Overview of how data is sent sent from the queues to the analysis step.

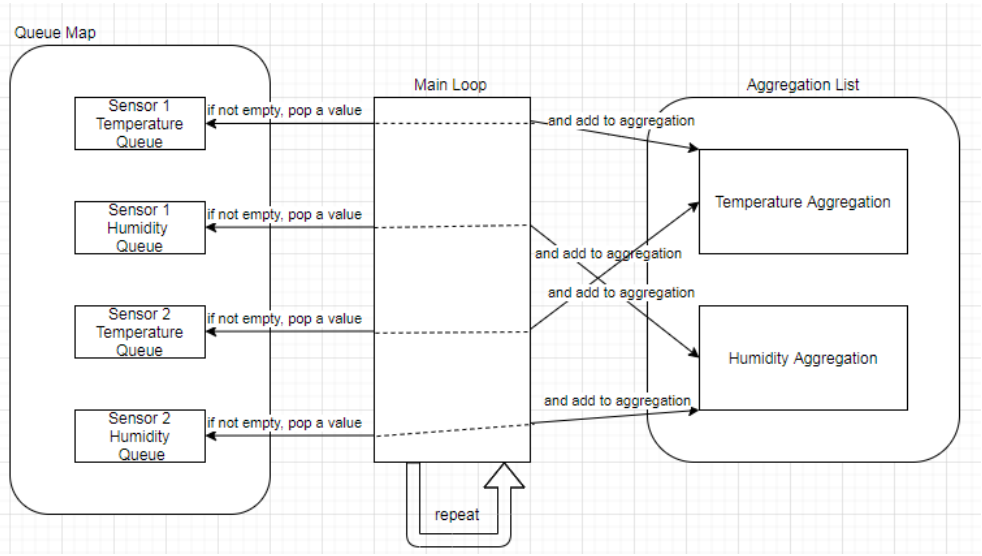


Figure 7.5: Actions in the main loop.

7.2 Test Environment

A test environment was created to properly evaluate the efficiency and speed of the different techniques and get realistic and unbiased results. The environment needs to be identical (i.e., fixed) when running the different techniques so that no technique receives an advantage or disadvantage due to some minor changes in the environment. The fixed aspect of the environment minimizes the chance of biases and noise.
The test environment consists of the hardware which the framework runs on and pre-generated datasets consisting of sensor data.

7.2.1 Equipment (Hardware)

The hardware that the framework was tested on was a *Raspberry Pi 3 Model B* [19]. It is a single-board computer with a quad-core 1.2 GHz CPU, 1 GB of RAM, and has the capability of adding miscellaneous modules like sensors through either USB-ports or GPIO pins. It is essentially the size of a credit card, specifically 85mm x 56 mm x 17 mm. It is recommended to use the Debian-based operating system Raspbian that was created specifically for Raspberry Pi computers.
This device was chosen due to its mobility, size, price (from 35$), limited CPU, and the ability to attach sensors to it. These aspects make this device a valid edge computing device. While the Raspberry Pi is not specifically designed as an edge computing device, it shares many qualities (small size, sensor compatible, limited computing power) with common edge computing devices. Therefore, we believe it provides a realistic case example.

7.2.2 Data Preparation

To ensure that all the techniques will be evaluated fairly, the environment and data they analyze should be as similar as possible. While testing and evaluating one technique, if, for instance, a sensor is behaving abnormally and is sending less data than expected, it would be an easier task for the technique, since it would analyze less amount of data than expected.
To make the testing and evaluations as fair and similar as possible, the data that they analyzed was not live real-time data. Instead, a set of pre-generated datasets was selected that the techniques were evaluated on. By evaluating the techniques on the same datasets, it assured that every technique was evaluated fairly.
As presented in Section 5.3, the techniques are evaluated on how fast they can assess a dataset. To enable this, all the data in a dataset were added to the queues before the analysis step. When all the data was added, the main loop was started and ran until the queues were empty. The techniques were evaluated by the length of the duration of the main loop.
By loading the data directly to the queues, the sockets of the framework were not used. This will cause the runtimes during the evaluations to be faster since one part of the framework is not running. However, as explained in Section 7.1.3.1, the sockets are running in parallel in separate threads, due to this, the sockets should

not affect the runtimes significantly.

7.2.3 Datasets

The datasets used for the evaluations were generated by AD cars and provided by Volvo Car Corporation. Due to the computational limitations of the testing hardware (i.e., the Raspberry Pi), only a fraction of the total dataset generated by an AD car was used. More specifically, only the timestamps and numeric values from four columns were used; velocity, pitch, roll, and yaw rate. The data points in the selected dataset were generated with the same car and during the same period (within the same hour). Each numeric column was separated together with the timestamp column into separate files, i.e., one file for each column (e.g., velocity). The frequency of the generated data was roughly 10 Hz (i.e., 10 values per second), and there were in total 16.000 data points in each for the four datasets. The reason for limiting the datasets to 16.000 data points was that having more data points would increase the time it would take to perform the evaluations of the techniques. It took almost 24 hours to perform all the evaluations in this thesis.
To enable the evaluation of multiple sensor systems, the original velocity, pitch, roll, and yaw rate datasets were replicated three times each. Namely, there were in total four velocity datasets, four pitch datasets, four roll datasets, and four yaw rate datasets. The replication of the datasets was done by taking the original datasets and adding Gaussian noise to their values. However, the timestamps remained unchanged.

8

Evaluation of Data Profiling-Based Quality Assessment Techniques

The goal of this chapter is to answer the second research question, which is::

- Can data profiling efficiently be used in edge computing to assess the quality of sensor data, and in that case, how?

This chapter lists all the implemented techniques to determine the data quality dimensions selected in Chapter 6. Furthermore, it contains the pseudo-code, time and space complexity, and the runtimes on the datasets of each implemented technique.

8.1 Evaluation Scenarios

The techniques were tested on five scenarios based on the datasets presented in Section 7.2.3. These five scenarios were:

- **Scenario 1**: 1 Velocity Sensor with 4.000 data points.
- **Scenario 2**: 1 Velocity Sensor with 16.000 data points.
- **Scenario 3**: 1 Velocity Sensor, 1 Pitch Sensor, 1 Roll Sensor, and 1 Yaw rate sensor, all with 16.000 data points.
- **Scenario 4**: 4 Velocity Sensors with 16.000 data points.
- **Scenario 5**: 4 Velocity Sensors, 4 Pitch Sensors, 4 Roll Sensors, and 4 Yaw rate sensors, all with 16.000 data points.

Note, sensors refer to sources (i.e. number of datasets). Each type of sensor (e.g. Velocity) is one sensor aggregate, as explained in Section 7.1.3.2.
These scenarios were selected to find how the runtime increased when the data points, the number of different sensors, and/or number of identical sensors increased. Every technique was run on every scenario 30 times to control for non-determinism in execution time. Based on the runs, boxplots by the runs of each technique were created and presented in this chapter, along with a table presenting the median runtimes of each technique-scenario pair, see Table 8.1. Important to note when viewing the figures is the fact that the datasets contain 16.000 data points (4.000 data points in Scenario 1), and the sensors which generated the datasets in the first place had a frequency of 10 Hz (10 data points/second). The techniques need to process a single data point faster than 10 Hz to be feasible, i.e., the time to process a single data point for a technique should not exceed 0.1 seconds. Hence, the techniques should have a significantly lower runtime than $16.000 \cdot 0.1 = 1.600$ seconds (or 400 seconds in Scenario 1).

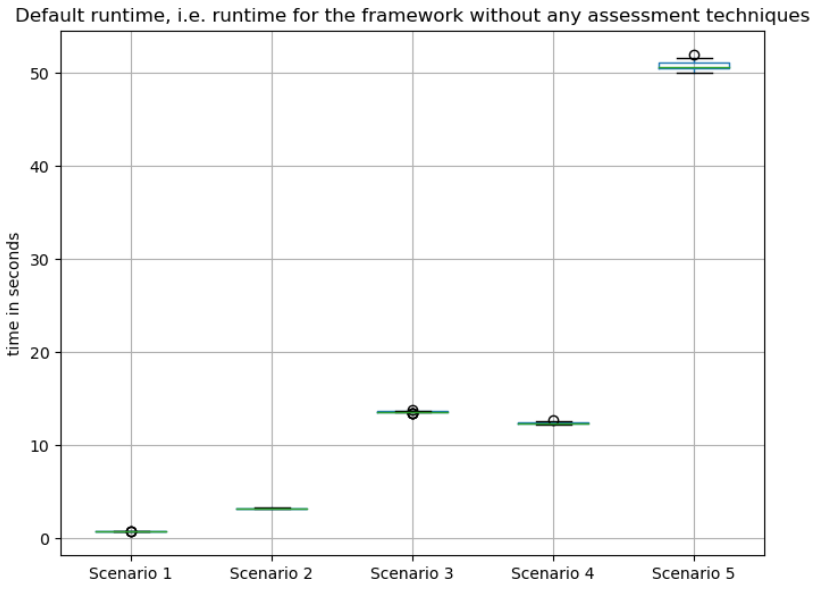


Figure 8.1: Runtimes for the framework without any techniques used in the various scenarios.

8.2 Implemented Techniques and Their Evaluations

In this section, all the techniques that were implemented are presented and described. Additionally, their time complexity, space complexity, and their runtimes on the scenarios listed in Section 8.1 are presented in this section.
The techniques being evaluated are only one part of the entire framework, the framework also contains the main loop, the handling of the queues and the sensors aggregates, these parts of the framework also contribute to the total runtime. To get a baseline to compare the techniques to, the framework was also evaluated without any added techniques, to see how long it took for just the framework to process the datasets. The runtimes for this can be seen in Figure 8.1. The runtimes in the default case are significant (50 seconds in Scenario 5) even though no techniques are applied. This is due to the other parts of the framework is still running, more specifically the queues, main loop, and the aggregations, see Figure 7.1. Note, as explained in Section 7.2.2, the sockets in the framework is not used since the data is directly loaded into the queues during the evaluations.
In the complexity analysis in this section, the number of sources is defined by s (i.e., Scenario 4 has four sources), and the total number of data points a technique has received thus far is defined as n.

8.2.1 Accuracy Techniques

Four accuracy techniques were implemented, each of which calculated accuracy slightly differently. All these techniques are based on the explanation given in Section 3.1.2.6, which is using the biggest difference of a set of values at a specific timestamp or specific time interval. Since these techniques take the biggest and smallest values, they are using the data profiling method cardinalities.

8.2.1.1 Basic Accuracy (Maximum difference)

As explained in Section 3.1.2.6, accuracy is defined as the biggest "data point difference" in a dataset, i.e., decided by the data point that has the two values with the biggest difference. In this context, it is computed as the biggest difference between values within a sensor aggregate at the same timestamp. This implementation requires that a sensor aggregate has two or more sensor sources connected to it. The pseudo-code for this technique can be seen in Algorithm 1.

Data: metric_value
input: new_data
delta = max(new_data.values) - min(new_data.values)
if *delta > metric_value* **then**
| metric_value = delta
end

Algorithm 1: Pseudo-code of the basic accuracy technique.

Complexity Analysis: The max and min operation requires a linear scan of all the values, thus, the time complexity of this algorithm is linear in the number of sensors used in an aggregate. The algorithm only requires storing one variable, thus the space complexity is constant. Hence, the time complexity is $O(s)$ and the space complexity is $O(1)$.

The runtimes of the basic accuracy technique can be seen in Figure 8.2.

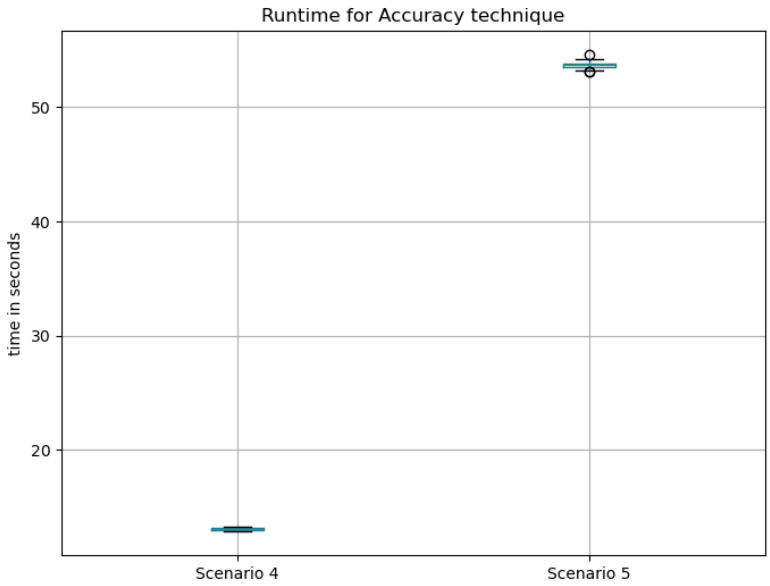


Figure 8.2: Runtimes for the basic accuracy technique in the various scenarios. Only Scenario 4 and 5 was used since this technique requires at least two sensor sources.

8.2.1.2 Average Accuracy

The average accuracy technique is similar to the basic accuracy technique, with the difference of instead finding the biggest difference, it calculates the average maximum difference of each data point. The pseudo-code for this can be seen in Algorithm 2.

Data: metric_value
Data: nbr_of_values
input: new_data
delta = max(new_data.values) - min(new_data.values)
nbr_of_values += 1
metric_value = metric_value * ((nbr_of_values - 1) / nbr_of_values)
metric_value += delta * (1 / nbr_of_values)

Algorithm 2: Pseudo-code of the average accuracy technique.

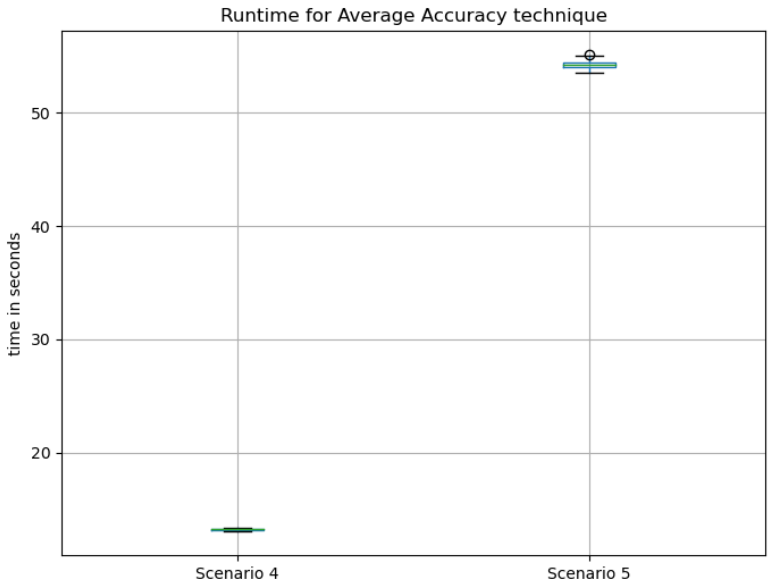


Figure 8.3: Runtimes for the average accuracy technique in the various scenarios. Only Scenario 4 and 5 was used since this technique requires at least two sensor sources.

Complexity Analysis: Since this implementation does not differ that much from the basic accuracy technique, the complexities of these techniques are the same, i.e., the time complexity is $O(s)$ and the space complexity is $O(1)$.
The runtimes of the average accuracy technique can be seen in Figure 8.3.

8.2.1.3 Accuracy Ratio

The accuracy ratio technique determines what percentage of the data points have a value difference that is below a specified allowed threshold. This technique is similar to the earlier accuracy techniques, with the difference that this technique will give a normalized metric value (between 0 and 1). The pseudo-code for the implementation of the accuracy ratio can be seen Algorithm 3.
Complexity Analysis: The time and space complexity of this implementation are the same as the previous accuracy techniques, i.e., the time complexity is $O(s)$, and the space complexity is $O(1)$.
The runtimes of the accuracy ratio technique can be seen in Figure 8.4.

