Automatic DAC Quality Control

Bachelor Thesis: HBO-IT Software Engineering

Wybren Oppedijk

Ruud Greven (Graduation Supervisor)
Jeroen Linssen (Graduation Supervisor)
Willie Aarnink (Company Supervisor)

1 ABSTRACT

Malvern Panalytical is the manufacturer of diffractometers. Diffractometers are used to measure particle sizes and chemical composition. This project focuses on the sensor chip inside the machine. In the assembly of a machine, chips are tested in three different stages: first when the chips arrive on a wafer, secondly when a chip is mounted on chipboard and finally when the chipboard is mounted in a sensor module. The chips value is increased by a factor ten, for every assembly phase it proceeds.
The objective of this study investigates whether it is possible to create a model that uses the historical test data to label defect digital analogue converters on chips automatically. Automatically labelling should lead to a decrease in manual labour and fewer defect that proceed to a next assembly phase.
The study researches two approaches. One approach is using a cluster algorithms and the other by combining different statistical filters to classify defect chips.
The statistical model can consistently find more defect chips as that the human operator would. This model is capable of reducing the number of defect chips that normally would continue in the assembly process by 10 per cent. From this study also became clear that the current tests process should be more standardised, and that the current way of labelling is not specific enough. Further improvement may be to build model trained with all test data instead of only the partial test data.

TABLE OF CONTENTS

TABLE OF FIGURES

Glossary

API	Application Programming Interface
CRISP-DM	Cross-Industry Standard Process for Data Mining
CSS	Cascading Style Sheets
DAC	Digital to Analog Converter
KNN	K nearest neighbours
OS	Operating System
PCB	Printed Circuit Board
PDF	Portable Document Format
REST	Representational State Transfer
SVM	Support Vector Machine
URL	Uniform Resource Locator
Wafer	The semi-finished product that is used to create chips.

2 INTRODUCTION

Recently big data is a hot topic, it can improve all kinds of processes such as; better processes understanding being more effective in marketing or reducing costs by eliminating mistakes and automate manual labour.

This project is done for Malvern Panalytical. Malvern Panalytical is located in Almelo, where they build and develop diffractometers. Diffractometers are highly specialised measurement devices that work by sending an X-Ray beam on a sample. Based on the diffraction of the beams, particle sizes and chemical composition can be retrieved. An example use of a diffractometer is to ensure the quality of milk powder for babies.

The assembly of a sensor module happens in three steps. First, the chip is sawed from a wafer and placed on a circuit board, then the circuit board is mounted in a sensor module and finally the sensor module is assembled in the final machine.

In all these stages the chip is tested on defects. This testing procedure is crucial because, with a malfunctioning chip, the machine does not work or gives unexpected results.

The results of the different tests are currently manually reviewed. This review is prone to errors and takes much time. The goal is to investigate if it is possible to automatically detect defective chips with the tests data accumulated of the years.

3 Working Methodology

The research and development of the prototype are entirely done according to the steps of the CRISP-DM methodology, which stands for *Cross-Industry Standard Process for Data Mining* [1]. CRISP-DM is the de facto standard for approaching data science-related projects. It helps to approach the project in the right order and gives handlebars in considering if a particular task has succeeded or not. It also helps to assess if the project is on the right track. The CRISP-DM process is iterative, which means that a project can go through the cycle multiple times. It is also possible to stop halfway and start back at the begin because things might turn out different than anticipated at the beginning. One total iteration consists of six steps;

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modelling
5. Evaluation
6. Deployments

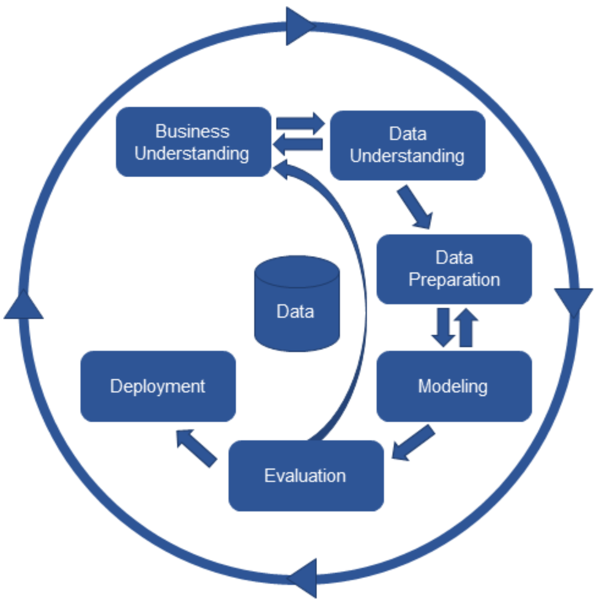


Figure 1: CRISP-DM

Due to the limited time of this project, the goal is to go at least once through all steps. The chapters in this report match the steps in the CRISP-DM model.

Business Understanding
The first step is Business Understanding and starts by asking the following questions:

- What is the goal of the project?
- What are the project boundaries?
- What is the impact of solving the problem?

Working on these questions should make it clear if it is feasible to succeed. At the end of this phase, there should be a list of precise requirements.

Data Understanding
Data Understanding is probably the most critical and time-consuming step of the whole project. The first question that should be asked is, is the data accessible? If this is answered by no, then find a way to make it accessible. Maybe the data is accessible but not yet insightful, and something should be developed to make it insightful. Also, a decision should be made if the quality of the data is good enough to solve the business goals. If this is not the case, the Business Understanding phase should be revisited and changed accordingly.

Data Preparation
In most cases, the data is not collected specifically for data science. Therefore, it needs some tweaking to make it suitable. Think of cleaning the data, reformatting it or artificially adding data if needed.

Modelling
In this phase, one knows what he can expect from the data and based on this knowledge, a technique for creating a model can be chosen. Different techniques can be compared to find the model that yields the best results. It is also essential to explain why a particular technique is chosen because the commissioning party should put its trust in the results. Therefore a technique which is easily explainable may have favour over something more advanced.

Evaluation
After the model is created, it has to be assessed if it is any good. The assessment will not only go about the model itself but also about the entire process around the project. This evaluation should give some points to improve upon next time combined with the how-to advice. Possibly a list is created with next steps or goals in the project.

Deployment
The deployment phase is the last step in the CRISP-DM methodology. This phase should make clear how an end-user can use the model in an application. Also, there will be a section of how maintenance or tweaks on the model can be made. Finally, this step includes presenting the results and writing those down in a report.

3.1 Planning

A planning methodology will be used to keep the project organised. This planning methodology will be inspired by Scrum. A standard Scrum project consists of a product owner, a scrum master and a development team. Because this a one-man project, this formation is not entirely possible. However, Scrum does have some useful aspects. For example, the project will have sprints of 1 week. Each sprint will be closed with a retrospective, and it will be checked against the definition of done.

Beforehand, an estimation is made how many sprints each phase of the CRISP-DM model will take, to ensure that at least one cycle has passed in the end. Every week two meetings take place with the product owner at Malvern Panalytical. During the first meeting is checked if tasks can be completed by the end of the week and if resources are needed to complete the task. During the second meeting, a sprint retrospective is held, and new tasks are created for the following sprint

The reason that an agile planning methodology fits well with CRISP-DM is that the process can always take a different direction than could have been anticipated beforehand.

3.1.1 Definition of Done

A task is finished if it meets the following requirements:

- The code tasks contain tests and these tests pass
- Acceptance Criteria are met
- A short presentation of the task can be given to Willie Aarnink
- Willie Aarnink agrees that the task is finished.

4 Business Understanding

4.1 Company Background

Malvern Panalytical is a high-tech company located in Almelo. The company is a spinoff of Philips in Eindhoven. A couple of years ago the Panalytical fused with the English company Malvern to bundle its knowledge. At Malvern Panalytical in Almelo, there are about 400 employees. The skillset of these employees is extensive. There are chemists, physicists, embedded software engineers and more. Malvern Panalytical recently started to deepen itself in big data and machine learning to optimise different processes.

4.2 Product Background

One of Malvern Panalytical its products is the Empyrean. The Empyrean is an X-Ray diffractometer. Among the capabilities of the diffractometer are measuring particle size, particle shape and chemical composition. An example of a company that uses a diffractometer is a Japanese demolition company. The government in Japan states that it is not allowed to demolish apartment buildings with asbestos in the concrete. This company uses the diffractometer to analyse the concrete and to see if it contains asbestos. Another example is a chip manufacturer who wants to know the thickness of the chip layers.

Figure 2: Malvern Panalytical Diffractometer

The machine itself contains multiple high-tech components. This project focuses on the assembly of digital to analogue converter (DAC) in the chip inside the machine. More about what a DAC is in *Appendix 1: What is a DAC*.

4.3 Project Motivation

The heart of the diffractometer is a sensor chip. The chip is an essential part of which the quality must be very high. The sensor module in the diffractometer is fully assembled at Malvern Panalytical. The assembly of a sensor module consists of 3 steps. First, a wafer arrives at Malvern Panalytical, which contains 109 chips. Every chip can potentially become a sensor module, see *Figure 3*. Each chip is tested on several different criteria. The testing of a wafer is very complex and requires specialised equipment; therefore, this is done by an external party. Each chip that meets the testing criteria is placed on a printed circuit board (PCB), see *Figure 4*. After this step, the chip is tested again; if it still meets the criteria, it is built into a sensor module. The same process happens one more time on the sensor module, see *Figure 5*. When the chip still functions according to expectations, it is built into the final machine.