Data: metric_value
Data: threshold
Data: nbr_of_valid
Data: nbr_of_invalid
input: new_data
delta = max(new_data.values) - min(new_data.values)
if *delta* $\leq$ *threshold* **then**
 | nbr_of_valid += 1
else
 | nbr_of_invalid += 1
end
metric_value = nbr_of_valid / (nbr_of_valid + nbr_of_invalid)

Algorithm 3: Pseudo-code of the accuracy ratio technique.

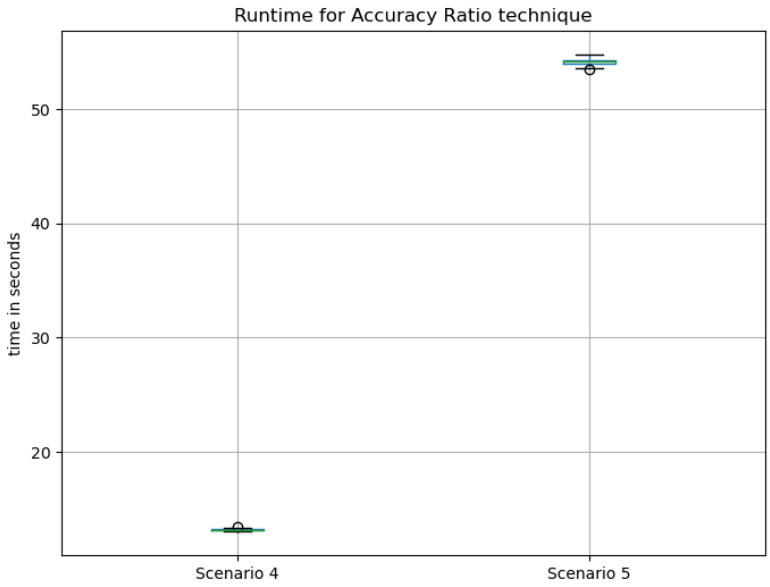


Figure 8.4: Runtimes for the accuracy ratio technique in the various scenarios. Only Scenario 4 and 5 was used since this technique requires at least two sensor sources.

8.2.1.4 Aggregated Accuracy

The previously presented accuracy techniques require two or more sensor sources. The aggregated accuracy aggregates values from multiple timestamps that are close to each other (in time) and computes the accuracy (i.e., max difference) between those. Due to this aggregation, single sensor sources can also be assessed regarding the accuracy dimension. This technique inputs the desired size of the aggregations (i.e., number of timestamps) and the delay of the sensors. The technique aggregates

the minimum and maximum value of all values received from a timestamps between t and $t + \textit{aggregation size} \cdot \textit{delay}$ to a hash map. The pseudo-code for the implementation of the aggregated accuracy can be seen in Algorithm 4.

```
Data: metric_value
Data: aggregation_size
Data: delay
Data: start_date
Data: hashmap (timestamp → (min,max))
input: new_data
if start_date is uninitialized then
 |  start_date = new_data.date
end
key_date = new_data.date - ((new_data - start_date) %
 (delay * aggregation_size))
max_val = max(new_data.values)
min_val = min(new_data.values)
if hashmap.contains(key_date) then
 |  max_val = max(max_val, hashmap.get(key_date).max)
 |  min_val = max(min_val, hashmap.get(key_date).min)
 |  hashmap.put(key_date, (min_val, max_val))
else
 |  hashmap.put(key_date, (min_val, max_val))
end
delta = max_val - min_val
if delta > metric_value then
 |  metric_value = delta
end
```

Algorithm 4: Pseudo-code of the aggregated accuracy technique.

Complexity Analysis: Two things in the code determine its time complexity. Firstly, the min and max operations are performed in linear time in the number of values (number of sources). Secondly, the hashmap operations are expected to be done in constant time but have a worst-case scenario of linear in the number of aggregations (i.e., key-value pairs). Since the max and min values of each aggregation of data points are stored in a hashmap, the space required is linear in the number of aggregations. The number of aggregations is roughly $\frac{n}{\textit{aggregration\_size}}$, where n is the number of data points (number of unique timestamps) the technique has received. However, in the worst case, many data points could be missing and that there is only one data point for each aggregation. Thus, the number of aggregations is bounded by n. Hence, the time complexity is $O(n + s)$ and $\Theta(s)$, and space complexity is $O(n)$ and $\Theta(\frac{n}{\textit{aggregration\_size}})$.
The runtimes of the aggregation accuracy technique can be seen in Figure 8.5. This technique was evaluated with the aggregation size set to 10.

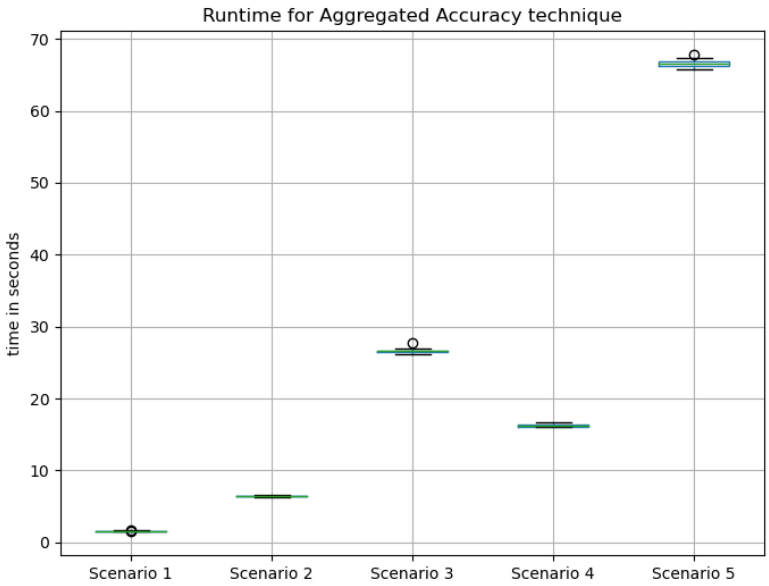


Figure 8.5: Runtimes for the aggregated accuracy technique in the various scenarios.

8.2.2 Completeness Techniques

Only one completeness technique was implemented. The technique was based on the data profiling category cardinality by using the min and max value on the timestamp column (i.e., the oldest and the most recent timestamps). It was implemented accordingly to the explanation of completeness given in Section 3.1.2.1, i.e., by taking an input for the expected frequency of the sensor, and computing the ratio of the number of data points it received to the number of data points that were expected given the interval between the min and max timestamp and the expected frequency. The technique was implemented such that it would work with unsynchronized data, i.e., the latest received data point was not necessarily the data point with the most recent timestamp, vice versa with the earliest data point. See Algorithm 5 for pseudo-code of this technique.
Complexity Analysis: The algorithm only contains some if-else statements and some comparisons, and there are only five stored values. Hence, both the speed and space complexity is constant, i.e., $O(1)$.
The runtimes of this technique can be seen in Figure 8.6.

```
Data: delay
Data: nbr_of_values
Data: metric_value
Data: earliest_data
Data: latest_data
input: new_data
nbr_of_values += 1
if earliest_data is uninitialized then
    earliest_data = new_data.timestamp
    earliest_data = new_data.timestamp
    metric_value = 1
else
    if earliest_data > new_data.timestamp then
        earliest_data = new_data.timestamp
    end
    if latest_data < new_data.timestamp then
        latest_data = new_data.timestamp
    end
    expected_nbr_of_values = ((latest_data - earliest_data) / delay) + 1
    metric_value = nbr_of_values / expected_nbr_of_values
end
```

Algorithm 5: Pseudo-code of the completeness technique.

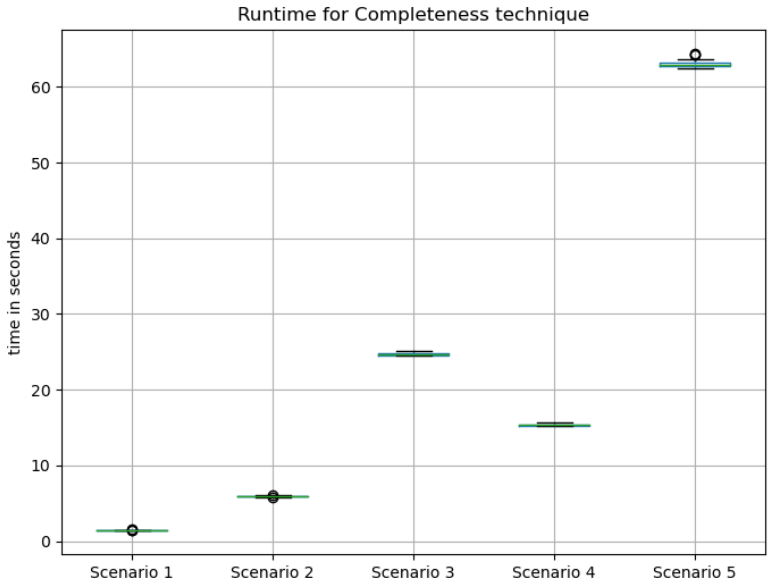


Figure 8.6: Runtimes for the completeness technique in the various scenarios.

8.2.3 Timeliness Techniques

Two timeliness techniques were implemented, both of which use the cardinality category, where one of them counts the number of a specific occurrence (as a ratio), and the other computes the mean. The techniques are based on the definition of timeliness given in Section 3.1.2.4, i.e., computing the difference between a new data point's timestamp and the current timestamp of the system.

8.2.3.1 Average Timeliness

The average timeliness is computed by averaging the timeliness of each data point, see Algorithm 6 for pseudo-code.

Data: metric_value
Data: number_of_values
input: new_data
current_time = time.now()
delta = current_time - new_data.date
number_of_values += 1
metric_value *= (number_of_values - 1) / number_of_values
metric_value += delta / number_of_values

Algorithm 6: Pseudo-code of the average timeliness technique.

Complexity Analysis: Both the speed and space complexity is constant, i.e., $O(1)$.
The runtimes of this technique can be seen in Figure 8.7.

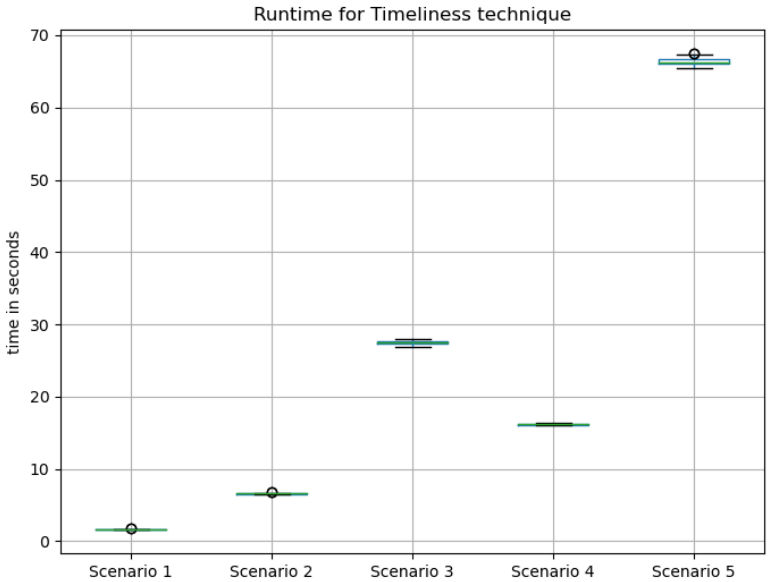


Figure 8.7: Runtimes for the average timeliness technique in the various scenarios.

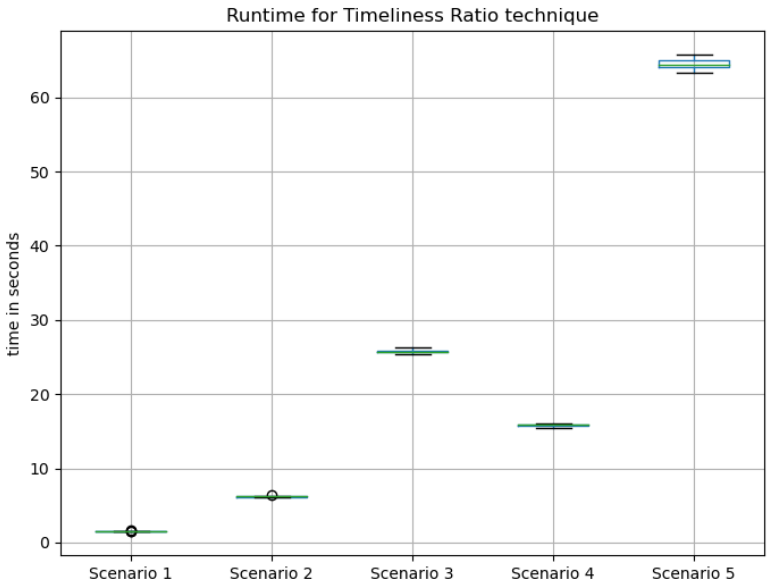


Figure 8.8: Runtimes for the timeliness ratio technique in the various scenarios.

8.2.3.2 Timeliness Ratio

The timeliness ratio technique is similar to the average timeliness technique, with the difference of that it inputs a threshold for how high the timeliness of a data point is allowed to be. This technique presents its metric value as the ratio of the number of data points with a timeliness value below the given threshold to the total number of data points. The benefit of this over the average timeliness technique is that it provides a normalized value. See Algorithm 7 for pseudo-code.

Data: metric_value
Data: number_of_valid
Data: number_of_invalid
Data: threshold
input: new_data
current_time = time.now()
delta = current_time - new_data.date
if *delta* > *threshold* **then**
 | number_of_invalid += 1
else
 | number_of_valid += 1
end
metric_value = number_of_valid / (number_of_valid + number_of_invalid)

Algorithm 7: Pseudo-code of the timeliness ratio technique.

Complexity Analysis: Both the speed and space complexity is constant, i.e., $O(1)$.
The runtimes of this technique can be seen in Figure 8.8.

8.2.4 Consistency Techniques

Consistency is defined in this thesis as the consistency of a data quality dimension within a dataset, see Section 6.3.2.1. Thus, consistency could be applied to any of the other selected data quality dimensions. However, developing a consistency technique for every other data quality dimension would be time-consuming and redundant. Therefore, only two consistency techniques were developed, completeness consistency and accuracy consistency. Both of these techniques follows the definition given in Section 6.3.2.1 and are based on the value distribution data profiling category, specifically, computing the constancy of a data quality dimension (i.e., of completeness or accuracy).

8.2.4.1 Completeness Consistency

Completeness consistency shows to what degree data points in a dataset agree with each other regarding the completeness. The technique for this was based on aggregating multiple data points and see how well the data point aggregations agree on a completeness metric. Aggregations were necessary since if the consistency were to be determined on single data points, it would mimic the completeness metric since each data point has a binary completeness metric, either present (i.e., 1) or missing (i.e., 0). The implementation of the completeness consistency takes in aggregation size as input and then aggregates data points that are close to each other in time to an aggregation. Each aggregation has a completeness metric. The completeness consistency metric is then computed by computing the constancy of these aggregations, i.e., the percentage of aggregations that agree on the most popular completeness metric. See Algorithm 8 for pseudo-code.