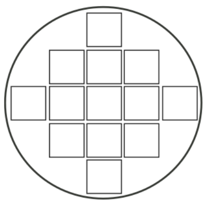


Figure 3: Wafer

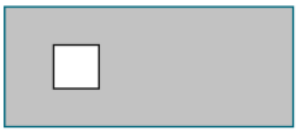


Figure 4: PCB

Figure 5: Sensor Module

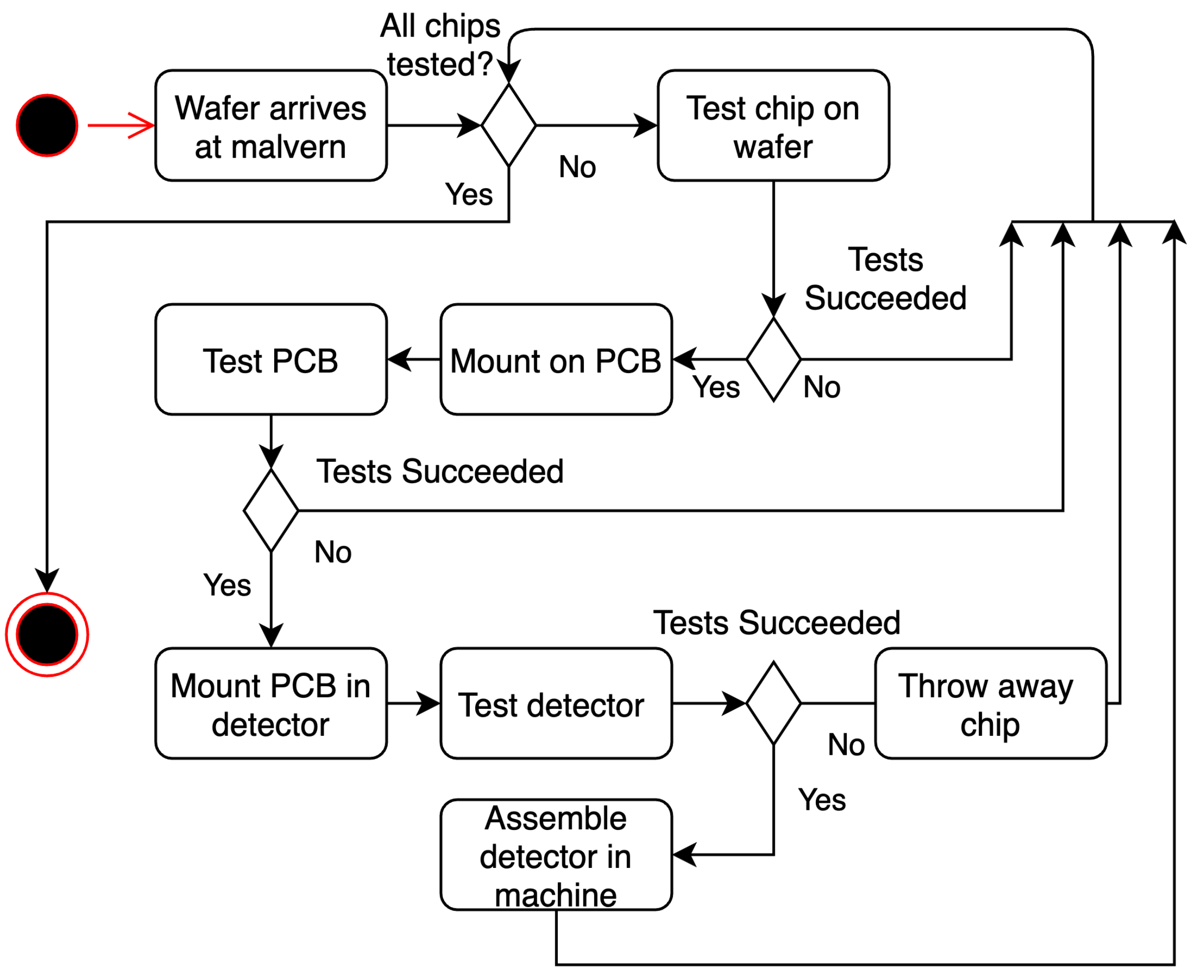


Figure 6: Chip Life Cycle

Figure 6 is a schematic representation of the assembly process. The project mainly focuses on the part of the tests that take place at wafer-level, specifically the tests that test DACs inside the chip. Each chip contains 27 different DACs, all of which must have a particular curve. A deviating curve can result in a disturbed measurement which makes the diffractometer unusable.

In the current situation, the tester receives the raw data from the company testing the wafers. This raw data is put in an excel file and studied with the bare eye by the tester. For each DAC a point is determined with a lower and upper limit, if the DAC curve does not pass this gate, the whole chip will be rejected. This way of testing is not accurate. There are a lot of chips that pass this filter but still don't work. See the illustration in *Figure 7,* the black line in the middle represents the gate where a DAC curve has to go through. The blue lines are broken but do not cross this gate. Therefore the tester has to view the measurements chip by chip to detect any deviations in a curve. This process is very labour intensive and takes almost 3 weeks for a batch of 25 wafers. This process is also sensitive to human error. For instance, the average height difference per wafer is not taken into account. This randomness leads to inconsistent classifications.

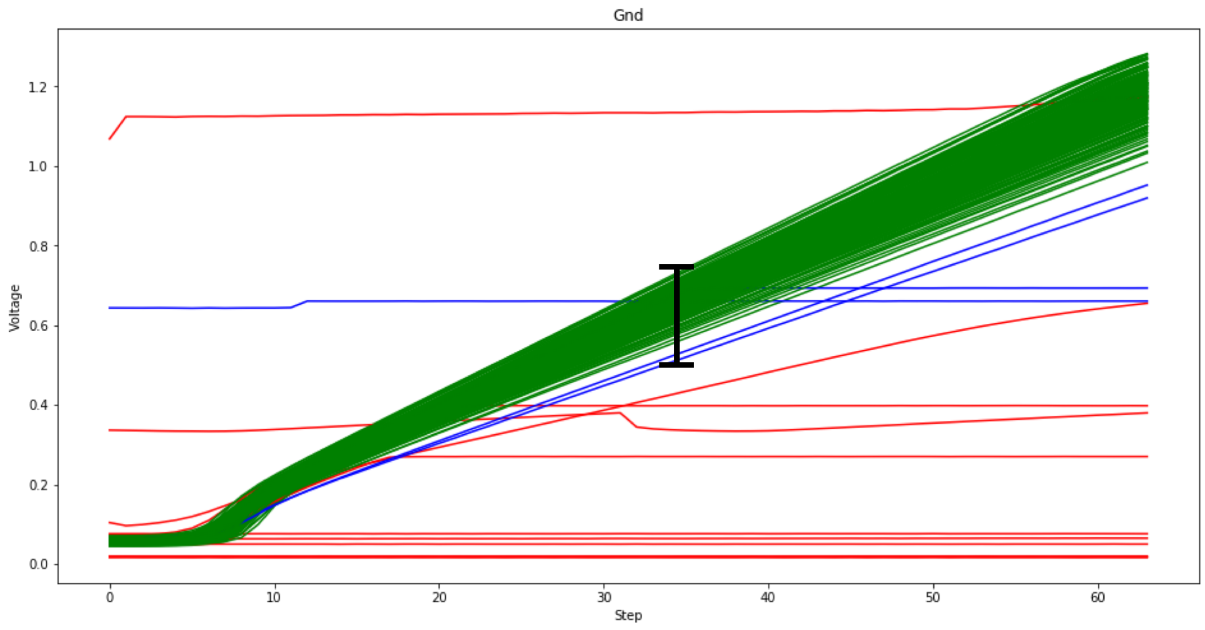


Figure 7: Gate Test Illustration

The goal of this project is to collect the test data centrally and make it insightful. With the centrally collected test data, a model must be created that detects faulty chips that pass the gate filter, thus detect the blue lines in *Figure 7.*

4.4 Research Questions

The main question in this project is: *How can Malvern Panalytical automatically label faulty DACs with historical measurement data?* This question alone raises a couple of sub-questions:

- How can the test reports data be made available in a database?
- How can faulty chips be classified?
- Is it possible to predict failure instead of classifying it?
- How can the results of a model be visualised?

4.5 Deliverables

At the end of the project, a pipeline should be delivered that consists of three components. This pipeline watches test directories for new raw test data. If new test data is added, the pipeline will give it a label and stores it into a database. Eventually, a frontend application can access the data through a REST API and visualise the data to the test operator.

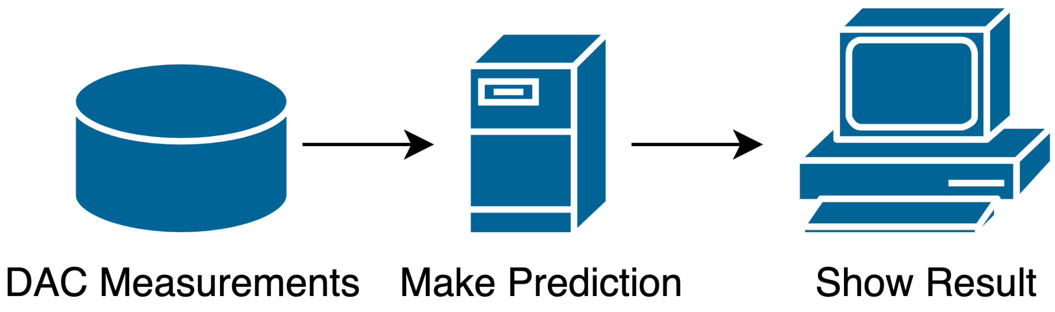


Figure 8: Software Pipeline

4.5.1 DAC Test Result Parser

The current testing process stores its results in text files on a server. Every test generates a new text file. In total, this means lots of directories with thousands of files inside it. Working with this system is not easy because there is no automated process to look up specific test data. Visualising is even worse since it is raw data. A parser could help to optimise this process. This parser parses all data from different text files into a database. With the help of the model, the DAC chips are classified as faulty or valid. After classification, the data is stored in a database in a way it is easily searchable by all its properties. All the test result should be accessible via a rest API so that a frontend can visualise the data.

4.5.2 DAC Failure Model

The model should be capable of identifying wrong DAC test curves. This model should work for all the 27 different DAC tests. The model should have the same performance or better as the current manual process.

4.5.3 Dashboard

The dashboard should visualise all the test data. Within this dashboard, a test operator should be able to lookup; wafers, PCB and detector tests. The dashboard should also visualise curves and show which are rejected. With this tool, an operator should see immediately, which DACs on the chip are failing.

4.6 Requirements

The deliverables have the following technical and functional requirements.

4.6.1 Functional Requirements

Nr.	Description
1.	The model can classify defective chips.
2.	The dashboard gives insight into why the chip is disapproved.
3.	The dashboard shows a list of bad chips to the test operator.
4.	The model can predict failure in a successive phase.

4.6.2 Non-Functional Requirements

Nr.	Description
1.	The backend programming language must be in C# or Python.
2.	There is a manual to operate the software.
3.	The software is easy to use.
4.	The software is easy to deploy.
5.	The software processes new data without manual interaction needed.

4.7 Project Boundaries

This project includes some boundaries to ensure success in the end. The project has the following boundaries.

- The backend will be programmed in Python due to my experience and the experience at Malvern Panalytical.
- The frontend is programmed in Angular due to my experience.
- Malvern Panalytical supplies the test data of all three phases. If the data changes later on due to any reason, this considered outside the scope of the project.
- The responsibility of verifying the quality of the classification model lays by the domain expert at Malvern Panalytical.
- Security for the database frontend and backend is not applicable. The tools will only be used within the company.

5 Data Understanding

Understanding the data is a crucial part of the process. In the data understanding phase, the goal is to explore the data and find out if the data quality is sufficient to reach the goals of Malvern Panalytical. During the data understanding, a tool is developed to gather the data and make it easier accessible for data science tools. Also, a frontend application is developed to give the tester more insight into the data.

5.1 Data Gathering

As mentioned earlier, Malvern Panalytical tests its chips in three different stages. Among the tests are the DAC tests. Each of the 27 different DACs is tested individually. All these tests output text files containing the measurements. Finding the right file is hard due to the complicated naming. The file itself contains raw numbers which are hard to interpret. The first step is to make a tool which gives easy access to the data, so exploring the data becomes more convenient. To answer the question "*How can the test reports data be made available in a database*?" the test reports first have to be examined. These test reports are stored in a directory as text files. The files together exceed 100GB, which lead to some challenges. For instance, moving the files over the network is not an option because this takes too much time. A better approach is to parse the text files from the storage server.

5.1.1 Parser

The goal of the parser is to transform the raw data stored in textfiles in different directories into a database. With the data inside a database, it becomes easier to perform thorough data analysis.
Let's take the wafer test files as an example to demonstrate the structure of such a data directory.

- Data
    - Wafer name
        - Chip1 Test A.txt
        - Chip1 Test B.txt
        - Chip2 Test A.txt

The second top folder contains the name of the wafer. The folder stores all text files. The name of the text file contains the chip identifier and which test result it contains. Per wafer, there are 2943 text files for all DAC tests.

The parser starts indexing all folders inside the test directory. The indexing speeds up by making use of multiple processor cores if these are available. Multithreading also introduces some extra difficulties. For example, two threads that try to parse the same file. Every thread has its folder, to prevent such behaviour. The operator can start the parser and can thereby specify the test directory that needs to be processed.

Inside the data directory, the parser creates a file named *parserDB*. This file contains every folder that the program has already successfully parsed. The program uses this file to prevent itself from reparsing a folder over and over again. After the program finishes with parsing, it starts up a file watcher. This file watcher detects changes inside the data directory. For example, if a tester adds a new folder to the test directory, the program automatically detects this and starts parsing the newly added folder.

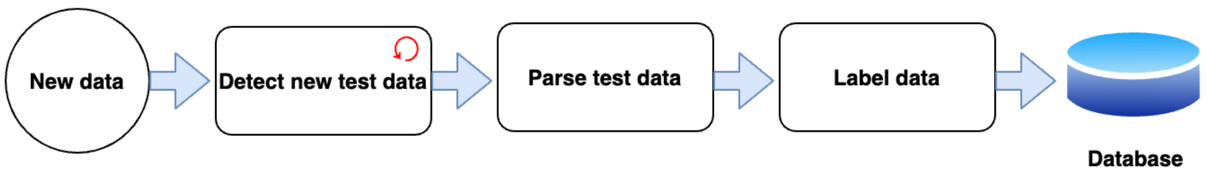


Figure 9: Parsing Diagram

Before the parser stores the data in the database, it first gets a label from the model. The reason that the test gets the label at parse time is that predicting a label is performance demanding. Therefore it is better to classify a label well in advance instead of classifying a label upon using the dashboard. If the model classified the data at request, it would slow down the system unnecessarily because the test data does not change over time; more information about the labelling in chapter *6 Modelling.*

5.1.2 Database

There are several different database technologies available. All these technologies have their advantages and disadvantages. In this chapter for every category, the most popular technology is chosen to find which technology suits the project the best. An important factor is how well it integrates with popular data science tools such as Matlab and Python, and if the data fits the design conventions of the database.

Hadoop

Apache Hadoop [2] is a popular technology in the field of data science. It is an opensource application written by the Apache foundation. The goal of Hadoop is to offer distributed storage and processing of big data. Hadoop stores data inside its own Hadoop Distributed File System. These files get distributed across nodes in a cluster. Besides storage Hadoop also offers processing tools with its MapReduce paradigm. Hadoop is particularly useful with high volume data streams and if you want real-time analytics.

The great thing of Hadoop is that it integrates well with python [3]. But Hadoop is mainly built for high volume data streams. This application will incidentally have a set of new data but does not have any streams. Another disadvantage is that Hadoop cannot cope with a large number of small files [4] because every file will end up in an HDFS block of 64MB. The text files are usually around 2KB, which creates an enormous memory overhead if stored in a 64MB block.

MySQL

MySQL [5] is one of the most established database technologies. It has excellent community support which makes finding resources easy. MySQL is a relational database management system. Relational databases reduce data duplication inside the database if there are relations in the data.

MySQL has a perfect integration with popular Data Science tools [6] [7]. However, the test data cannot benefit from reduced duplication since the data is not relational. MySQL requires to create a table before data can be inserted. If the data later changes, the table must be changed accordingly.

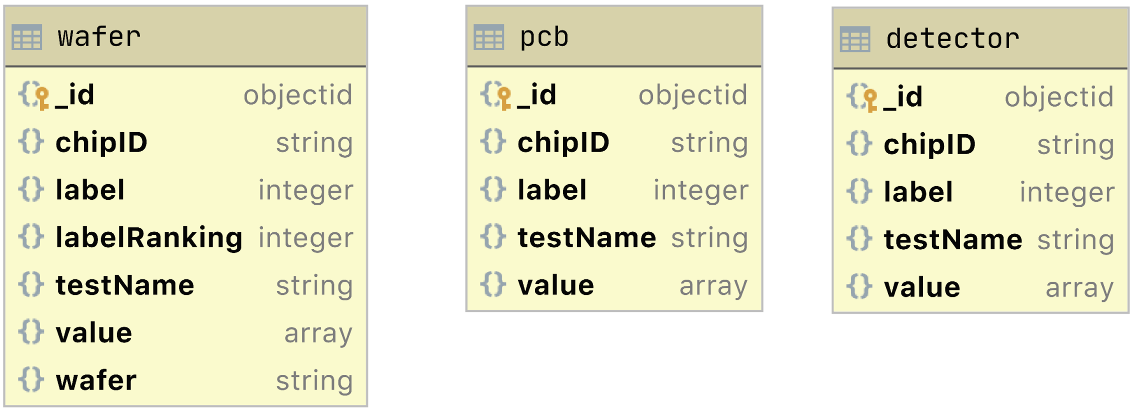
MongoDB
MongoDB [8] is a relatively new technology in database land. Its first release was in 2009. In opposite to MySQL, MongoDB is an unstructured database management system. MongoDB stores data inside files named documents that are written in Bson. Bson is highly similar to the data interchange format Json.

MongoDB has almost flawless integration with Python. Python data structures such as dictionaries can be persisted in the database with the bare minimum of database-specific query language [9]. Another advantage of MongoDB is that it does not require structure in the data.

Conclusion
Hadoop does not suit this project well because of two reasons. There is no constant stream of data, and the project is too small to make use of its distributed nature. Another disadvantage is that this system deals with lots of small files which Hadoop cannot handle well. MongoDB and MySQL are both valid options. However, there is also not an advantage to use MySQL over MongoDB since the data is unstructured and cannot benefit the advantage of a relational database. MongoDB, on the other hand, gives ease during development due to the great integration with Python. Another point in its favour is that in the current situation, only the data of DAC tests are stored. If later is decided to store additional tests data, this will be no problem. Where MySQL likely needs to change its table structure to cope with the new data. This comparison leads to the decision to use MongoDB for this project.

Database Design
The first approach to store the data inside MongoDB was to use one document for all the chip test results. After testing this approach, it quickly became apparent that data was missing and read queries were extremely slow. This behaviour had to do with MongoDB its limit of 16MB per document. Another design was made to cope with this limitation. Every single test result will become a single document. In this way, the limit 16MB will likely not be exceeded. This design also fits better with the conventions of MongoDB. After testing query times again, it was significantly faster.