Complexity Analysis: Many of the operations in the implementation are hash map operations, which are expected to be performed in constant time. However, worst-case speed will be linear in the number of aggregations, due to the operations on the *aggregation_counter* hash map. If there exist missing aggregations between the current *match_date* and *latest_date*, these missing aggregations are added through the while-loop, and this takes linear time in the number of missing aggregations. Missing aggregations should be rare as it requires at least *aggregation_size* many sequentially missing data points. Nonetheless, it adds to the time complexity, indicated as *cma* (current missing aggregations), which is defined as $cma = \frac{key\_date - latest\_date}{delay \cdot aggragation\_size}$. When calculating the metric value, a linear scan is done on the frequency hash map to find the most frequent metric, which is done in linear time to the number of possible metrics. The number of available metrics is $aggregation\_size + 1$, since if the aggregation size is 3, there are three points assigned to every aggregation, each of which is either present or absent, the completeness of such aggregation is either: 0, $\frac{1}{3}$, $\frac{2}{3}$, or 1, i.e., $aggregation\_size + 1$ different values. The space complexity is determined by the sizes of the two hash maps, the first hash map has the size of the number of aggregations, and the second hash map has the size of possible metrics, i.e., $aggregation\_size+1$. The number of aggregations is bounded by the number of data points n. Hence the time complexity of this implementation is $O(n+aggregation\_size)$ and $\Theta(aggregation\_size+cma)$, and the space complexity is $O(n + aggregation\_size)$.

```
Data: metric_value
Data: delay
Data: aggregation_size
Data: aggregation_counter (Hashmap: timestamp → integer)
Data: number_of_aggregates
Data: start_date
Data: latest_date
Data: frequency_map (Hashmap: completeness metric → frequency)
input: new_data
if start_date is uninitialized then
    start_date = new_data.date
    latest_date = new_data.date
end
key_date = new_data.date - ((new_data - start_date) % (delay *
 aggregation_size)
if aggregation_counter.contains(key_date) then
    prev_completeness = aggregation_counter.get(key_date) /
     aggregation_size
    frequency_map.decrement(prev_completeness)
    aggregation_counter.increment(key_date)
else
    prev_date = key_date - (delay * aggregation_size)
    // Make sure aggregations where all data points are missing is added
    while prev_date > latest_date do
        number_of_aggregates += 1
        aggregation_counter.add(prev_date, 0)
        frequency_map.increment(0)
        prev_date -= delay * aggregation_size
    end
    number_of_aggregates += 1
    aggregation_counter.add(key_date, 1)
    latest_date = match_date
end
completeness_metric = aggregation_counter.get(key_date) / aggregation_size
 if frequency_map.contains(completeness_metric) then
    frequency_map.increment(completeness_metric)
else
    frequency_map.add(completeness_metric, 1)
end
metric_value = max(frequency_map.values()) / number_of_aggregates
// When an aggregated is full, its metric can not change anymore
// Thus, it can be removed to save space
if aggregation_counter.get(key_date).size() == aggregation_size then
    aggregation_counter.remove(key_date)
end
```

Algorithm 8: Pseudo-code of the completeness consistency technique.

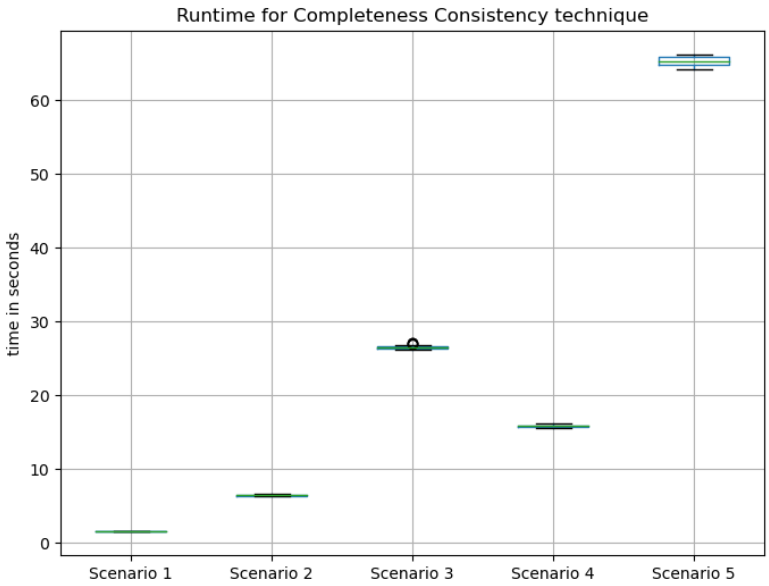


Figure 8.9: Runtimes for the completeness consistency technique in the various scenarios.

The runtimes of the completeness consistency technique can be seen in Figure 8.9. This technique was evaluated with the aggregation size set to 10.

8.2.4.2 Accuracy Consistency

The accuracy consistency implementation computes the biggest difference between two values in data points (i.e., the accuracy) and puts it in a sorted list. Then it selects the median difference (delta) from the sorted list. Given an input "threshold", the technique computes how many (as a ratio) of the data points has an accuracy that does not deviate from the median accuracy more than the threshold value, i.e., the allowed interval is $[median\_value - threshold, median\_value + threshold]$. This technique will thus show how consistent the difference/accuracy is between the data points. See Algorithm 9 for pseudo-code.
Complexity Analysis: The time complexity of this implementation is decided by the sorted list operations. To add a new value to a sorted list takes $O(log(n))$ time, to find the median takes constant time, and the time to find the number of values within a given value interval is logarithmic in n, since it takes logarithmic time for look-ups in a sorted list. Furthermore, the max and min operations of the incoming values are linear in s (number of sensors/sources), due to a linear scan needs to be done. All data points are stored in the sorted list, the required space for this is n. Hence, the time complexity is $O(log(n) + s)$ and the space complexity is $O(n)$.
The runtimes of this technique can be seen in Figure 8.10.

Data: metric_value
Data: threshold
Data: sorted_accuracy_list
input: new_data
max_val = max(new_data.values)
min_val = min(new_data.values)
delta = max_val - min_val
sorted_accuracy_list.sorted_add(delta)
median = sorted_accuracy_list.find_median()
lower = median - threshold
upper = median + threshold
number_of_valid = sorted_accuracy_list.number_of_values_between(lower, upper)
metric.value = number_of_valid / sorted_accuracy_list.size()

Algorithm 9: Pseudo-code of the accuracy consistency technique.

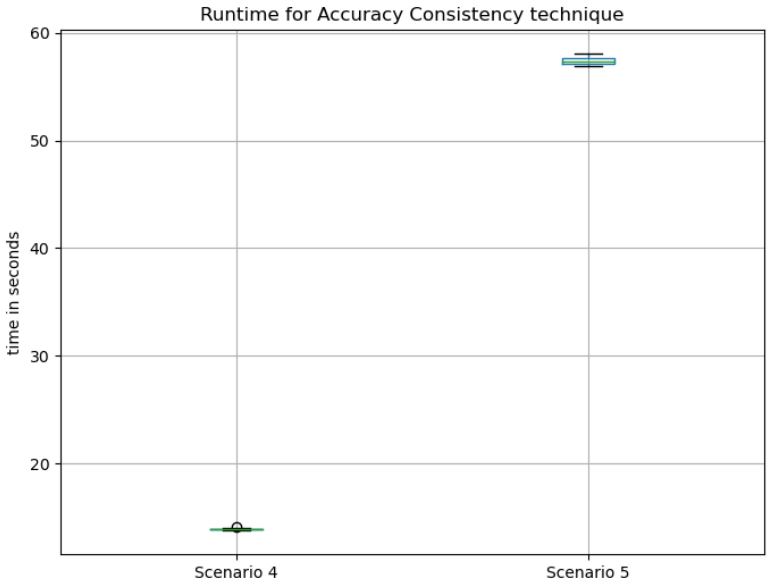


Figure 8.10: Runtimes for the accuracy consistency technique in the various scenarios. Only Scenario 4 and 5 was used since this technique requires at least two sensor sources.

8.2.5 Currency Techniques

Two currency techniques were implemented, both of which technically did not use any established data profiling methods since the currency relies on how frequent new data points are received. The currency implementations are based on the definition of currency given in Section 3.1.2.5. Since currency is computed by taking the difference between the time the last data point was received and the current time,

the currency techniques need to be updated continuously even though no new data is available. Otherwise, the currency would always be perfect (i.e., 0).

8.2.5.1 Basic Currency

The basic currency technique shows a current currency of the sensor aggregate, i.e., how recent it was updated. It is done by subtracting the current time with the time of when the technique received its most recent data point. See Algorithm 10 for pseudo-code.

Data: metric_value
Data: latest_update
input: new_data
if *new_data not Empty* **then**
| latest_update = time.now()
end
current_time = time.now()
metric_value = current_value - latest_update

Algorithm 10: Pseudo-code of the currency technique.

Complexity Analysis: Both the speed and space complexity is constant, i.e., $O(1)$.
The runtimes of this technique can be seen in Figure 8.11.

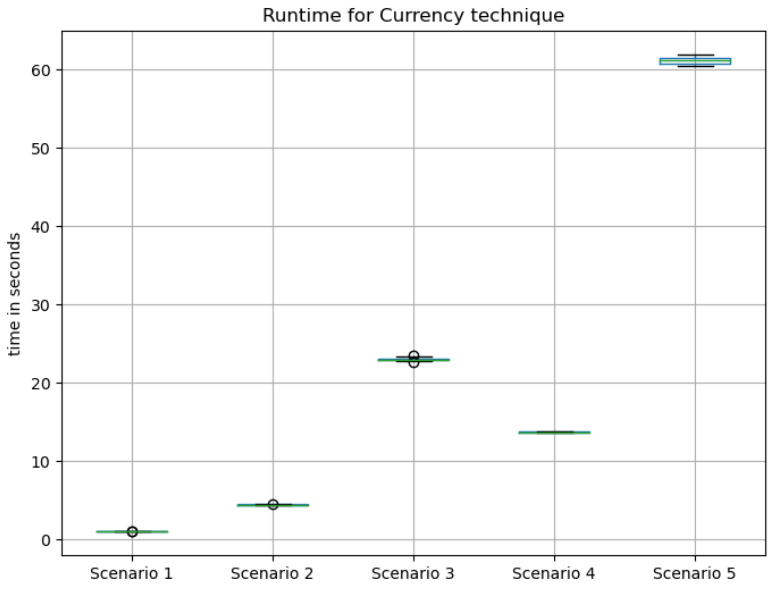


Figure 8.11: Runtimes for the basic currency technique in the various scenarios.

8.2.5.2 Currency Ratio

The currency ratio technique is similar to the basic currency technique, with the difference of that it inputs an allowed maximum threshold for how high the cur-

rency is allowed to be. The technique presents its metric value as the ratio of the number of times/occasions that is below the given threshold to the total number of times/occasions. The benefit of this over the basic currency is that it provides a normalized value. See Algorithm 11 for pseudo-code.

```
Data: metric_value
Data: latest_update
Data: threshold
Data: number_of_valid
Data: number_of_invalid
input: new_data
if new_data not Empty then
 |  latest_update = time.now()
end
current_time = time.now()
delta = latest_update - current_time
if delta > threshold then
 |  number_of_invalid += 1
else
 |  number_of_valid += 1
end
metric_value = number_of_valid / (number_of_valid + number_of_invalid)
```

Algorithm 11: Pseudo-code of the currency ratio technique.

Complexity Analysis: Both the speed and space complexity is constant, i.e., $O(1)$.

The runtimes of this technique can be seen in Figure 8.12.

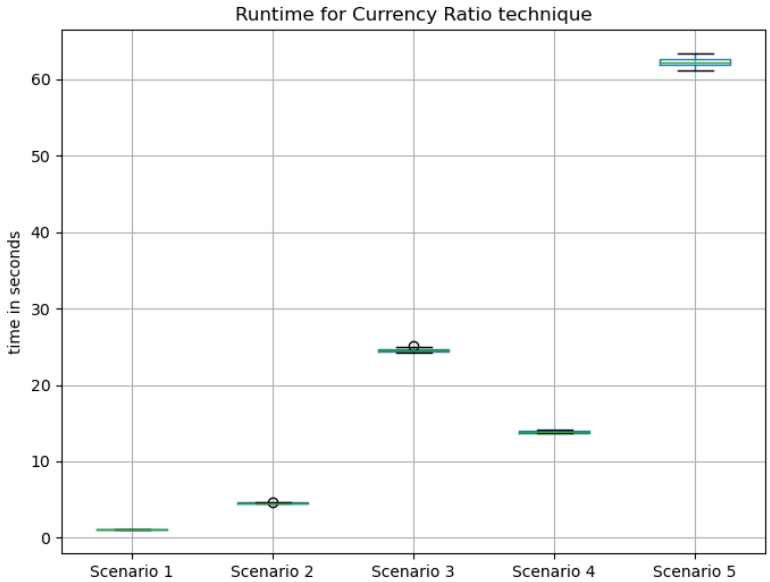


Figure 8.12: Runtimes for the currency ratio technique in the various scenarios.

8.2.6 Duplicate Techniques

Only one duplicate assessment technique was implemented. The duplicate technique computes the ratio of the number of unique data points to the total number of data points, i.e., as explained in Section 3.1.2.14. The technique was based on the data profiling category cardinality by counting the occurrences of duplicates. It was implemented by storing timestamps in a hash set, whenever a new data point was added, look if the hash set already contains the timestamp of that data point. To not run out of space, old timestamps are removed. See Algorithm 12 for the pseudo-code.

```
Data: metric_value
Data: number_of_values
Data: number_of_duplicates
Data: timestamp_set
Data: queue
Data: max_size_set
input: new_data
if timestamp_set.contains(new_data.date) then
    number_of_duplicates += 1
    queue.remove(new_data.date)
else
    timestamp_set.add(new_data.date)
end
queue.append(new_data.date)
if queue.size() > max_size_set then
    timestamp_set.remove(queue.pop())
end
number_of_values += 1
metric_value = (number_of_values - nbr_of_duplicates) / number_of_values
```

Algorithm 12: Pseudo-code of the duplicate assessment technique.

Complexity Analysis: To see if a hash set contains a specific value or to remove a value from a hash set is expected to be done in constant time. However, the worst-case scenario is linear in the size of the hash set. Furthermore, to remove a specific value from a queue is done in linear time. However, the removal of specific objects (i.e., dates) is only done whenever a duplicate is found, which usually is rare. Furthermore, the step of removing a specific object from the queue could be removed, since that removal is not necessary. This implementation requires saving max_size_set number of dates in both a hash set and queue, i.e., linear in that variable. Hence, the time complexity of this implementation is $O(max\_size\_set)$ and $\Theta(1)$, and the space complexity is $O(max\_size\_set)$.
The runtimes of this technique can be seen in Figure 8.13.

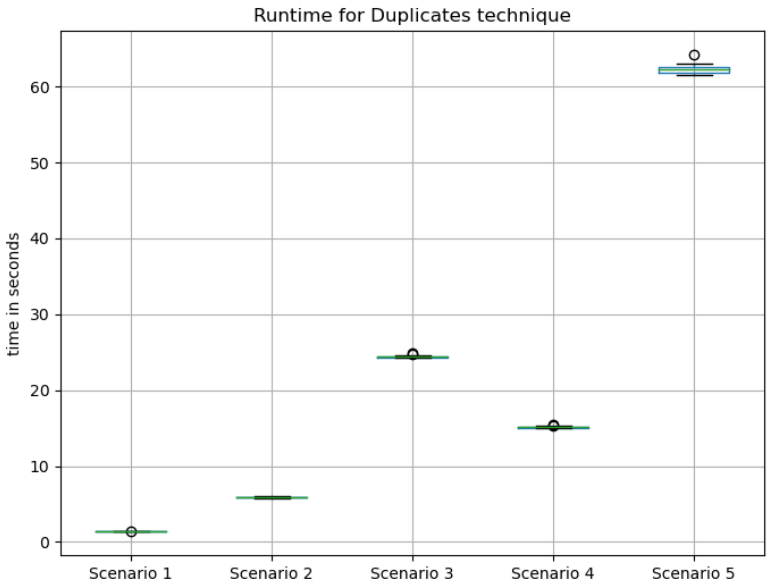


Figure 8.13: Runtimes for the duplicate assessment technique in the various scenarios.

8.2.7 Precision Techniques

Two precision techniques were developed, both of which are computed by calculating the standard deviation of sensor data, as explained in Section 3.1.2.15. Both techniques computed standard deviation through the Python package *Numpy*. Both techniques use a data profiling cardinality approach, as they both compute the mean of something (i.e., mean standard deviation).

8.2.7.1 Basic Precision (Average precision)

The basic precision computes the average standard deviation of the data points in a sensor aggregate. It requires data points with two or more numeric values. See Algorithm 13 for pseudo-code of this technique.

Data: metric_value
Data: number_of_values
input: new_data
std_dev = numpy.std(new_date.values)
number_of_values += 1
metric_value *= (number_of_values - 1) / number_of_values
metric_value += std_dev / number_of_values

Algorithm 13: Pseudo-code of the basic precision technique.

Complexity Analysis: The speed of this technique is decided by the computation of standard deviation, which is linear in the number of values (i.e., the number of sensors/sources). Two variables are stored, thus the space complexity is constant. Hence, the time complexity is $O(s)$ and the space complexity is $O(1)$.

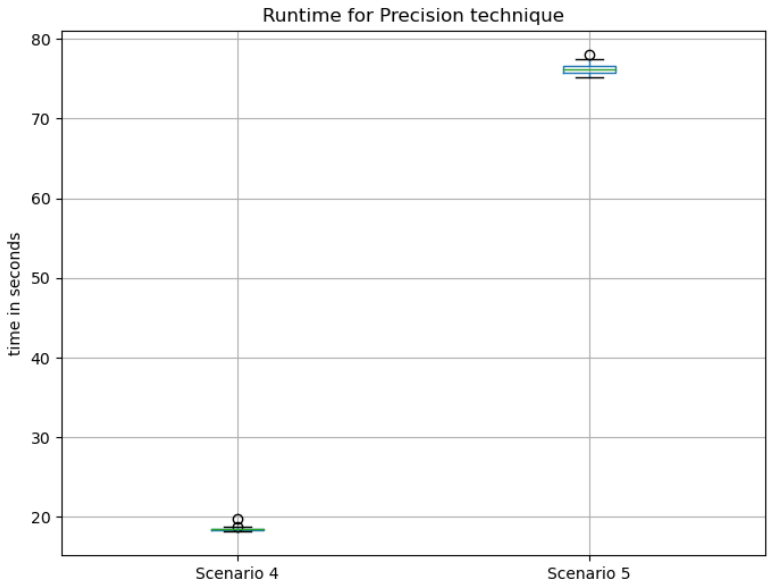


Figure 8.14: Runtimes for the basic precision technique in the various scenarios. Only Scenario 4 and 5 was used since this technique requires at least two sensor sources.

The runtimes of this technique can be seen in Figure 8.14.

8.2.7.2 Aggregated Precision

The aggregated precision implementation is similar to the basic precision, with the difference that it calculates the standard deviation on values from multiple data points that are close to each other in time. The aggregation enables calculating the precision metric when only one sensor/source is present. The size of the aggregations is decided by an integer input. See Algorithm 14 for pseudo-code of this technique.
Complexity Analysis: The speed of this implementation is decided by the computation of standard deviation, which is linear in the number of values in the current aggregation, which is somewhere between $[s, s \cdot aggregation\_size]$. The number of values in aggregations is thus bounded by $s \cdot aggregation\_size$. Additionally, the expected speed of the operations on the hash map is constant, however, worst-case is linear in the number aggregations. The number of aggregations is bounded by the number of data points n. Furthermore, all data points are stored in the hash map until its aggregation is filled, however, aggregation might never be filled due to missing data points. Thus the space required is at worst linear in the number of received data points. Hence, the time complexity is $O(n + s \cdot aggregation\_size)$ $\Omega(s \cdot aggregation\_size)$ and the space complexity is $O(n)$.
The runtimes of this technique can be seen in Figure 8.15. This technique was evaluated with the aggregation size set to 10.

```
Data: metric_value
Data: number_of_aggregates
Data: aggregation_size
Data: delay
Data: start_date
Data: value_map (Hash map: timestamp → list of values)
input: new_data
if start_date is uninitialized then
  | start_date = new_data.date
end
key_date = new_data - ((new_data - start_date) % (delay *
 aggregation_size)
if value_map.contains(key_date) then
  | previous_std = numpy.std(value_map.get(key_date))
  | metric_value -= previous_std / number_of_aggregates
  | value_map.get(key_date).addAll(new_data.values)
  | std = numpy.std(value_map.get(key_date))
  | metric_value += std / number_of_aggregates
else
  | value_map.add(key_date, new_data.values)
  | number_of_aggregates += 1
  | std = numpy.std(value_map.get(key_date))
  | metric_value *= (number_of_aggregates - 1) / number_of_aggregates
  | metric_value += std / number_of_aggregates
end
// Remove the aggregation if it has received all its values
if value_map.get(key_date).size() ≥ aggregation_size * new_data.values.size()
 then
  | value_map.remove(key_date)
end
```

Algorithm 14: Pseudo-code of the aggregated precision technique.

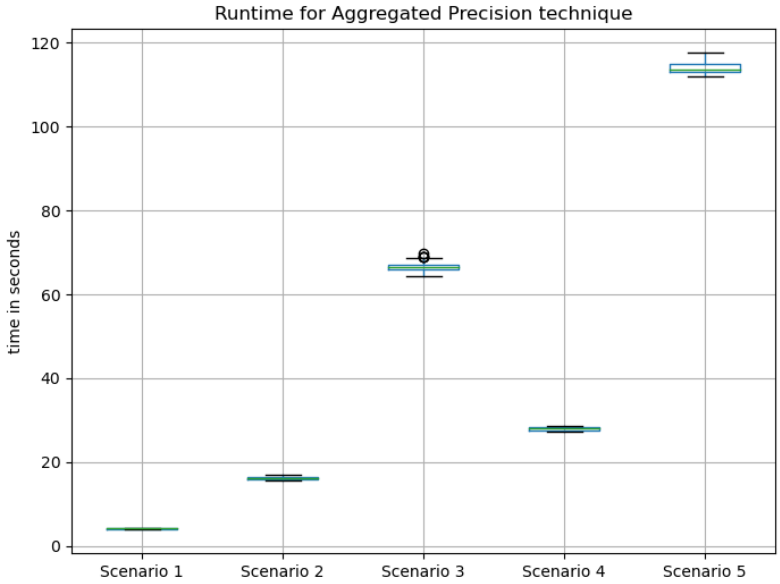


Figure 8.15: Runtimes for the aggregated precision technique in the various scenarios.

8.3 Comparison of the Quality Assessment Techniques

This section contains a figure for each of the five scenarios with the running times of all the techniques. Furthermore, two tables are presented at the end of this section, the first containing the median time of each technique in every scenario and the other containing the time and space complexities of the techniques. The purpose of this section is to aggregate the results in an easy-to-read manner since Section 8.2 is relatively long.
The objective of the runs was both to find out the various techniques' actual speeds and to see if the execution of the techniques were faster than the speed of which the data theoretically would be generated in a real-world scenario. As explained in Section 8.1, for a technique to be feasible, it must be able to process a single data point in under 0.1 seconds. Thus, the threshold which the techniques would have to be under was 1.600 seconds (or 400 seconds for Scenario 1) to satisfy the requirement of processing a single data point in under 0.1 seconds. However, this approach does not measure the time of processing each individual data point, the time to process a single data point is inferred by the median time it takes to process a single data point. As seen on Figures 8.16, 8.17, 8.18, 8.19, and 8.20, all techniques are under this threshold. Most of the techniques are also fairly similar in their runtimes, however, the precision techniques are deviating from the rest as they have significantly higher runtimes than the other techniques. Further analysis of these results is given in Section 10.1.2.

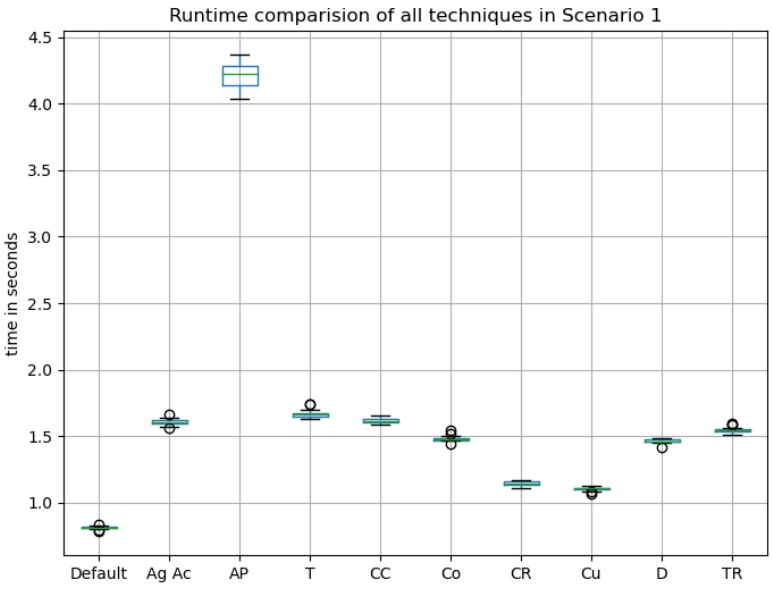


Figure 8.16: The runtime of the various techniques in Scenario 1.

Default = Framework running without any techniques, Ag Ac = Aggregated Accuracy, AP = Aggregated Precision, T = Basic Timeliness, CC = Completeness Consistency, Co = Completeness, CR = Currency Ratio, Cu = Basic Currency, D = Duplicates, TR = Timeliness Ratio

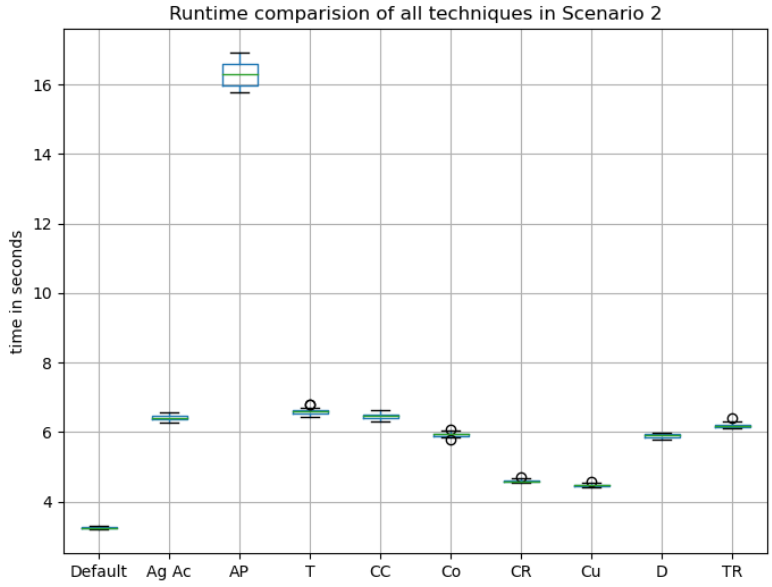


Figure 8.17: The runtime of the various techniques in Scenario 2.

Default = Framework running without any techniques, Ag Ac = Aggregated Accuracy, AP = Aggregated Precision, T = Basic Timeliness, CC = Completeness Consistency, Co = Completeness, CR = Currency Ratio, Cu = Basic Currency, D = Duplicates, TR = Timeliness Ratio

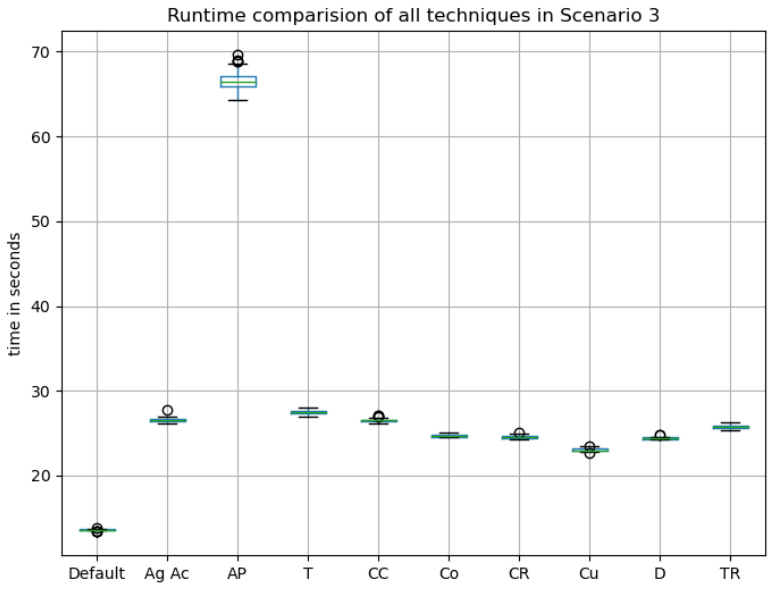


Figure 8.18: The runtime of the various techniques in Scenario 3.
Default = Framework running without any techniques, Ag Ac = Aggregated Accuracy, AP = Aggregated Precision, T = Basic Timeliness, CC = Completeness Consistency, Co = Completeness, CR = Currency Ratio, Cu = Basic Currency, D = Duplicates, TR = Timeliness Ratio

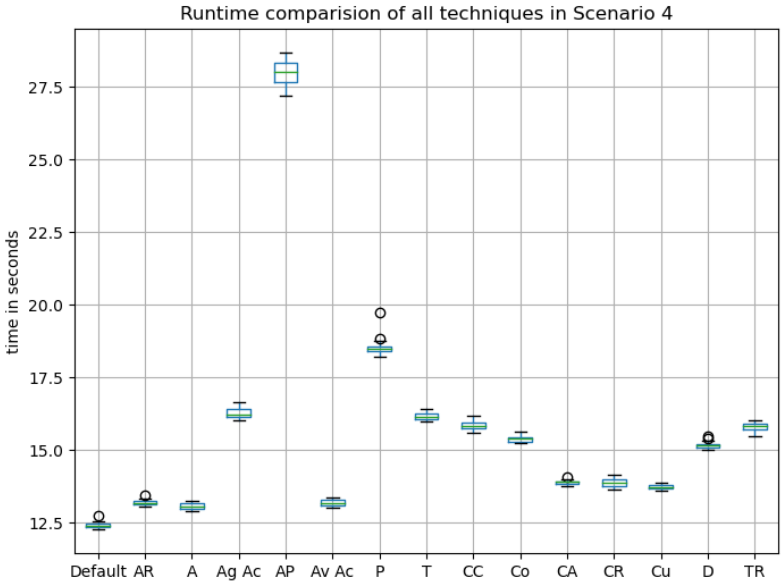


Figure 8.19: The runtime of the various techniques in Scenario 4.
Default = Framework running without any techniques, AR = Accuracy Ratio, A = Basic Accuracy, Ag Ac = Aggregated Accuracy, AP = Aggregated Precision, Av Ac = Average Accuracy, P = Basic Precision, T = Basic Timeliness, CC = Completeness Consistency, Co = Completeness, CA = Consistency Accuracy, CR = Currency Ratio, Cu = Basic Currency, D = Duplicates, TR = Timeliness Ratio