Three collections are made to categories the data. Each category corresponds to a test phase. Every document is identifiable by the unique *chipID* in combination with the *testName.*



wafer	
_id	objectid
chipID	string
label	integer
labelRanking	integer
testName	string
value	array
wafer	string

pcb	
_id	objectid
chipID	string
label	integer
testName	string
value	array

detector	
_id	objectid
chipID	string
label	integer
testName	string
value	array

Figure 10: Different collections inside the MongoDB database

5.1.3 Data Service
After all the data has been converted from text files and stored in a MongoDB database, it is essential to investigate how the data can be efficiently made available to external applications.
Moving all the test files at once from one computer to another computer will create a slow system due to the large number of test files. This makes a web API a perfect solution for this. Because it only sends the data that is necessary over the network.

The API will be created following the REST convention instead of SOAP. This because REST is more lightweight and takes less time to develop [10]. The central aspect of this convention is that the API is stateless. Stateless means that you can make multiple requests in any particular order, and it should always return the same response. The API can be hosted on a server, and only the data that is necessary will be sent over the network to the end-user.

There are several different frameworks available to build a REST API. Malvern Panalytical uses C# and Python as their primary programming languages. Based on the preferences of Malvern Panalytical it is chosen to develop the application in Python. Python offers as most popular frameworks Django [11] and Flask [12]. Django contains the most features, but it takes more time to build an application with then Flask while Flask is lightweight and has not many features out of the box [13]. This Rest API for Malvern Panalytical will be used internally and will not be exposed to heavy load, nor is it vulnerable for outside attacks. For that reason, there are no security requirements, and speed and scalability are not an issue. This circumstance makes Flask the best framework since it is the quickest to implement. An added pre is that most data science-related tools use Python; this makes combining the tools more convenient.

The API itself only contains GET endpoints, since all data that will be added goes directly trough the parser to the database. After this, there are not going to be any mutations to the data. The URLs of the endpoint are built in a way they should explain which data can be expected. Extensive API documentation is available in *Appendix 3: API Documentation*.

5.2 DASHBOARD
There are different answers to the question "*How can the results of a model be visualised?*". The results of a model can be printed into a PDF or to a console. However, in this way, interacting with the system will be very hard. Also finding the right results is not easy because it is dependent on filenames and the file explorer of the operator's operating system. A desktop app could be a solution, but it requires everyone to install the app. A hosted dashboard, on the other hand, gives the possibility to interact with the data, and no one needs to install anything to access the application. For this reason, in consultation with Malvern Panalytical, it has been chosen to develop a dashboard web application.

To get the best out of the dashboard first, the wishes of the end-user must be clear. Therefore an interview was organised about how the domain expert currently visualises tests results. From this interview became clear that he visualises the data per wafer per one of the 27 DAC tests. His wishes for a dashboard would be that it is easy to search for a specific wafer. It should also be possible to visualise specific chips on a wafer, and the model should help to find deviating curves. The list of deviating DACs should be printed somewhere in the dashboard.

With the wishes of the domain expert a list of goals for the dashboard was put up;

- Easily search and navigate through wafers,
- Switch between test phases, wafer, PCB and detector,
- Plot all test of wafer together,
- Show a list of failed DAC tests,
- Have the possibility to isolate a single test.

With the goals set, the main flow of the dashboard should look like *Figure 11*.

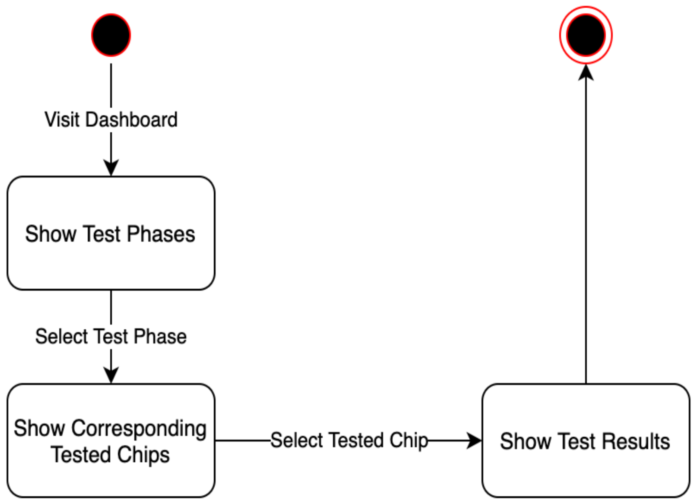


Figure 11: User flow of the dashboard

Malvern Panalytical had no requirements in which framework or language the dashboard should be programmed. The three most used web frameworks according to the stack overflow survey of 2019 [14] were; JQuery, React.js and Angular. All three frameworks are well-documented, and communities are large. All three would get the job done perfectly. My experience with Angular makes it the best choice because development will be the fastest.

Angular Material [15] is used to make the development of the frontend more efficient. Angular Material is a package with frequently used components such as tables, pagination buttons, switches. While using Angular Material, you are still free to change styles, but it is not needed to develop everything from scratch.

The dashboard will use a custom CSS theme to keep consistency in style all over the website. This is realised by using a custom theme which is possible with the CSS framework SASS [16]. SASS is an enhancement over traditional CSS. It is easier to read and gives possibilities to reuse your code. The dashboard has two files, one which defines spacings such as font size, margins and padding and the other which defines colours, see example in *Figure 12*. The style template makes changing the values easy because you only have to do it in one place instead of in every style sheet all over the website. It also gives the advantage to add other themes to a website at a later point such as dark mode or a light mode.

```
$oppedijk-spacing: (
  small: 10px,
  medium: 15px,
  large: 30px,
  extra-large: 60px
);

$oppedijk-radius: (
  small: 5px,
  medium: 8px,
  large: 10px
);

$oppedijk-font: (
  small: 12px,
  medium: 18px,
  large: 24px,
  extra_large: 48px
)
```

Figure 12: Sass template example

Angular applications are single page websites which mean there happens no real refresh on entering a new page. This feature works great because you don't have to wait on page loads but has a disadvantage that sharing a URL of a specific page does not work out of the box. During the development of the application, it must be kept in mind that the URL should contain the state of the website. For example; the URL postfix "*/wafer/W1025/Fbk*" tells the application three things, first that it goes about a wafer test, and that it should retrieve the data for waferID "W2025". Fbk tells the website that it should visualise the results for the Fbk test.

5.2.1 Dashboard Home Page

Figure 13 on the next page shows the home page of the dashboard. It has 4 components build upon the wishes of the domain expert. Per component is explained what they do and how they work.

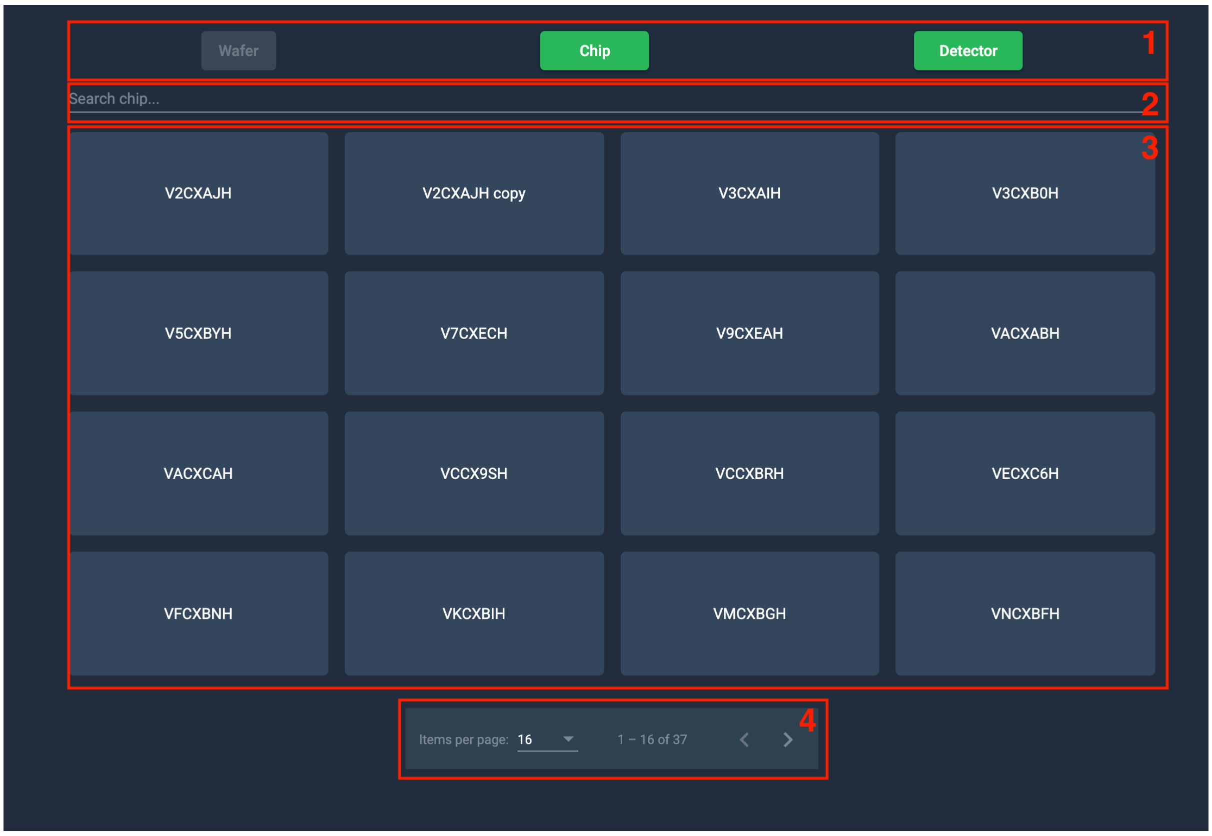


Figure 13: Home Page Dashboard

Test Phase Selection (1)
Upon entering the dashboard, there is the option to select the desired testing phase. The button is disabled if the browser URL matches the test stage. A change in the test stage updates the overview of devices (component 3) accordingly.