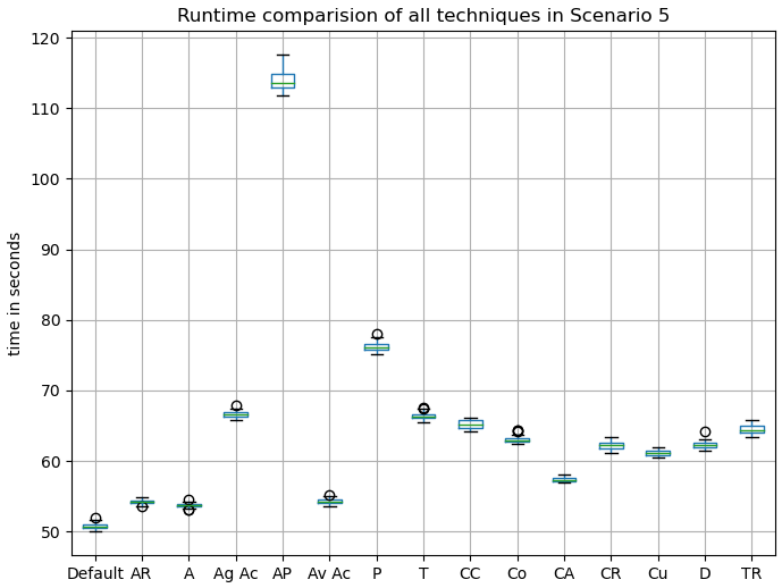


Figure 8.20: The runtime of the various techniques in Scenario 5.
Default = Framework running without any techniques, AR = Accuracy Ratio, A = Basic Accuracy, Ag Ac = Aggregated Accuracy, AP = Aggregated Precision, Av Ac = Average Accuracy, P = Basic Precision, T = Basic Timeliness, CC = Completeness Consistency, Co = Completeness, CA = Consistency Accuracy, CR = Currency Ratio, Cu = Basic Currency, D = Duplicates, TR = Timeliness Ratio

	1	2	3	4	5
Default	0.81	3.25	13.62	12.40	50.63
Accuracy	-	-	-	13.08	53.73
Accuracy Ratio	-	-	-	13.19	54.17
Average Accuracy	-	-	-	13.20	54.27
Aggregated Accuracy	1.61	6.42	26.61	16.22	66.58
Precision	-	-	-	18.48	76.16
Aggregated Precision	4.23	16.31	66.41	28.00	113.52
Timeliness	1.66	6.58	27.49	16.15	66.28
Timeliness Ratio	1.54	6.19	25.72	15.82	64.35
Completeness	1.48	5.93	24.66	15.39	62.96
Completeness Consistency	1.61	6.46	26.50	15.82	65.18
Accuracy Consistency	-	-	-	13.90	57.34
Currency	1.11	4.46	22.99	13.73	61.21
Currency Ratio	1.15	4.59	24.53	13.87	62.20
Duplicates	1.46	5.90	24.42	15.15	62.25

Table 8.1: Median runtimes of the techniques in the various scenarios.

	Time $O(.)$	Time $\Theta(.)$	Space $O(.)$	Space $\Theta(.)$
Accuracy	s	s	1	1
Accuracy Ratio	s	s	1	1
Average Accuracy	s	s	1	1
Aggregated Accuracy	n+s	s	n	n/as
Precision	s	s	1	1
Aggregated Precision	n + s · as	s · as	n	n
Timeliness	1	1	1	1
Timeliness Ratio	1	1	1	1
Completeness	1	1	1	1
Completeness Consistency	n + as	as + cma	n + as	n + as
Accuracy Consistency	log(n) + s	log(n) + s	n	n
Currency	1	1	1	1
Currency Ratio	1	1	1	1
Duplicates	ss	1	ss	ss

Table 8.2: Time and space complexities of the techniques.
s = number of sources, n = number of data points received thus far, as = aggregation size, cma = number of currently missing aggregations, ss = size of set.

8.4 Results of Data Profiling for the Evaluation Scenarios

While calculating the speed of the techniques is important in the context of this thesis, the purpose of these techniques is to ultimately assess the quality of data. In the given scenarios in Section 8.1, the techniques calculated the quality metrics seen in Table 8.3. The right-most columns are empty on Scenario 1, 2, and 3 because those scenarios only have one sensor source and the corresponding techniques require two or more sensor sources. The time-based dimensions are not included because they are dependant on time. Currency is not in the table because during the evaluations it was not possible to replicate and measure the true currency of the dataset since it would require knowledge about the exact time each data point was received by the system which originally collected the dataset. Instead, the currency was computed by the currency of the framework with preloaded data queues. Since all the data were preloaded into the queues and processed without any delays, the currency was always 0 or close to 0. Timeliness was not used since it would only show how old the dataset is, based on the provided timestamps. Timeliness is usually used to know how long time it takes for data to be sent from the sensor to its destination.
The reason why the intrinsic metrics (accuracy and precision) are very similar in the Scenario 4 and 5 rows is that while those scenarios use four different sensor sources, it is, in reality, only one dataset that has been duplicated three times but with added noise. The added noise in all scenarios was Gaussian with the same standard deviation. Due to the added noise being the same for all replicated sensor sources, the intrinsic metrics are similar to each other in Scenario 4 and 5.
The following paragraphs describe and explain the metrics of the scenarios.

Completeness: The completeness of all scenarios except for scenario 1, was 0.81. The completeness metrics were identical because the various data points were taken from the same dataset, where every column had the same completeness. The absent values came in chunks, i.e., there were long periods that did not have any data. Since the missing data points came in chunks, in most cases, the adjacent data points of a missing data point were also missing, which makes it difficult to interpolate the value of a missing data point. In such scenarios, there is probably something wrong with the sensors, which this framework would have noticed in a real-world setting. Furthermore, the completeness for scenario 1 was high, only a few data points were missing, and they did not come in chunks. In that scenario, enhancement would have been possible.
Aggregated Accuracy: The aggregated accuracy was performed in aggregations of size ten, i.e. since the sensor frequency was 10 Hz, the data points generated within the same second were aggregated. The aggregated accuracy metrics of the scenarios without Gaussian noise added to them (i.e., Scenario 1, 2, and 3) was quite good. For instance, the aggregated accuracy for the velocity was roughly $2m/s$, and this value indicates that there is probably no problem with the sensors. What this essentially means is that the sensor values from data points within the same second differs at most $2m/s$, this is plausible since the acceleration and deceleration (by braking) of $2m/s^2$ is possible in a car. The aggregated accuracy metrics of pitch, roll, and yaw-rate are lower (better) than the aggregated accuracy metric of velocity. These metrics were lower because the values of these columns in the dataset were generally much lower than the velocity values. Thus, they are expected to deviate less. Scenario 4 and 5 have significantly higher (worse) aggregated accuracy, and also almost identical metrics to each other. The reason for the metric being similar is because of the added Gaussian noise, as explained above.
Aggregated Precision: The aggregated precision metrics are similar to the aggregated accuracy metrics, by that, they are low in Scenario 1, 2, and 3, and higher for scenario 4 and 5. The reason for this is the same as explained above in the aggregated accuracy paragraph, i.e., due to the added Gaussian noise.
Duplicates: The dataset did not contain any duplicates at all, and thus has the perfect duplicate metric value, i.e., 1. The sensors used to create the datasets which the evaluation scenarios were based upon were probably designed such that duplicates will never occur. Thus, the duplicate dimension would probably be unnecessary for those sensors, since those sensors will always have a perfect duplicate metric.
Completeness Consistency: The completeness consistency was 0.98 for Scenario 1, and 0.80 for all the other scenarios. These metrics are similar to the completeness metrics. The reason for the completeness consistency almost mirroring the completeness metrics is because the missing data point came in large chunks, as explained above (in the completeness paragraph). Due to this, most aggregations will have a completeness of either 1 or 0, since most aggregations will contain either data points exclusively from the chunks of missing data points, and have a completeness of 0, or contain only present data points and have a completeness of 1. Completeness consistency would probably be more suitable in situations where these "chunks" of data is present, but these chunks simply have a slightly lower completeness metric, and not 0. In these cases, the basic completeness metric will not change that much,

but the completeness consistency will change noticeably.
Accuracy: The accuracy metrics are based on the added Gaussian noise in Scenario 4 and 5. As expected, the accuracy metrics are similar but slightly lower than the aggregated accuracy metrics, since they are calculated similarly, but the aggregated accuracy uses multiple data points and thus giving slightly higher metrics.
Average Accuracy: The average accuracy metrics are very similar to each other, which is because of the added Gaussian noise.
Accuracy Ratio: The accuracy ratio was computed by allowing an accuracy of at most 1, i.e., the biggest allowed difference among the sensor sources is 1. Since the average accuracy is around 1.77, substantially higher than 1, the accuracy ratio metrics are low (bad).
Precision: As with the other intrinsic data quality metrics, the precision metrics are almost identical, due to the Gaussian noise.
Accuracy Consistency: The accuracy consistency was computed by allowing the accuracy values of the various data points to deviate at most by 2 from the median. This allowed deviation created a big range of allowed accuracy values, and since the accuracy values are Gaussian distributed (due to the Gaussian noise), the accuracy values should not deviate that much. Hence, the accuracy consistency metrics are quite high, almost perfect.

	C	Ag Ac	AP	D	CC	Acc	AA	AR	P	AC
S1 Vel	0.99	1.97	0.074	1	0.98	-	-	-	-	
S2 Vel	0.81	1.97	0.048	1	0.80	-	-	-	-	
S3 Vel	0.81	1.97	0.048	1	0.80	-	-	-	-	
S3 Pitch	0.81	0.078	0.0013	1	0.80	-	-	-	-	
S3 Roll	0.81	0.052	0.0018	1	0.80	-	-	-	-	
S3 Yaw	0.81	0.44	0.0042	1	0.80	-	-	-	-	
S4 Vel	0.81	6.97	0.85	1	0.80	6.46	1.77	0.17	0.68	0.97
S5 Vel	0.81	6.97	0.85	1	0.80	6.46	1.77	0.17	0.68	0.97
S5 Pitch	0.81	6.86	0.84	1	0.80	6.11	1.77	0.17	0.68	0.97
S5 Roll	0.81	7.13	0.85	1	0.80	5.95	1.78	0.17	0.68	0.97
S5 Yaw	0.81	6.72	0.84	1	0.80	6.16	1.76	0.18	0.67	0.97

Table 8.3: A table containing the data quality metrics (except the time-related metrics) of the evaluation scenarios given in Section 8.1. All values are rounded down to the closest value with two decimal places.
C = Completeness, Ag Ac = Aggregated Accuracy with aggregation size 10, AP = Aggregated Precision with aggregation size 10, D = Duplicates, CC = Completeness Consistency with aggregation size 10, Acc = (Basic) Accuracy, AA = Average Accuracy, AR = Accuracy Ratio with threshold 1, P = Precision, AC = Accuracy Consistency with allowed deviation 2 from median.

9

Exploration of Data Enhancement Techniques for Sensor Data

This chapter aims to answer the third research question:

- Could sensor data of bad quality be efficiently enhanced?

The presented techniques in this chapter should theoretically be able to improve the quality of sensor data concerning at least one data quality dimension presented in Chapter 6. The techniques presented in this chapter were selected from researching enhancement techniques. Additionally, two enhancement techniques were implemented as a proof of concept, their pseudo-code, time and space complexity, and runtimes on datasets are presented in this chapter.
This chapter is divided into two sections, the first section presents the various enhancement techniques (including the proof of concepts implementations). The second section presents the evaluations of the proof of concept implementations, which is similar to the evaluations in Chapter 8, i.e., complexity analysis and runtimes.

9.1 Data Enhancement Techniques

In this section, the various investigated data enhancement techniques are presented.

9.1.1 Data Deduplication

Data deduplication is a technique where duplicates of data are removed and thus improving the duplicate data quality dimension [25]. The approach of how this is solved consists of two steps, finding duplicates and removing the found duplicates. In this thesis, a simple algorithm for data deduplication was implemented and tested as a proof of concept.
To detect duplicates, it is beneficial to know if the data points available have any unique identifiers, to enable duplicate detection. For example, if a temperature sensor sends two data points with value 20, it does not necessarily mean those two data points are duplicates since values (in this case temperature) given by a sensor can be identical, e.g., it could be 20 degrees (according to a sensor) multiple times. However, in some instances, the data points contain a timestamp of when the data was recorded, since a given time is only happening once, timestamps can be seen as unique identifiers. So whenever a timestamp appears more than once, it is probably due to duplicates.
Since the context of this thesis is the management of newly generated data at the

edge, the proposed algorithm is assumed to be applied in such a context. Therefore, duplicates will technically not be removed, instead, they will get filtered out (or ignored), since the algorithm is supposed to be intermediate to the source of the data and the final destination of data.
Proposed deduplication algorithm, see Algorithm 15.

```
Data: timestamp_hashset
Data: memory_queue
Data: max_queue_size
Data: new_data_point
if timestamp_hashset.contains(new_data_point.timestamp) then
    memory_queue.remove(new_data_point.timestamp)
else
    timestamp_hashset.add(new_data_point.timestamp)
    write_data(new_data_point)
end
memory_queue.add(new_data_point.timestamp)
if memory_queue.size() > max_queue_size then
    old_timestamp = memory_queue.pop()
    timestamp_hashset.remove(old_timestamp)
end
```

Algorithm 15: Pseudo-code for deduplication algorithm.

9.1.2 Interpolation

Interpolation is used whenever a data point or any value is missing. In those cases, the missing data point or value is interpolated based on other data points that are present [25]. Interpolation will improve the completeness of a dataset since the number of data points that are present will increase. A linear interpolation was implemented in this thesis as a proof of concept.
Due to the Markovian property of many phenomena that sensors are measuring, data points that are close in time to the missing point can be used for the interpolation. An example of when this would be reasonable is for temperature sensors, the value of two temperature readings that are close to each other in time should not differ that much. In the case of a temperature sensor that gives one reading each minute and the reading at, for instance, the time 09:10 is missing, but the adjacent readings are available, e.g., the temperature at 09:09 is 10.0 degrees and at 09:11 is 10.6 degrees. While one could never guarantee to predict and interpolate the true temperature, interpolation would give a reasonable estimate. For instance, if linear interpolation were used, the derivative between time 09:09 and 09:11 would be calculated like this:

$$\frac{10.6 - 10 \ degrees}{09{:}11 - 09{:}09} = \frac{0.6 \ degrees}{2 \ minutes} = 0.3 \frac{degrees}{minute}$$

Then, by taking the value of 09:09, i.e. 10, and adding it to the product of multiplying the derivative with the offset in minutes of the missing data point (i.e. 1 minute), and thereby, the temperature value of time 09:10 is interpolated to be: $10 + 0.3 \cdot 1 = 10.3$ degrees.

Pseudo-code for the implemented linear interpolation can be seen in Algorithm 16.

```
Data: prev_data
Data: new_data
Data: expected_delay
if (new_data - prev_data) > expected_delay then
    nbr_of_missing_data = (new_data.date - prev_data.date) /
     expected_delay - 1
    delta_vector = new_data.values - prev_data.values
    for i = 1; i ≤ nbr_of_missing_data; i++ do
        date = prev_data.date + i * delay
        interpolated_values = delta_vector * i + prev_data.values
        new_data_point = create_data_point(date, interpolated_values)
        write_data(new_data_point)
    end
end
write_data(new_data)
prev_data = new_data
```

Algorithm 16: Pseudo-code for interpolation algorithm.

9.1.3 Clustering and Outliers

The multiple column profiling technique clustering and outliers can be used to find outliers potentially caused by faulty data. This section can also be seen as an "outlier detection" section, as described in Section 3.3.1. Whenever an outlier is detected, it could be due to faulty data. If a faulty data point is removed, it would improve accuracy and precision, but lose completeness. To make up for the loss in completeness, interpolation can be used in conjunction with outlier detection. Furthermore, removing outliers would most likely improve the consistency since outliers are, by definition "inconsistent" with the rest of the dataset. However, improvements in consistency rely heavily on what kind of consistency dimension is used (i.e., the underlying data quality dimension it measures).
In this section, multiple clustering and outlier-based techniques will be presented.