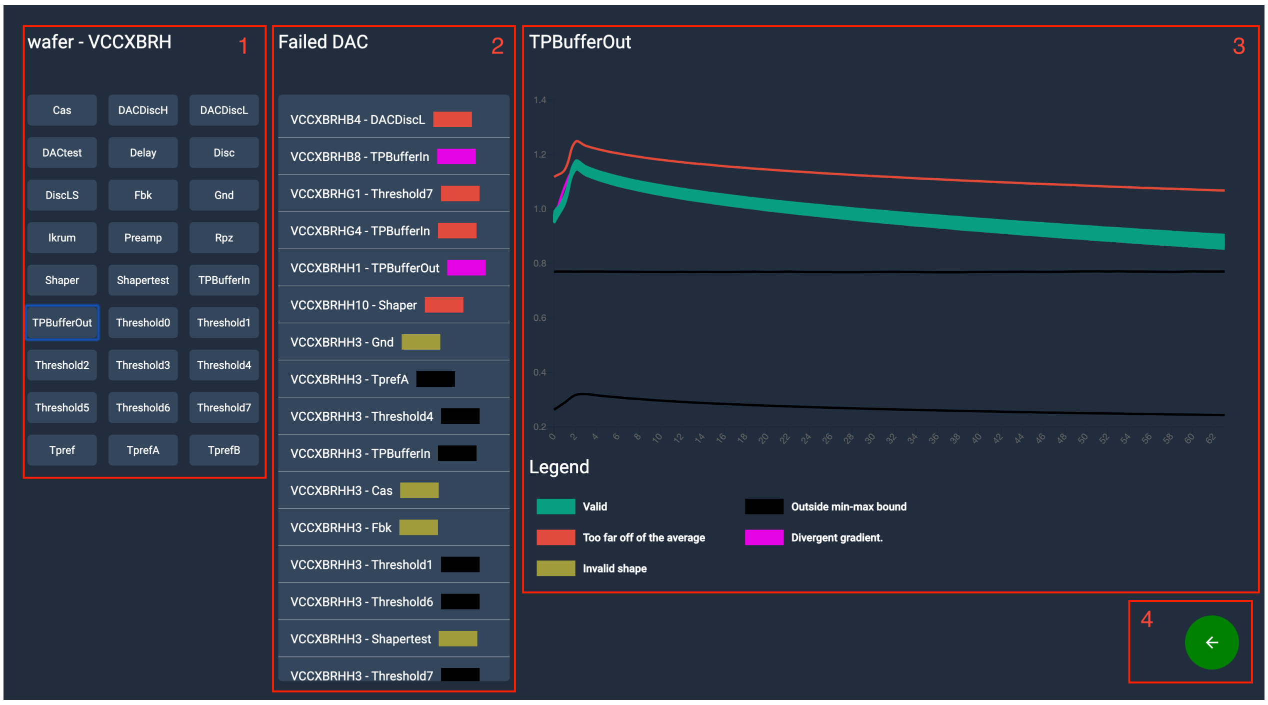
Search Field (2)
This search field searches within the backend for devices that match the query. The search field makes use of a debounce mechanism in sending the requests. This debounce mechanism prevents that a request is sent upon every keypress. Instead, after every keypress, a timer of 0.3 seconds is set. If a next keypress is detected within this time, the timer resets to 0.3. If the timer hits zero, the request is sent to the backend. This mechanism helps to prevent excessive load to the backend and database.

Tested Devices (3)
This grid shows all the tested devices, depending on the test phase; these are either wafers or chips. Upon clicking one of the gird items, a new view opens with the corresponding test results.

Pagination (4)
There is a lot of data available but to prevent the user from losing overview the webpage and backend work with pagination. Pagination helps to keep the overview and speed up load times. Instead of sending all tested devices to the dashboard, a subset of 16 devices is send. Through the pagination component, the subset size and page can be changed.

5.2.2 Dashboard Test Overview

Figure 14 shows all the test results. And visualise the labels of the model.



Failed DAC

- VCCXBRHB4 - DACDiscL 🟥
- VCCXBRHB8 - TPBufferIn 🟪
- VCCXBRHG1 - Threshold7 🟥
- VCCXBRHG4 - TPBufferIn 🟥
- VCCXBRHH1 - TPBufferOut 🟪
- VCCXBRHH10 - Shaper 🟥
- VCCXBRHH3 - Gnd 🟨
- VCCXBRHH3 - TprefA ⬛
- VCCXBRHH3 - Threshold4 ⬛
- VCCXBRHH3 - TPBufferIn ⬛
- VCCXBRHH3 - Cas 🟨
- VCCXBRHH3 - Fbk 🟨
- VCCXBRHH3 - Threshold1 ⬛
- VCCXBRHH3 - Threshold6 ⬛
- VCCXBRHH3 - Shapertest 🟨
- VCCXBRHH3 - Threshold7 ⬛

Legend

- 🟩 Valid
- 🟥 Too far off of the average
- 🟨 Invalid shape
- ⬛ Outside min-max bound
- 🟪 Divergent gradient.

Figure 14: Test Overview Page

Tests (1)
This grid shows all the 27 different DAC tests if one of these grid items is clicked the graphs in component 3 updates to show this specific test.

Failed Tests (2)
This list shows all the tests that the model disapproved. The colour code gives information on which criteria the test is disapproved. If the list item is clicked, component 3 gets updated so that the operator can view the disapproved graph in detail.

Graph (3)
Based on the input of component 1, this graph plots the test measurements. Each line gets assigned a colour based on the label that the model gave in the backend.

Home Button (4)
Upon clicking this button, the view changes back to the homepage. The button remembers the test phase setting that was selected.

5.3 Testing

The application is tested in several ways to make sure that it is reliable and works as expected.

The first test is manual, does the code fulfil the acceptance criteria of the task? By answering this question, most obvious bugs can be found. There is however a pitfall here, both the frontend and backend are written in scripting languages. Programmes written in scripting languages do not have to be compiled, which makes development fast. Still, simple programming mistakes remain unseen until that particular piece of code with the mistake is executed. Compiled languages such as Java find these mistakes before you can run the code.

The backend has additional PyTests [17] to find the tiny mistakes and an end to end test named Cypress [18] to test the whole package.

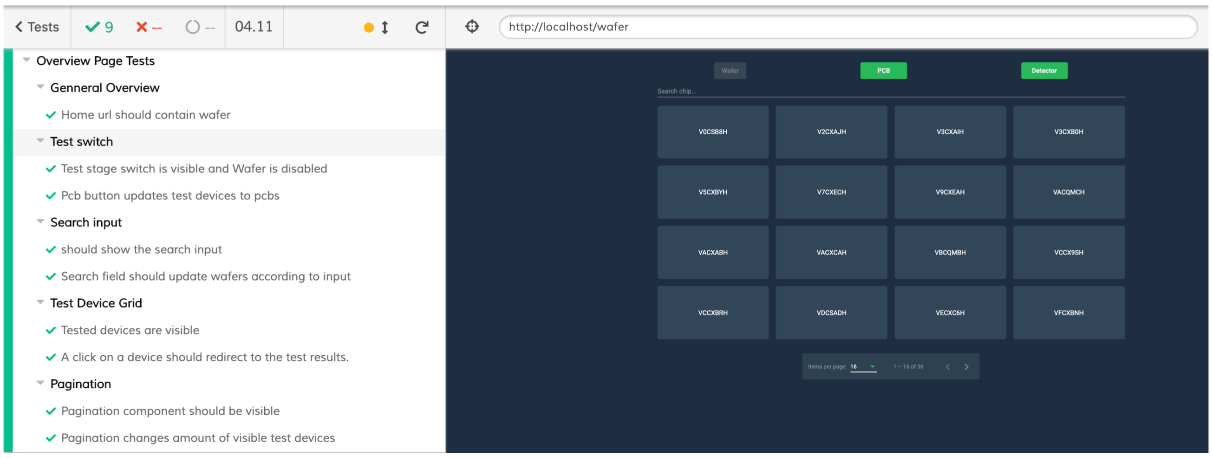
PyTest

PyTest is a lightweight testing framework which is similar to a Unit test. The backend application broadly consists of three parts; the endpoints that expose the data, the parser that converts text files into the database and the model which gives a classification to each database entry (test). The parser is built from the ground up, and it must transfer the text files into data in the database as intended. Therefore the tests cover every public method in the parser. The code that gets the label from the model is an essential step in the application. Therefore the classification part is entirely covered by tests. Lastly, the endpoints, none of the endpoints mutates the data nor implements custom functionality.
For this reason, this part is omitted in the tests because a test would mainly test the framework methods. For this application, the stability of the framework is taken for granted. Despite this, the endpoints are also tested by the end-to-end tests. Every test consists of at least one good weather test and where possible also bad weather tests.

End-to-end tests

End-to-end testing is a way to test the functionality of the application. The way these test works is similar to how a user would use the application. There are several test frameworks available most of them are based on Selenium [19]. Cypress is not; it is a new popular end-to-end testing framework. The approach of Cypress is to test as close as a human would do it. For example, if something may not work first, it will try again as most users would do. Another great feature of Cypress is that if a test fails, it saves the state of the browser DOM. Afterwards, you can inspect this state, and most of the time, it is easy to find the problem.

There are several testing strategies for an end-to-end test. For instance, do you use the real database or a predefined test database? Do you mock the backend or do you use the real backend? Etc. For this application, there is mocked as less as possible. By omitting mocks, you test the entire cycle of the application instead of only the fronted. For some applications, this may not be possible because the data can change. However, in this application data cannot be changed. The frontend application is 100% covered by tests, that means every function in the application is tested with an end-to-end test.