9.1.3.1 k-Nearest Neighbor

In k-Nearest Neighbor (kNN), each data point is assigned to one category. The assignment of a data point depends on its k-nearest neighbors. The k neighbors are determined by the distance (e.g., Euclidean distance) between the data point and all the other data points, the k data points with the shortest distances to the data point are selected as the neighbors [67]. A data point will be assigned to the category which most of its k-nearest neighbors are assigned to.
In kNN, there already exists a set of data points assigned to a set of existing and named categories, i.e., it is a supervised algorithm (however, there are also unsupervised kNN approaches [68]). A set of unclassified data points can be classified into one of the established categories based on the already classified data points.
Sheng et al. [69] developed an outlier detection algorithm based on kNN that is aimed towards sensor networks. The idea of the presented algorithm is that an out-

lier is a data point with a particular (long) distance to the other data points in the same category. The researchers had two approaches of how to decide if a data point had too great of a distance (and thus classified as an outlier). The first approach was that a data point was classified as an outlier if the distance between itself and the k:th nearest neighbor (singular) was larger than a set distance threshold d. The second approach was that a data point was classified as an outlier if its distance to its k:th nearest neighbor is one of the largest distances between a data point and its k:th nearest neighbor. The paper presents the performance of algorithms based on these two outlier approaches.

9.1.3.2 k-means

k-means is a clustering algorithm that is similar to the kNN algorithm, with the main difference being that it is an unsupervised algorithm instead of being supervised. Given k clusters and a set of unlabelled data points, the k-means algorithm assigns each data point to one of the k clusters. The initial step is to choose k data points arbitrary, each one of them represents one of the k clusters, then all the other data points are assigned to the cluster with the data point which is closest (e.g., by Euclidean distance). After this step, the algorithm will compute the centroid of each cluster (i.e., the mean value of a cluster), and then assign each data point to the cluster with the closest centroid. The will be done iteratively until the clusters converge (i.e., no data points change cluster) [70].
Souza and Amazonas [70] present a k-means algorithm whose purpose is to be implemented in an IoT architecture. The algorithm proposed first runs the k-means algorithm until it converges, afterward, it computes the distance between each data point and the centroid of its assigned cluster, if this distance is too large, the data point is classified as an outlier.

9.1.3.3 Hierarchical Temporal Memory (HTM)

Hierarchical temporal memory (HTM) is a learning algorithm that learns patterns in real-time. The HTM algorithm is inspired by the human brain, more specifically, the neocortex [71]. In [72], anomaly detection in data streams with HTM is presented. Since HTM can learn patterns in real-time, by inputting data (e.g., sensor data), the algorithm will be able to learn the general structure of how the data looks. The HTM assigns an anomaly score for each pattern (e.g., data) it receives, the score is between 0 and 1. If the HTM algorithm predicts the pattern, it gets the anomaly score 0, however, if it is very deviant from previous patterns (or data), it will get the anomaly score 1.

9.1.4 Data Cleaning

Data cleaning is an enhancement technique based on defining what an error constitutes. Due to this thesis having an agnostic view of the source of sensor data, it is not possible to define a set of properties that a data point should comply with to not be classified as a faulty data point. Due to this, the thesis did not investigate data cleaning further.

However, some data cleaning techniques contain outlier detection, which is presented in Section 9.1.3. Furthermore, deduplication, which is presented in Section 9.1.1, technically is data cleaning, since it follows the three main steps of data cleaning, i.e., defining what errors are, find the errors, and act on the errors. In deduplication, step 1 is that duplicates are errors, step 2 is finding the duplicates, step 3 is to remove the duplicates.

9.1.5 Timeliness and Currency Enhancement

Since both the dimensions timeliness and currency are dependant on time and are decided on whenever the framework receives the data, there is no explicit approach that could improve these dimensions in the context of a framework that receives data. If a time-based data quality metric is low for a system, the approach would be to improve the system or the communication between the sensors and the framework or possibly change sensors.
Due to this, any enhancement on timeliness or currency is out of the scope of this thesis.

9.2 Evaluation of Proof of Concept Implementations

The implemented proof of concept enhancement techniques deduplication and interpolation was evaluated as the data profiling-based techniques, i.e., by running them in the scenarios as described in Section 8.1. Their time and space complexity was also analyzed. Furthermore, the techniques were tested on a dataset containing sensor data of beach water quality [1] to see visualize the enhancements, see Figure 9.3 and 9.4.

9.2.1 Deduplication

The implementation of the deduplication algorithm seen in Algorithm 15 has an expected time complexity of constant time, since checking if an object exists in a hash set is done in constant time. However, the worst-case scenario is linear time in the number of objects in the hash set. To decrease the amount of storage required, there is a max size on the set. Whenever this size is reached, the timestamp that was added earliest in the hash set is removed. To keep track of in what order the timestamps were added, a queue is used. Whenever the maximum size is reached, the queue is popped, and the popped timestamp is then used to remove itself from the hash set. The speed of the removal is the same as checking if an object is in a hash set, i.e., expected time is constant and worst-case time is linear in the size of the hash set.
Hence, the speed complexity and space complexity of this algorithm is:

- Time Complexity: $\Theta(1)$ and $O(max\_queue\_size)$
- Space complexity: $O(max\_queue\_size)$

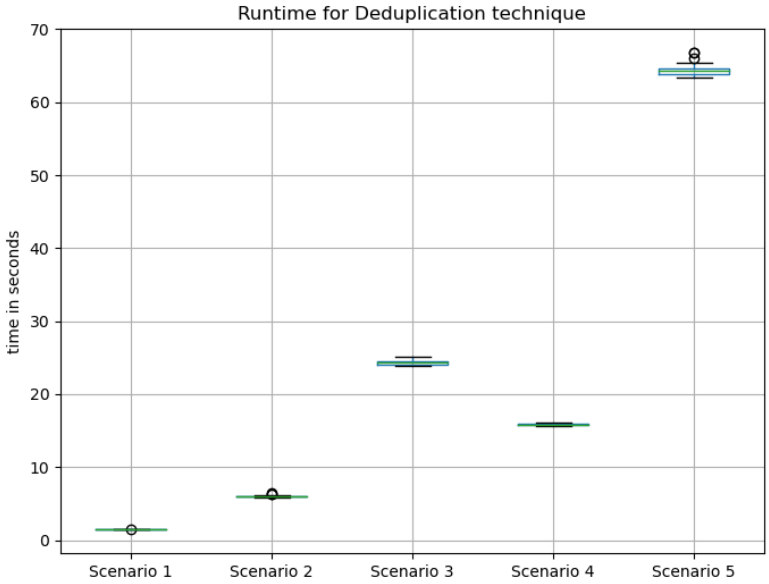


Figure 9.1: Runtimes for deduplication technique in the various scenarios.

This algorithm will be able to handle and deduplicate asynchronous data due to saving old data points. However, if the data is known to be synchronized, i.e., the data points will be received in an order such that the timestamps of the data points are in order, the worst-case speed and space complexity will be constant as well since there is no need to save more than one data point at any given time. The runtimes of the deduplication implementation can be seen in Figure 9.1 and in Table 9.1.

9.2.2 Interpolation

The speed complexity of the interpolation implementation given in Algorithm 16 is linear in the number of missing data points that need to be interpolated, due to the for-loop. If no data points need to be interpolated, the speed complexity is constant. Furthermore, within the interpolation loop, vector subtraction and addition is done, thus, each iteration is linear in the number of values of a data point, i.e., the number of sources, which is denoted as s. Since only one data point needs to be stored at any given time, the space complexity is constant.

- Time complexity: $O(x \cdot s)$, where x is the number of data points needed to be interpolated, s is the number of sensor sources.
- Space complexity: $O(1)$

Note, this algorithm does not support an asynchronous stream of data. However, in the case of handling asynchronous data, one could modify this algorithm, so whenever a received data point has already been interpolated, one could simply override the interpolated value by the newly received value. The runtimes of the interpolation implementation can be seen in Figure 9.2 and in Table 9.1.

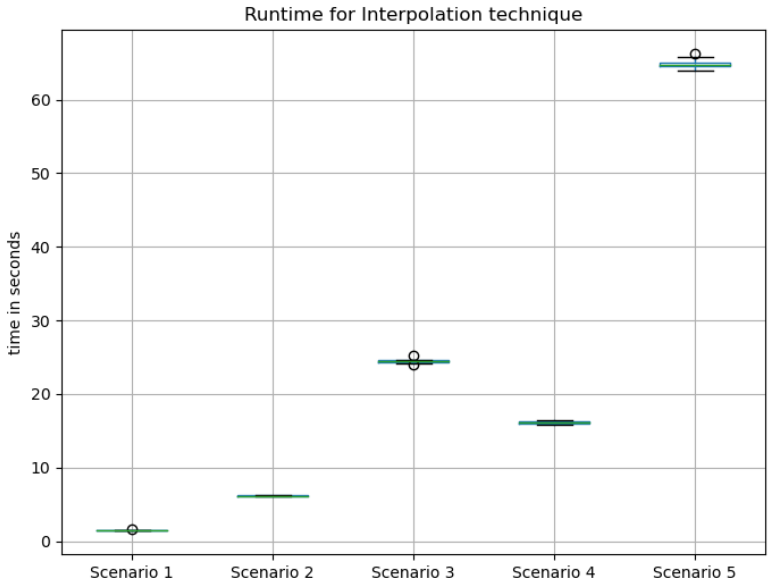
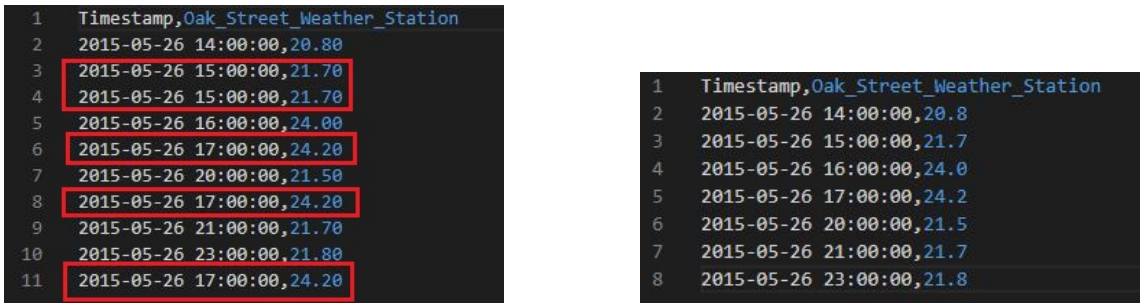


Figure 9.2: Runtimes for the interpolation technique in the various scenarios.

	1	2	3	4	5
Default	0.81	3.25	13.62	12.40	50.63
Deduplication	1.51	6.02	24.29	15.80	64.36
Interpolation	1.49	6.16	24.44	16.10	64.80

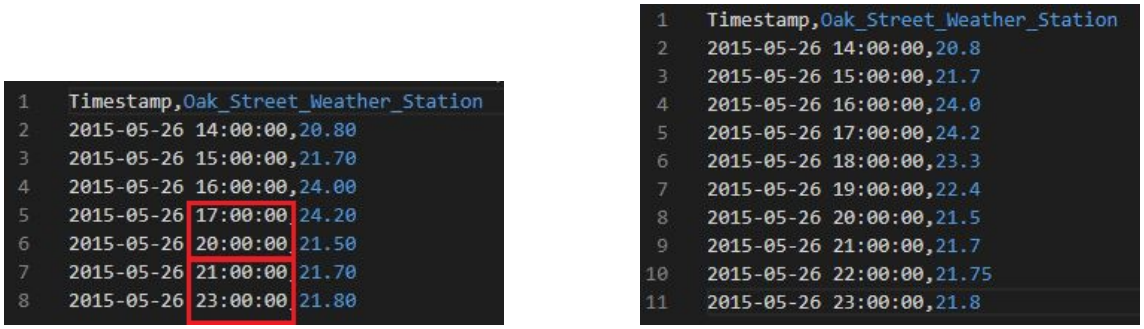
Table 9.1: Median runtimes of the enhancement techniques in the various scenarios.



```
Timestamp,Oak_Street_Weather_Station
2015-05-26 14:00:00,20.80
2015-05-26 15:00:00,21.70
2015-05-26 15:00:00,21.70
2015-05-26 16:00:00,24.00
2015-05-26 17:00:00,24.20
2015-05-26 20:00:00,21.50
2015-05-26 17:00:00,24.20
2015-05-26 21:00:00,21.70
2015-05-26 23:00:00,21.80
2015-05-26 17:00:00,24.20
```

```
Timestamp,Oak_Street_Weather_Station
2015-05-26 14:00:00,20.8
2015-05-26 15:00:00,21.7
2015-05-26 16:00:00,24.0
2015-05-26 17:00:00,24.2
2015-05-26 20:00:00,21.5
2015-05-26 21:00:00,21.7
2015-05-26 23:00:00,21.8
```

Figure 9.3: In the left image, there are three redundant data points that are highlighted together with their respective original data point (data points with timestamps 15:00:00 and 17:00:00). There are in total 10 data points, thus the duplicate metric of the dataset is 0.7. By applying the deduplication implementation, the data points seen in the right image is received, containing no duplicates, thus having a duplicate metric of 1. Hence, the deduplication technique was able to improve the quality of data.
The data seen in this figure is from the dataset "Beach Water Quality - Automated Sensors" provided by the City of *Chicago* [1]. The duplicate data points seen in the left figure were added manually, thus they can not be found in the original dataset.



```
Timestamp,Oak_Street_Weather_Station
2015-05-26 14:00:00,20.80
2015-05-26 15:00:00,21.70
2015-05-26 16:00:00,24.00
2015-05-26 17:00:00,24.20
2015-05-26 20:00:00,21.50
2015-05-26 21:00:00,21.70
2015-05-26 23:00:00,21.80
```

```
Timestamp,Oak_Street_Weather_Station
2015-05-26 14:00:00,20.8
2015-05-26 15:00:00,21.7
2015-05-26 16:00:00,24.0
2015-05-26 17:00:00,24.2
2015-05-26 18:00:00,23.3
2015-05-26 19:00:00,22.4
2015-05-26 20:00:00,21.5
2015-05-26 21:00:00,21.7
2015-05-26 22:00:00,21.75
2015-05-26 23:00:00,21.8
```

Figure 9.4: In the left image, three data points are missing, at times 18:00, 19:00, and 22:00. There are 7 data points present, thus the completeness metric of the dataset is 0.7. By applying the interpolation technique, the missing data points are interpolated, increasing the completeness from 0.7 to 1, see the right image. Hence, the interpolation technique was able to improve the quality of data.
The data seen in this figure is from the dataset "Beach Water Quality - Automated Sensors" provided by the City of *Chicago* [1].

This thesis report displays figures above containing data that has been modified for use from its original source, www.cityofchicago.org, the official website of the City of Chicago. The City of Chicago makes no claims as to the content, accuracy, timeliness, or completeness of any of the data in the figures on this page. It is understood that the data presented on this page of the thesis report is being used at one's own risk.

10
Discussion

This chapter discusses the results of the experiments presented in Chapter 6, 8, and 9. Additionally, threats to the validity of the results are discussed.