< Tests ✓ 9 ✗ -- ◯ -- 04.11

http://localhost/wafer

- Overview Page Tests
  - Genneral Overview
    - ✓ Home url should contain wafer
  - Test switch
    - ✓ Test stage switch is visible and Wafer is disabled
    - ✓ Pcb button updates test devices to pcbs
  - Search input
    - ✓ should show the search input
    - ✓ Search field should update wafers according to input
  - Test Device Grid
    - ✓ Tested devices are visible
    - ✓ A click on a device should redirect to the test results.
  - Pagination
    - ✓ Pagination component should be visible
    - ✓ Pagination changes amount of visible test devices

Figure 15: Overview of Cypress tests

5.3.1 Test Results

Both the end to end tests and the Pytests failed several times during development, which is excellent because it means that the tests are doing their job. It made the whole development process a lot quicker because fixing the issues right away is way quicker than realising later on that something does not work, and start to searching what component is broken.

An improvement that still can be made is dividing the frontend application further down in components, especially the dashboard test overview. At the start of development, it seemed logical to make this one component. But during the different development cycles, more and more functionality was added, which makes testing harder.

5.4 Exploring the Data

As mentioned in chapter 4.3 Project Motivation, there is test data of three stages; wafer, PCB and detector. In every stage, the chip goes through 27 tests. The names of these tests can be found on the graph labels in *Figure 16*. All the tests have a "Gate" this is a range on a specific point which the test curve should pass see *Figure 7*. Unfortunately, the test result data is not very specific. The tester only saves if a chip did or did not pass all tests instead of which test it failed.

A good start to explore the data is to look at how many DAC tests do not survive this gate. See the bar plot in *Figure 16* below. On average for each DAC, about 3 % fails. The tests *Cas* and *Fbk* are clear outliers here, which can have two reasons. One is that the gate is to narrow and that too many proper curves are marked as wrong. Another explanation can be that *Fbk* and *Cas* tests are more fragile and thus are more likely to fail.

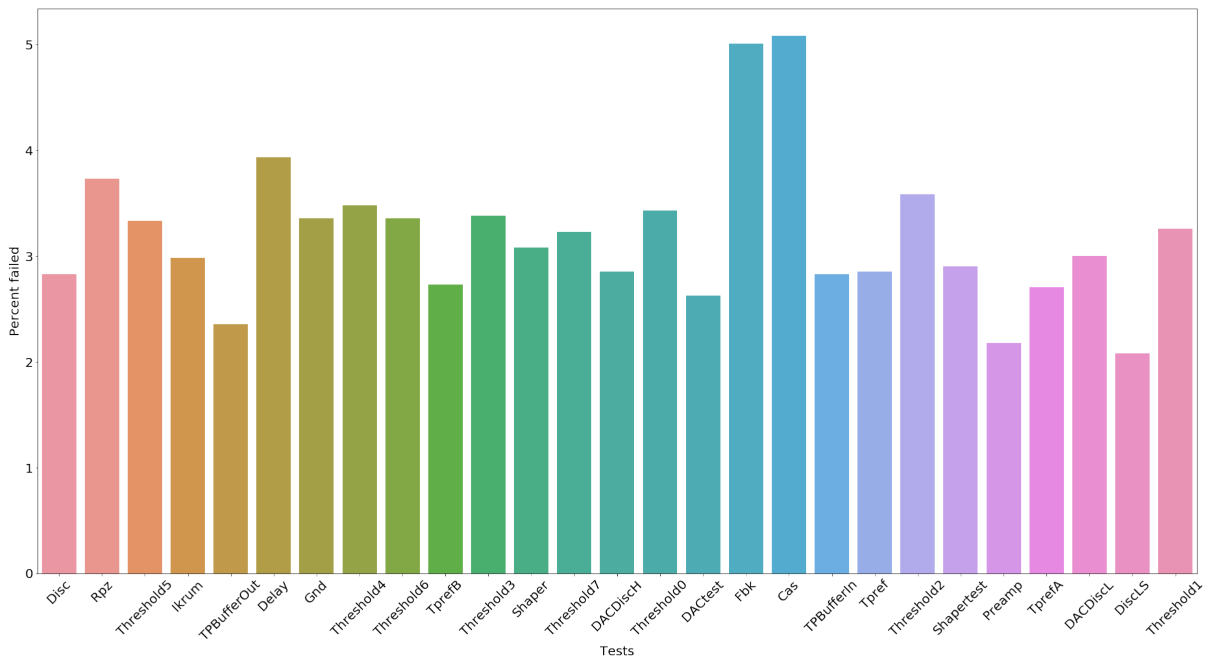
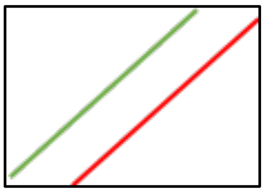


Figure 16: Percentage Failed Per Test

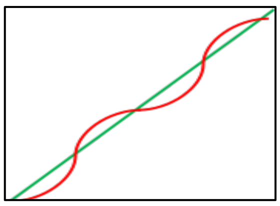
5.4.1 Deviating patterns

As mentioned before many test curves pass the gate filter, but are still defective. After consultation with the tester, it became clear that he looked for three kinds of patterns. He explained that those three patterns do not belong to a DAC curve. However, there is no hard proof that a curve with one of these patterns always results in a bad chip.



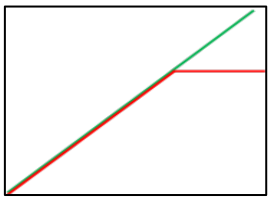
The first pattern is a curve that has a so-called offset. The curve has a good shape, but it has an offset in the vertical direction.

Figure 17: DAC curve with on offset in the vertical direction



The second pattern is an oscillation in the curve. This oscillation should never happen in any DAC curve because it should always descend or ascend not both.

Figure 18: DAC curve with oscillation



The last pattern is curve that suddenly deviates from the average curves.

Figure 19: DAC curve with a sudden deviation

Chapter 6 *Modelling* will explain in-depth how these patterns can be detected.

5.4.2 Test data differences

All tests at the wafer level take place at an external company. The other two test phases take place at Malvern Panalytical. Because there are two parties involved in the process, this could result in differences in measurement accuracy. See the graph in *Figure 20* below.

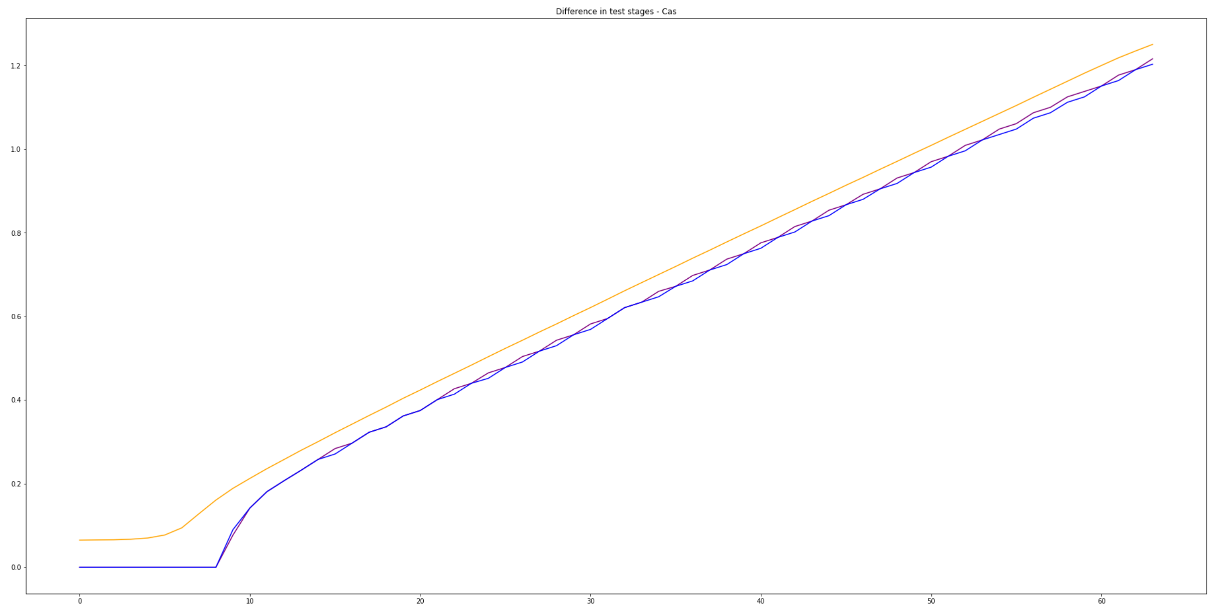


Figure 20: Measurement difference in different testing stages

All three curves in *Figure 20* come from the same DAC on the same chip. The only difference is the test phase of the measurement. The yellow line is the measurement taken on wafer-level at the external company. The purple line is the measurement on the PCB and, the blue line is the measurement taken on the sensor module. In theory, the lines should be all the same. However, there is a clear difference between the wafer measurement and the PCB and sensor module measurement. This difference is highly likely caused by the difference in the measurement accuracy of the testing devices used. These differences must be kept in mind if PCB and sensor module data are used; it needs preprocessing before it is fed into a model.

6 MODELLING

To answer the question "*How can faulty chips be classified?*" it is essential to investigate different modelling techniques to find out which results in the best fitting model.

6.1 MODELLING TECHNIQUES

Malvern Panalytical wants to expand her knowledge in the field of machine learning. Therefore they would like to solve this problem with a machine learning algorithm. Supervised machine learning algorithms work with a labelled dataset, which means that the model can learn from already existing answers.

However, the problem is that this knowledge is missing in the dataset. The lack of information has to do with the fact that it is complicated to test a chip when it is still on the wafer. The test data from the wafer test only provides insight into how current flows through the DACs and not whether a DAC is working or not.