10.1 Result Discussion

10.1.1 Data Quality Dimension Selection

The selection of data quality dimensions in Chapter 6 provides the reader with options on how to determine the quality of sensor data. While some of the dimensions may not be of interest, the overall selection can contribute to the data quality dimension selection of a specific device or domain. The advantage of this selection is that the dimensions are domain-agnostic, and thus it should theoretically be possible to utilize the dimensions in the selection on any sensor data.
The selection was divided into lists, a main and a secondary selection. The separate selections were made since some data quality dimensions provided insights that were relevant and important in some cases, and in other cases, would be unnecessary and redundant. For a practitioner wishing to assess the quality of some sensor data, a recommendation would be to first consult the dimensions in the main selection, and possibly adding dimensions from the secondary selection if they are relevant for the domain of the sensor data.
The data quality dimension selection was based on the literature review and the characteristics of sensor data. There are technically no "right" or "wrong" data quality dimensions for sensor data. Therefore, there might exist data quality dimensions that could be relevant for some kind of sensor data but were not considered. Also, some of the selected data quality dimensions should be overlooked in some instances, since they would not be suitable in every situation. An example of this is a voice assistant and the data quality dimension currency. If the owners of a voice assistant are on vacation, the voice assistant will not be used for a long time. Thus, no new data will be updated, and the currency will decrease as a result of this. Even though the currency decreases, there is no problem in this scenario. Devices like voice assistants do not create a continuous stream of data, they are meant to be used sporadically. Hence, the presented selected data quality selection is a suggestion regarding what should be considered when assessing the quality of sensor data. Some examples of which situations the data quality dimensions can be used in: Consider the example with the fridge, as explained in Section 2.7, i.e., a fridge with a temperature sensor that provides the fridge with the knowledge if it should lower the

temperature. In that case, an intrinsic dimension like accuracy or precision could be of interest since we must ensure that the values given from the temperature sensor are reliable. Furthermore, the currency dimension would also be important since we require a constant stream of data from the temperature sensor, i.e., a high currency. If the currency were low, the fridge would not know if it should lower the temperature.
Another example, assume a surveillance camera. It is crucial that no data is missing, thus the completeness dimension would be valuable. Furthermore, it is important that the sensor data is received quickly, otherwise, the detection of a potential robbery would be delayed. Thus, measuring timeliness would be vital.
In the case where the intention of collecting sensor data is to create a dataset for training a machine learning model, it is crucial that the data depicts the real world accurately. Thus the accuracy dimension would be valuable. Furthermore, the presence of duplicates could skew the distribution of the dataset, which would lead to a less accurate machine learning model. Thus, the duplicate dimension would be appropriate in this context.
When the duration of collecting sensor data is long, the consistency dimension could be important, this could, for instance, be the case for a smart thermostat. When the sensor is operating during long times, like multiple years, the sensor could degrade and get worse. An accuracy consistency metric would be able to detect changes created by the degradation of a sensor's accuracy. However, the accuracy consistency technique developed in this thesis has a space complexity of $\Omega(n)$ where n is the total number of collected data points, see Table 8.2. Thus, in this specific implementation, the technique would realistically not be able to run for long periods of time, since that would require a lot of space. Nevertheless, this implementation of accuracy consistency is just one specific implementation, accuracy consistency could be implemented slightly differently, e.g., by sampling data points, and thus improving the space complexity, and allowing measuring the accuracy consistency during long periods of time.

10.1.2 Data Profiling-Based Quality Assessment Techniques

In Chapter 8, we evaluated implementations of techniques intended to determine the data quality dimensions defined in Chapter 6. The final runtime of each technique is listed in Table 8.1. In the evaluated scenarios, 4.000 or 16.000 data points were used. In Scenario 5, which had 16.000 data points, the fastest technique, accuracy, took roughly three seconds longer than the default case. The default case refers to the framework running without any data quality assessment being performed. Thus, the accuracy technique can be estimated to process a single data point in $3/16000 \approx 0.00019$ seconds (or 0.19 milliseconds). The slowest technique, which was aggregated precision, took almost 63 seconds longer than the default case. Thus, the aggregated precision can be estimated to process a single data point in $63/16000 \approx 0.0039$ seconds (or 3.9 milliseconds). Both these times are fast, almost instantaneous. The framework can, therefore, update a data quality metric very fast, thus enabling the detection of faulty datasets quickly.
However, the evaluations were based on only assessing one single data quality dimen-

sion. In reality, multiple different techniques would probably be used in conjunction. To get a more realistic time, assume we would like to assess all the seven data quality dimensions given in Chapter 6. We need to select one technique for each of the data quality dimensions, and summing their runtimes together with the default case runtime. To make a conservative estimation, we take the slowest technique of each data quality dimension. The total time of this would be 195.19 seconds. Thus, the total time to process a single data point in this scenario would be approximately $195.19/16000 \approx 0.012$ seconds (or 12 milliseconds). To get a grasp of whether this time is reasonable, we could compare it to the update frequency of the sensors that created the underlying datasets for the various evaluation scenarios. The sensors' frequency was 10 Hz, i.e., a new data point each 0.1 second. For a framework to be reasonable, it would need to process the data at a much faster rate than the data is being generated at. The rate of which the combined slowest techniques processed a data point was 0.012 seconds. This is over eight times faster than the data is being generated. Thus, when evaluating all the seven dimensions with the slowest techniques presented in this thesis, it is still feasible to assess the quality in real-time in scenarios similar to the ones used in this thesis (at least for a Raspberry Pi).
The techniques in the framework were developed to run sequentially, in reality, it would be possible to implement the techniques to run in parallel. Parallelizing the techniques would make the framework substantially more efficient when using multiple techniques.
Accuracy: The three accuracy techniques "Basic Accuracy", "Accuracy Ratio", and "Average Accuracy" were the fastest techniques in Scenario 4 and 5, with a runtime in Scenario 5 only 3-4 seconds slower than the default case. Their speed was probably due to their time complexity being linear in the number of sources, and the number of sources in the various scenarios being either 1 or 4. The aggregated accuracy dimension was a few times slower, adding roughly 15 seconds in Scenario 5 over the default case. The aggregated accuracy technique was expected to be slower even though its expected time complexity was the same as the others since it required more operations than the non-aggregated accuracy techniques. Nevertheless, the aggregated accuracy technique is still reasonable and quite fast. However, if multiple sources are available, it is probably better to select one of the other accuracy techniques, as the main advantage of the aggregated accuracy is that it can be applied to single-source sensors.
Precision: The two precision techniques are the techniques that were slowest in every scenario they were evaluated on. In Scenario 5, the basic precision technique is 25 seconds slower than the default case, and the aggregation precision is over 60 seconds slower. While 25 seconds may not seem that bad, compared to the non-aggregated accuracy techniques, it is around ten times slower. Important to note also is, as discussed in Section 6.3.2.4, precision is similar to accuracy, and generally offers little extra information when used in conjunction with accuracy. Due to precision being that much slower, yet measuring almost the same thing as accuracy, precision should not be assessed if it is not crucial for the task at hand (or there are abundant computational resources available). The same holds for the precision aggregation technique, it is significantly slower than the accuracy aggregation technique. The relatively slow speeds of the precision techniques are a bit surprising,

especially for the "basic" precision technique, since it has the same time complexity as the non-aggregated accuracy techniques. The precision should be slower than its accuracy counterpart, as it requires more computations. However, a result of ten times the time of the accuracy techniques is still surprising. One aspect that differentiates the precision techniques to the other implemented techniques is that they were implemented with *Numpy*. Operations used within the *Numpy* library could be the reason for the slower speeds. However, time constraints prevented a deeper analysis of this.

Timeliness: Both the timeliness techniques were reasonably efficient compared to other techniques, both of which were just around 15 seconds slower in Scenario 5 than the default case. They were expected to be fast since they had a time and space complexity of $O(1)$.

Completeness: The single completeness technique was efficient, adding slightly above ten seconds over the default case in Scenario 5. The completeness implementation assumes a stream of data points with a constant frequency, and that the frequency is known. This assumption could be seen as a bit simple, however, in many sensor systems, this is the case. While doing a more complex completeness implementation could be better in some contexts, it requires information about the context. In conclusion, the implementation proposed is efficient and should function in most contexts.

Consistency: Using consistency as a data quality dimension as defined in this thesis is a bit unusual, as described in Section 6.3.2.1. Due to this, the implementation and presentation of the consistency were not evident. The origination of the implementation was to compute some constancy, i.e., the ratio of the most frequency event/value to the total number of events/values, see Section 3.2.1.1.3. The implemented accuracy consistency is logical in the sense that in a sensor environment with multiple sensors, the presence of some constant difference among the sensors can be assumed, due to possible miscalibrations. The completeness consistency was implemented so that it will be able to notice a change in completeness. However, for the completeness consistency to notice changes in completeness, it requires that before and after that "change", the completeness is relatively constant, which might not be the case. Nevertheless, both of the implemented consistency techniques performed relatively well, especially considering the number of operations both of the techniques had. The accuracy consistency was just seven seconds slower than the default case in Scenario 5, and the completeness consistency was roughly 15 seconds slower.

Currency: Both of the currency techniques were fast, just above ten seconds slower in Scenario 5 than the default case. The currency techniques were faster than all other techniques in Scenario 1, 2, and 3. The currency techniques were expected to be fast as they do not technically process any received data, they only measure time. Due to this, it was expected that the currency techniques would be the fastest among all techniques. However, as Table 8.1 shows, they are slower than the non-aggregated accuracy techniques in Scenario 4 and 5. The fact that the currency techniques were slower than the non-aggregated accuracy techniques is probably because the currency is implemented slightly differently from the other techniques. As explained in Section 8.2.5, the currency techniques need to be updated even though

no new data is available. This mechanism was implemented as an ad-hoc solution by letting the main loop of the framework call all currency techniques an extra time each iteration of the loop, a call that would happen regardless of whether or not new data was available. In practice, it resulted in the currency techniques being called twice for each iteration of the loop during the evaluations. However, even though the currency techniques are slower than the non-aggregated accuracy techniques, they are still relatively fast overall.
Duplicates: The single duplicate assessment technique was efficient, and it has coincidentally almost identical results to the completeness technique. However, the absence of a significant amount of duplicates may probably be the reason for its runtimes. In contexts where duplicates are common, the duplicate technique would probably have been slower. However, that could be fixed by removing the step of removing an object in the queue whenever a duplicate is found. Then, the duplicate technique will only be dependant on the speed of hash maps, which is expected to be constant. Additionally, the duplicate technique could easily be modified by instead of just measuring duplicates, also removing or filtering out duplicates. The proof of concept deduplication algorithm presented in Section 9.1.1 and the duplicates technique presented in 8.2.6 are very similar.

10.1.3 Data Enhancement

Based on the data quality selection of Chapter 6, Chapter 9 presents techniques and algorithms on how to improve the data quality concerning those dimensions. The presented enhancement techniques provide information on how to combat problems with data quality. However, Chapter 9 is mostly limited to just presenting various solutions, except for the proof-of-concept implementation of a deduplication technique and an interpolation technique. The runtimes of those techniques were similar to most of the runtimes of the data profiling-based quality assessment techniques, i.e., they were below the 1600 (and 400) seconds threshold, thus they are technically feasible. In the context of edge computing, it is crucial to ensure that the techniques are efficient enough to be applied to computationally limited devices, which the implemented techniques proved to be. Furthermore, as shown in Figure 9.3 and 9.4, the implemented enhancements proved to improve to quality of sensor data. The datasets presented in those figures were very small, this was done to illustrate the improvement, it could trivially be done in larger datasets as well.
To enhance data in real-time, change or removal of data points must be done automatically without any human judgment. Data that seem to be faulty and in need of modification could actually be correct. The challenge of detecting actual faulty data points makes it very difficult to automate the enhancement of data. When applying enhancement techniques, there should be sufficient information about the domain, so no correct data gets accidentally modified. If an enhancement technique were to modify correct data points, it would decrease the quality of data instead of improving it.
While the method of researching data enhancement techniques was mostly limited to presenting various techniques, doing any actual large-scale implementation and evaluation would be difficult with the full scope of this thesis. To further investigate

enhancement techniques on sensor data concerning data quality, it should be done on a specific domain or product with a specific purpose. A further investigation would benefit from limiting the type of enhancement technique to just a subset of available techniques available, due to the vast amount of enhancement techniques. Many machine learning methods are based on learning some pattern and then performing predictions and inferences. Among the current machine learning methods, there are probably many methods that would be suitable for data enhancement.

10.2 Threats to Validity

This section highlights possible threats to the validity of the result of this thesis.

10.2.1 Internal Validity

The techniques were evaluated in situations where data was already prepared in pre-generated datasets and loaded into the framework before any analysis was performed. In a real-world scenario, data points are generated continuously in real-time. This difference in the evaluation scenarios and the real-world might introduce differences in the result of this thesis compared to what would be attained in the real world.
Theoretically, there could be some implications of performing the evaluations on pre-generated datasets. However, the framework is implemented so that it will not make any difference whether real-time or pre-generated data is used. In both cases, data will be loaded into the input queues, see Figure 7.5. Furthermore, the decision to use pre-generated datasets ensures that the techniques are evaluated on the same data, and the evaluations avoid unpredictable external errors that could occur between the framework and sensors (e.g., missing data).
An additional internal validity threat is that the techniques are evaluated by the total time it takes for them to assess 16.000 data points (or 4.000), and the time it takes to assess a single data point is estimated by diving the total time with the number of data points. By this approach, we might miss outliers in the dataset, i.e., some specific data points might take significantly longer to assess than the average data point. In a real-world scenario, it would be preferable that all data points, even outliers, are being assessed at a rate that is higher than the update frequency. However, all techniques were shown the be considerably faster than the update frequency of the original dataset. In the case of some rare outliers occur that take longer time, the framework might get slightly delayed for a brief moment, but will, due to the average speed of the framework, soon be up to date with the current data. Furthermore, the time complexity analysis provided an upper bound on how long time processing a single data point could take. Most techniques had either constant time complexity or linear in a low variable, and most of the techniques had the same time complexity for the worst-case and the average case. Hence, we know that, theoretically, outliers that take significantly more time to assess than the average data point does not exist (in the techniques which have the same time complexity in Big O and Big Omega notation).

10.2.2 External Validity

The result of evaluating data processing implementations, which in this case is the data profiling-based quality assessment techniques, are heavily dependant on the data they are processing. For instance, the duplication assessment technique's speed depends on the number of duplicates present in the dataset. Consequently, it will perform well on datasets with fewer duplicates and worse on data with a high number of duplicates. The issue with the evaluations done in this thesis is due to evaluation scenarios. The scenarios are based on the same datasets, which were created by the same device during the same specific period. In other contexts, with different data, the result might be different. However, by using actual datasets, we have shown the actual runtimes of the techniques in a specific domain. Thus, the result is applicable to similar domains.
Along with the quality of a dataset, the total size of a dataset, the number of different sensors, and the number of sensor sources might also affect the outcome. The aspects of a different total size of the data set, a different number of sensors, and the number of sensor sources were emulated in the various evaluated scenarios. Although all the scenarios contained the same datasets, it still gives an idea and an intuition about how well the techniques might perform in other contexts.
An additional variable that affects the runtimes of the techniques is the hardware that is used. In this thesis, only one specific computer was used, a Raspberry Pi. If a different computing device would have been used, the results would probably be different. However, even though the results would be different, the difference in runtime would probably be similar, i.e., if technique **A** is slower compared to technique **B** on a Raspberry Pi, **A** would probably also be slower than **B** on another device if the same implementations are executed.

10.2.3 Construct Validity

A literature study on, in this case, data quality dimensions, will provide the most commonly used and referred data quality dimensions in the scientific community, provided that it was performed systematically. However, it could miss how the data in the real world is assessed concerning its quality. How data quality is assessed in the real world is particularly important since it will display what aspects of data quality is important in the real-world. The original plan for the selection of data quality dimensions was to perform the literature study in conjunction with asking a set of open-ended data quality questions to engineers at *VCC* working with sensors or data collection, either by interviews or via emails. The result of these interviews would have improved the validity of this thesis since it would give insights into how data is assessed and evaluated in practice. The reason no interviews were done with engineers was due to the Coronavirus pandemic, which was active during most of the duration of this thesis, which made it challenging to prepare interview meetings or get in contact with engineers, which consequently lead to the decision of cutting that part of the thesis. However, the conducted literature study still provided many insights about data quality dimensions. The data quality dimensions covered by the literature study would probably encompass the data quality dimensions that would have been given by asking engineers at *VCC*, since they work in a specific

domain, only a subset of the data quality dimensions is probably relevant for them. Furthermore, some of the papers were actually applicable in a practical scenario as they focused on IoT-devices and described actual problems with sensor data, e.g., the paper [25].
A technique would need to be evaluated on an edge device to make conclusive statements if it is feasible or not for edge computing. In this thesis, the evaluations were limited to a Raspberry Pi computer. While Raspberry Pi can act as an edge computing substitute, it is not designed specifically to be a proper edge computing device. The validity of the evaluations is limited to the degree a Raspberry Pi is similar to an actual edge device. However, the most significant limitation of edge computing devices is computational power. The fact that the power of a Raspberry Pi computer is limited makes evaluations on a Raspberry Pi a reasonable emulation of how other limited computational devices would perform. Additionally, similar research has been done on edge computing that used Raspberry Pi computers as computational devices [42].

10.3 Ethical Aspects

While performing data profiling with the sole intent of assessing the quality of sensor data does not entail any ethical aspects, performing data profiling in general in some domains might have. For example, IoT-devices like "wearables" such as smartwatches which track individuals through sensors, e.g., heart rate monitors, thermometer, and GPS receiver. By performing data profiling on data from wearables, it might be possible to infer what the individual wearing the wearable is currently doing or underlying medical conditions, both of which might be sensitive to the individual. For instance, from a sudden rise in heart rate and a specific location from the GPS receiver, a data profiling technique might be able to find some correlation or association rules between these and some specific physical activity, e.g., exercising at a gym.
Furthermore, in situations where data profiling is applied to personal data like, for instance, passwords and other private information, it is important not to leak information, and also make sure it is not possible to infer the private information of someone given the result of the data profiling.
However, the data used in this thesis was not sensitive to any specific individual, and the framework itself is agnostic to the domain. Thus, it cannot infer anything about the content of the data itself since the final result from the framework is entirely contextual. Hence, no ethical dilemma was encountered during the work of this thesis.

11
Conclusion

This chapter presents the scientific contribution of this thesis, summarizes the answers to the initial research questions and presents possible future work similar to this thesis, and thus concludes this thesis report.

11.1 Scientific Contribution

The contributions of this thesis are the following: providing a set of data quality dimensions relevant for the quality assessment of sensor data, presenting a framework architecture suitable for the assessment of data, presenting data profiling-based quality assessment techniques aimed toward sensor data and evaluations of those techniques, and presenting enhancement techniques aimed toward improving the quality of sensor data.

11.2 Inferences of the Results

Each subsection in this section summarizes the answer to one of the research questions.

11.2.1 Data Quality Dimension Selection

The data quality dimension selection in Chapter 6 gave answers to the first research question, i.e., "Which data quality dimensions are relevant to determine the quality of sensor data?". The result of Chapter 6, and thus the answer to the first research question is the list:

1. Completeness
2. Accuracy
3. Timeliness
4. Consistency
5. Currency
6. Duplicates
7. Precision

These data quality dimensions cover most aspects of the data, i.e., the quality of the content of the data (accuracy, precision), time-related aspects of the data (timeliness, currency), and the quality on a dataset level (completeness, duplicates, consistency). While these seven dimensions should not be used in all cases when handling sensor

data, selecting a subset of these is probably relevant in many cases when assessing the quality of sensor data, or even data in general.

11.2.2 Data Profiling Implementation

To answer the second research question, i.e., "Can data profiling efficiently be used in edge computing to assess the quality of sensor data, and in that case, how?", a framework with a set of data profiling-based quality assessment techniques was implemented. The implemented techniques were developed to assess the data quality dimensions selected in Chapter 6, and they were all based on some data profiling principle (except for currency). Among the data profiling categories presented in Section 3.2, only cardinalities and value distributions were shown to be useful to assess the quality of data.
The developed techniques were described through text and pseudo-code. Their time and space complexity was analyzed and presented. Furthermore, the techniques were evaluated by assessing real sensor data. These things were done to ensure that the techniques are efficient and that they can be deployed on edge computing devices. As the results showed, all techniques were, in fact, feasible in all test scenarios. In Scenario 5, the majority of the techniques were roughly 10-15 seconds slower compared to the default case. The deviating techniques were the non-aggregated accuracy techniques and the precision techniques. The non-aggregated accuracy techniques were the fastest, in Scenario 5, they were roughly 3-4 seconds slower than the default case. Meanwhile, the two precision techniques were the slowest. They were roughly 25 and 63 seconds slower in Scenario 5 compared to the default case.

11.2.3 Data Enhancement Techniques

The data enhancement techniques presented in Chapter 9 answers the third research question, i.e., "Could sensor data of bad quality be efficiently enhanced?". The basis of the presented data enhancement techniques was that they would improve the quality regarding one of the data quality dimensions selected in Chapter 6. Two enhancement techniques were developed as a proof of concept, a data deduplication and an interpolation technique. They were evaluated by their time and space complexities, and their runtimes, like the data profiling-based quality assessment techniques. Furthermore, the techniques were proved to improve the data quality of a dataset. This was limited to the dimensions completeness and duplicates. Nevertheless, the proof of concept techniques proved enhancement of data quality was possible. Figure 9.3 and 9.4 shows the enhancement of the implemented techniques. The deduplication technique improved the duplicate metric from 0.7 to 1, and the interpolation technique improved the completeness metric from 0.7 to 1. Hence, the answer to the third research question "Could sensor data of bad quality be efficiently enhanced?" is yes, sensor data could be enhanced, and as shown in Table 9.1, the enhancement techniques were efficient.
Furthermore, various clustering and outlier techniques were presented, showing promise in detecting erroneous data and improving accuracy, consistency, and pre-

cision. However, Chapter 9 did not provide any technique to improve the timeliness or the currency of a dataset, this was not done since those dimensions are dependant on time.

11.3 Future Work

A possible extension of this thesis could take the same approach, but limit the research to a specific domain, like a specific IoT-device. Deploying the techniques presented in this thesis or deploying similar techniques in real-world situations would be exciting as it would give solid evidence that the techniques are suitable for edge computing. Additionally, research within developing and deploying actual novel data enhancement techniques for edge computing could be a compelling direction for future work within this subject. With the emerging interest of machine learning and deep learning, it would be entirely plausible that utilizing such techniques to try to improve data quality would be successful.

Bibliography

[1] CITY OF CHICAGO. Beach water quality—automated sensors, 2019.

[2] J Quain. As cars collect more data, companies try to move it all faster. *The New York Times*, August 2018.

[3] Markus Böhm, Stefanie Leimeister, CHRISTOPH Riedl, and Helmut Krcmar. Cloud computing and computing evolution. *Technische Universität München (TUM), Germany*, 2010.

[4] W. Shi and S. Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, May 2016.

[5] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[6] Tien Fabrianti Kusumasari et al. Data profiling for data quality improvement with openrefine. In *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 1–6. IEEE, 2016.

[7] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Profiling relational data: a survey. *The VLDB Journal*, 24(4):557–581, 2015.

[8] Felix Naumann. Data profiling revisited. *ACM SIGMOD Record*, 42(4):40–49, 2014.

[9] Markus Böhm, Stefanie Leimeister, CHRISTOPH Riedl, and Helmut Krcmar. Cloud computing and computing evolution. *Technische Universität München (TUM), Germany*, 2010.

[10] Amazon web services (aws). `https://aws.amazon.com/`. Accessed: 2020-03-31.

[11] Dialogic corporation. Introduction to cloud computing, 2017.

[12] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[13] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[14] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[15] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[16] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile edge computing: A survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[17] David Reinsel, John Gantz, and John Rydning. The digitization of the world from edge to core. *IDC White Paper*, 2018.

[18] Jetson nano developer kit. `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`. Accessed: 2020-03-31.

[19] Raspberry pi. `https://www.raspberrypi.org/`. Accessed: 2020-03-31.

[20] Vivek Teegalapally, Kiran Dhote, Vamsi S Krishna, and Shubham Rao. Survey on data profiling and data quality assessment for business intelligence. *International Research Journal of Engineering and Technology (IRJET) e-ISSN*, pages 2395–0056, 2016.

[21] Wei Dai, Isaac Wardlaw, Yu Cui, Kashif Mehdi, Yanyan Li, and Jun Long. Data profiling technology of data governance regarding big data: review and rethinking. In *Information Technology: New Generations*, pages 439–450. Springer, 2016.

[22] Leo L Pipino, Yang W Lee, and Richard Y Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.

[23] Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.

[24] Carlo Batini and Monica Scannapieca. Data quality dimensions. *Data Quality: Concepts, Methodologies and Techniques*, pages 19–49, 2006.

[25] Aimad Karkouch, Hajar Mousannif, Hassan Al Moatassime, and Thomas Noel. Data quality in internet of things: A state-of-the-art survey. *Journal of Network and Computer Applications*, 73:57–81, 2016.

[26] A. Klein and W. Lehner. Representing data quality in sensor data streaming environments. *J. Data and Information Quality*, 1(2):10:1–10:28, September 2009.

[27] Tamraparni Dasu and Theodore Johnson. *Exploratory data mining and data cleaning*, volume 479. John Wiley & Sons, 2003.

[28] YU Huh, FR Keller, Thomas C Redman, and AR Watkins. Data quality. *Information and software technology*, 32(8):559–565, 1990.

[29] Fatimah Sidi, Payam Hassany Shariat Panahy, Lilly Suriani Affendey, Marzanah A Jabar, Hamidah Ibrahim, and Aida Mustapha. Data quality: A survey of data quality dimensions. In *2012 International Conference on Information Retrieval & Knowledge Management*, pages 300–304. IEEE, 2012.

[30] Li Cai and Yangyong Zhu. The challenges of data quality and data quality assessment in the big data era. *Data science journal*, 14, 2015.

[31] Ikbal Taleb, Hadeel T El Kassabi, Mohamed Adel Serhani, Rachida Dssouli, and Chafik Bouhaddioui. Big data quality: A quality dimensions evaluation. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 759–765. IEEE, 2016.

[32] Theodore Johnson. Data profiling. *Encyclopedia of Database Systems*, 1:604–608, 2009.

[33] Merriam-Webster.com Dictionary. metadata.

[34] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[35] Varun Chandola and Vipin Kumar. Summarization–compressing data into an informative representation. *Knowledge and Information Systems*, 12(3):355–378, 2007.

[36] Jonathan I Maletic and Andrian Marcus. Data cleansing: Beyond integrity analysis. In *Iq*, pages 200–209. Citeseer, 2000.

[37] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, 2017.

[38] Xiang Sun and Nirwan Ansari. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[39] Bin Cheng, Apostolos Papageorgiou, and Martin Bauer. Geelytics: Enabling on-demand edge analytics over scoped data sources. In *2016 IEEE International Congress on Big Data (BigData Congress)*, pages 101–108. IEEE, 2016.

[40] Xiaomin Xu, Sheng Huang, Lance Feagan, Yaoliang Chen, Yunjie Qiu, and Yu Wang. Eaaas: Edge analytics as a service. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 349–356. IEEE, 2017.

[41] Roberto Casado-Vara, Fernando de la Prieta, Javier Prieto, and Juan M Corchado. Blockchain framework for iot data quality via edge computing. In *Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems*, pages 19–24, 2018.

[42] Jianhua He, Jian Wei, Kai Chen, Zuoyin Tang, Yi Zhou, and Yan Zhang. Multitier fog computing with large-scale iot data analytics for smart cities. *IEEE Internet of Things Journal*, 5(2):677–686, 2017.

[43] Hugo Hromic, Danh Le Phuoc, Martin Serrano, Aleksandar Antonić, Ivana P Žarko, Conor Hayes, and Stefan Decker. Real time analysis of sensor data for the internet of things by means of clustering and event processing. In *2015 IEEE International conference on communications (ICC)*, pages 685–691. IEEE, 2015.

[44] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. Data profiling: A tutorial. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1747–1751, 2017.

[45] Yang Zhang, Nirvana Meratnia, and Paul Havinga. Outlier detection techniques for wireless sensor networks: A survey. *IEEE communications surveys & tutorials*, 12(2):159–170, 2010.

[46] Nagapramod Mandagere, Pin Zhou, Mark A Smith, and Sandeep Uttamchandani. Demystifying data deduplication. In *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, pages 12–17, 2008.

[47] Jonathan I Maletic and Andrian Marcus. Data cleansing: Beyond integrity analysis. In *Iq*, pages 200–209. Citeseer, 2000.

[48] Petr Berka. Data cleansing using clustering. In *Man–Machine Interactions 4*, pages 391–399. Springer, 2016.

[49] Alan Hevner and Samir Chatterjee. Design science research in information systems. In *Design research in information systems*, pages 9–22. Springer, 2010.

[50] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[51] Google scholar. `https://scholar.google.se/`. (Accessed on 06/23/2020).

[52] Bestoun S Ahmed, Miroslav Bures, Karel Frajtak, and Tomas Cerny. Aspects of quality in internet of things (iot) solutions: A systematic mapping study. *IEEE Access*, 7:13758–13780, 2019.

[53] Anja Klein. Incorporating quality aspects in sensor data streams. In *Proceedings of the ACM first Ph. D. workshop in CIKM*, pages 77–84, 2007.

[54] Sasank Reddy, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. A framework for data quality and feedback in participatory sensing. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 417–418, 2007.

[55] Sabrina Sicari, Alessandra Rizzardi, Cinzia Cappiello, Daniele Miorandi, and Alberto Coen-Porisini. Toward data governance in the internet of things. In *New advances in the internet of things*, pages 59–74. Springer, 2018.

[56] Aimad Karkouch, Hajar Mousannif, Hassan Al Moatassime, and Thomas Noel. A model-driven architecture-based data quality management framework for the

internet of things. In *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pages 252–259. IEEE, 2016.

[57] Jung-Eun Kim, Tarek Abdelzaher, Lui Sha, Amotz Bar-Noy, Reginald Hobbs, and William Dron. On maximizing quality of information for the internet of things: A real-time scheduling perspective. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 202–211. IEEE, 2016.

[58] Yongrui Qin, Quan Z Sheng, Nickolas JG Falkner, Schahram Dustdar, Hua Wang, and Athanasios V Vasilakos. When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications*, 64:137–153, 2016.

[59] Diane M Strong, Yang W Lee, and Richard Y Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.

[60] Giri Kumar Tayi and Donald P Ballou. Examining data quality. *Communications of the ACM*, 41(2):54–57, 1998.

[61] Howard Veregin. Data quality parameters. *Geographical information systems*, 1:177–189, 1999.

[62] Christopher Fox, Anany Levitin, and Thomas Redman. The notion of data and its quality dimensions. *Information processing & management*, 30(1):9–19, 1994.

[63] Yair Wand and Richard Y Wang. Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11):86–95, 1996.

[64] Monnit. Duplicate sensor readings.

[65] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[66] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[67] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[68] Szilárd Vajda and KC Santosh. A fast k-nearest neighbor classifier using unsupervised clustering. In *International conference on recent trends in image processing and pattern recognition*, pages 185–193. Springer, 2016.

[69] Bo Sheng, Qun Li, Weizhen Mao, and Wen Jin. Outlier detection in sensor networks. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 219–228, 2007.

[70] Alberto MC Souza and Joseé RA Amazonas. An outlier detect algorithm using big data processing and internet of things architecture. *Procedia Computer Science*, 52:1010–1015, 2015.

[71] Kjell Jørgen Hole. The htm learning algorithm. In *Anti-fragile ICT Systems*, pages 113–124. Springer, 2016.

[72] Kjell Jørgen Hole. Anomaly detection with htm. In *Anti-fragile ICT Systems*, pages 125–132. Springer, 2016.