Later in the second test phase, it is possible to test whether a chip works or not because the chip is cut from the wafer and placed on a PCB. Firmware is available on the PCB to test full functionality. Unfortunately, the data only tells if the chip works or not; this does not tell if a DAC is working or not. It is possible that the DAC in the chip works perfectly but that the problem is with another component in the chip. Therefore, the label whether the chip works or not is not specific enough to train a machine learning model solely based on DAC curves. Based on the fact that the labels are not sufficient enough, research has been performed to find alternative methods.

6.1.1 Clustering

A side branch in machine learning is unsupervised learning. Unsupervised models do not need labels. An example of unsupervised learning is clustering. Clustering works by looking for clusters based on a specific set of properties. For example, imagine a model that has to cluster apples and mandarins. The model can cluster them based on size or colour, without knowing beforehand what an apple is and what a mandarin.

This principle can also be applied to the DAC curves with the hypothesis that a wrong curve looks different from one that works properly. Based on the experience of the tester, false DAC curves are typified by patterns described in *5.4.1 Deviating patterns.*

Several algorithms can be used to apply clustering to a dataset. *Figure 21* shows an overview of a sample of different algorithms. Four algorithms are chosen to reduce the research scope. These algorithms are picked based on the behaviour in the red-marked boxes in *Figure 21*. These patterns in the red boxes are considered to look the closest to the DAC curves. The chosen algorithms are:

- GaussianMixture [20]
- MeanShift [21]
- DBSCAN [22]
- OPTICS [23]

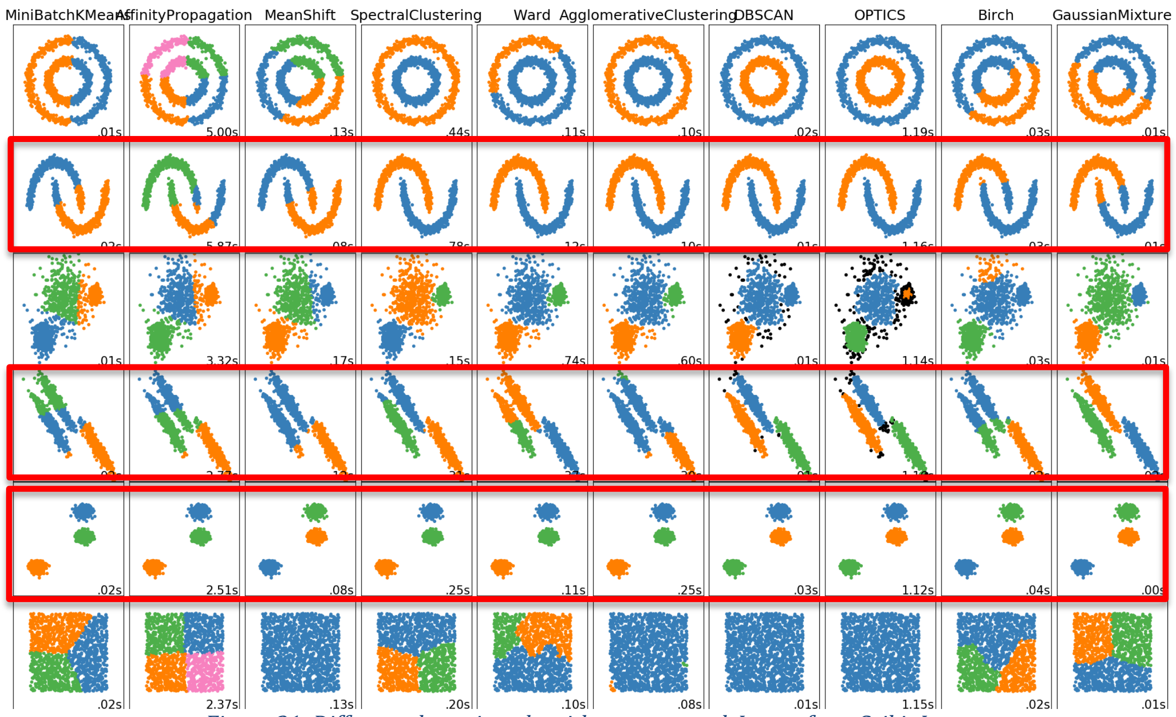


Figure 21: Different clustering algorithms compared. Image from Scikit Learn

All four algorithms are trained separately on all 27 different DAC curves. The test dataset for every DAC curve consists of 1500 samples. For every algorithm, about an hour is spent on optimising the parameters. These parameters are there to tweak the algorithm to better suit a specific problem.

After visualising the plots, it became clear that GuassianMixture is the only algorithm capable of clustering a substantial number of deviating curves. Both MeanShift and OPTICS were not capable of detecting any abnormal behaviour. DBSCAN did create two clusters in one of the plots; however, this was not an abnormal curve. Based on the results in the plots, GuassianMixture is chosen for further research. A selection of the plots is available in *Appendix 4: Cluster Algorithm Plots*. More plots are available in an external Jupyter Notebook file.

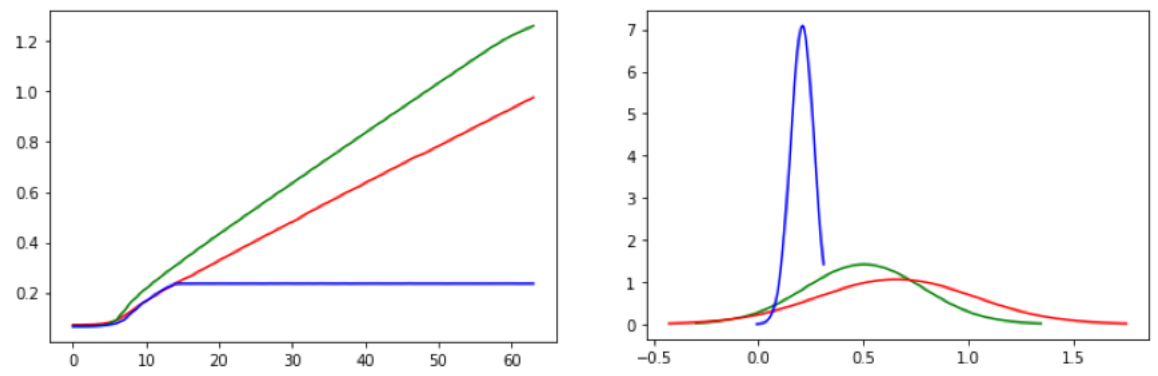


Figure 22: Three DAC curves with a plot of their Gaussian distribution

First, how does Gaussian Mixture work? Gaussian Mixture clusters the curves based on the gaussian distribution (sometimes named normal distribution) as property. *Figure 22* is a behind the scene representation how the Gaussian Mixture algorithm summarised works. On the left is a plot of three different DAC curves and on the right is a plot of their Gaussian distribution. An elaborate explanation can be found on Towards Datascience: Gaussian Mixture Models Explained [24].

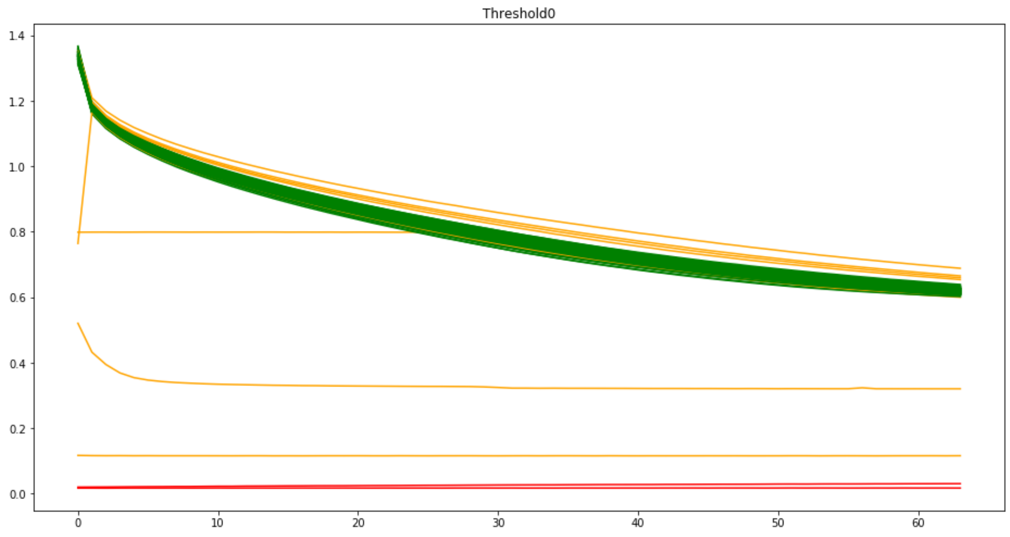


Figure 23: Example of a good result with Gaussian Mixture

Figure 23 is a plot clustered by Gaussian Mixture of one of the 27 different DAC curves. In this plot, the algorithm does a perfect job of finding deviating patterns.

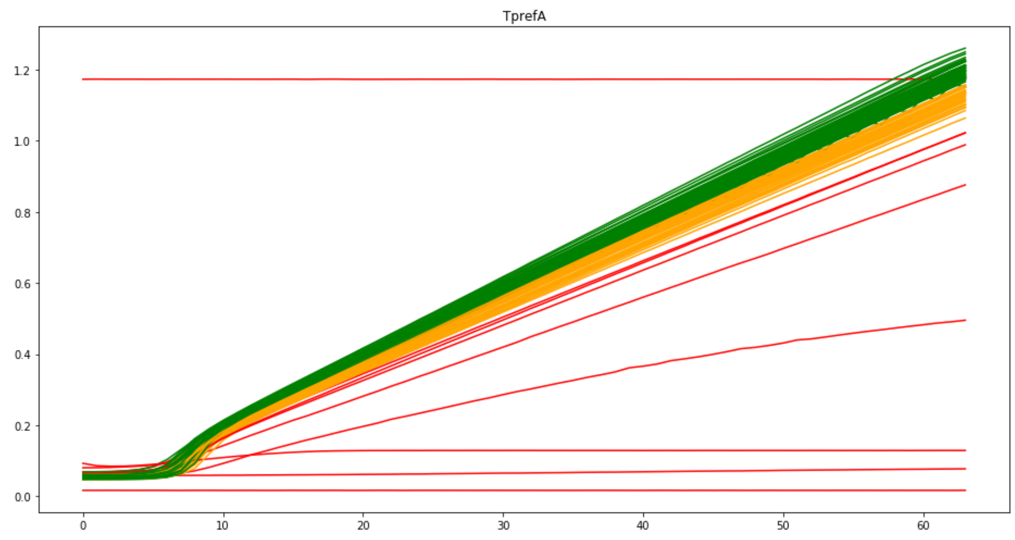


Figure 24: Example of a less optimal result with Gaussian Mixture

Figure 24 is the Gaussian Mixture cluster plot from a different DAC. The spread of good curves is with this DAC bigger. This broad spread confuses the algorithm, and it starts clustering within good curves. Furthermore, it seems that this algorithm has a particular problem with detecting oscillating curves, as shown in *Figure 18*. This behaviour makes the algorithm not usable for detecting defect DAC curves.

6.1.2 Statistical Model

An alternative can be to combine different statistical filters, to filter on a specific behaviour. This method has the advantage that it does not require labels. However, there need to bee consensus about what a failing DAC curve looks like. Another benefit of using statistics is that it is easier to explain than a machine learning model.

The goal of this model is to emulate the behaviour of the tester. The model should be capable of pointing out the same curves as the human tester would do manually. The patterns that the tester looks for are explained in *5.4.1 Deviating patterns.* These patterns are translated into four filters. The filters are made by statistical knowledge, trial and error and through interviewing a student who studies mathematics

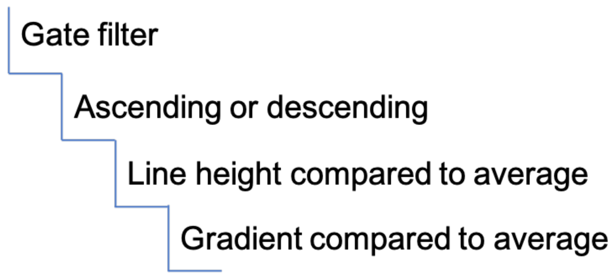


Figure 25: Statistical Model Filters

The model starts with the already existing gate filter. This gate filter has an upper and a lower bound at a specific point in the curve for every DAC test. If the curve does not pass this filter, it is labelled as "Did not pass gate".

The second filter checks the curves for kinks and flat lines which means a line that starts ascending can never go descending or vice versa. There is a possibility to skip the first few measurements since certain DACs have a threshold which means that they start with a flat line.

```
line = line[skip:]
increase = True
for i in range(0, len(line)):
    if i + 2 < len(line):
        if (line[i] < line[i]) == (line[i] < line[i + 2]):
            increase = True
        else:
            return False
return increase
```

Figure 26: Increase/Decrease Filter

The first two filters together filter out all rough outliers. Still, not all curves are good at this point. The third filter takes during training the average of the training dataset every point in the curve. These averages result in an average curve. Every line that passes this filter should lay within an *x* amount of standard deviations above or below the average line. The advantage of using a standard deviation over a fixed number is that it grows or shrink according to the spread of the curves. If the line deviates, the model also returns the percentage that it deviates from the bound.

The last filter is for all exceptional lines that are faulty but still passes the above filters. An example of such a line is an oscillating curve that at every point, ascends or descends. This filter works the same as the third filter. Instead, it takes the average gradient of each point in the curve. If the curve its gradients lays within an *x* amount of standard deviations above or below the average gradient it is approved. Also, for this filter, the model returns the percentage that it deviates from the bound.

The *x* in both filers are independently configurable. The amount of standard deviation tolerance is determined by the point of diminishing returns, which is tested on a test dataset of 1962 curves and a validation dataset of 763 curves.

This model is a definite improvement over the clustering model. See *Figure 27* and *Figure 28* for the same set of curves but now with the statistical model labelling the DACs. It is not only more accurate, but it also gives an explaining label. An extra advantage is that this can be retrained on other DAC test of other machines which makes it reusable.

Label	Description	Colour
0	Valid	🟩
2	Did not pass gate	⬛
3	Neither ascending nor descending	🟨
4	Average too far off	🟥
5	Gradient too far off	🟪

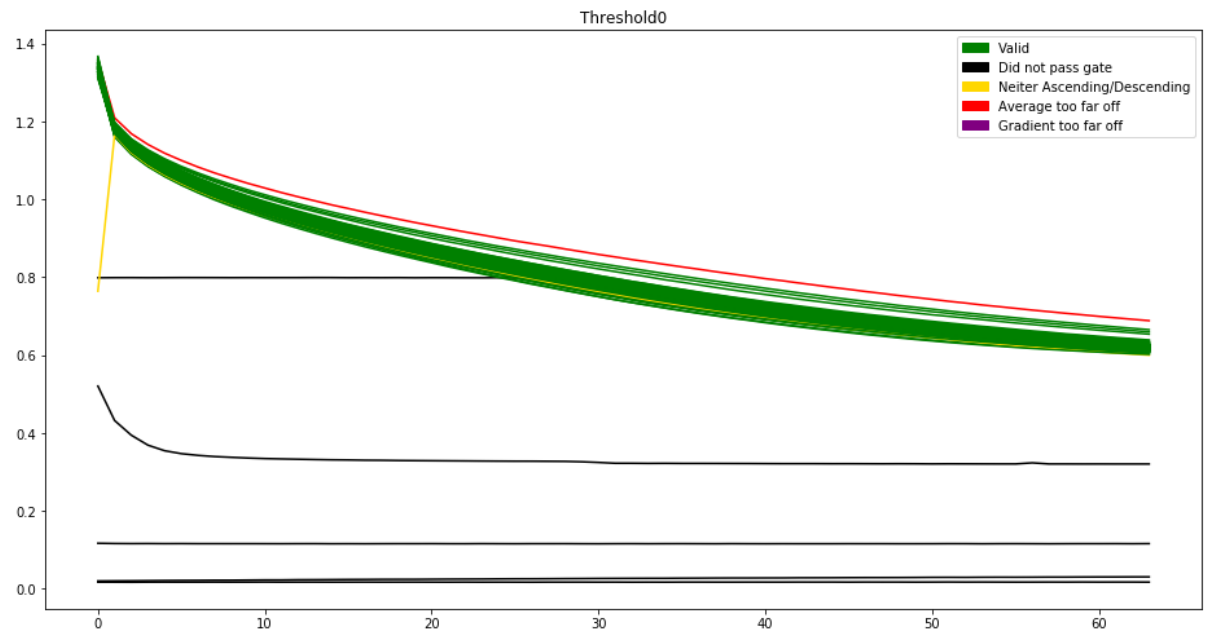


Figure 27: Threshold DAC labelled by the statistical model

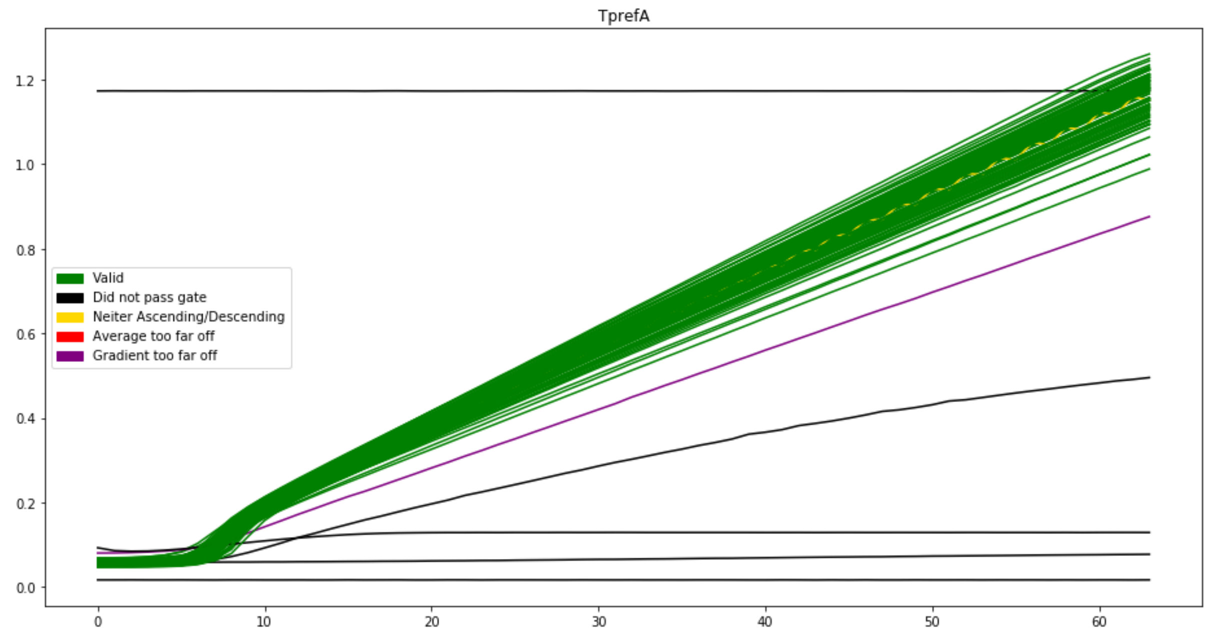


Figure 28: TprefA DAC labelled by the statistical model

7 Evaluation

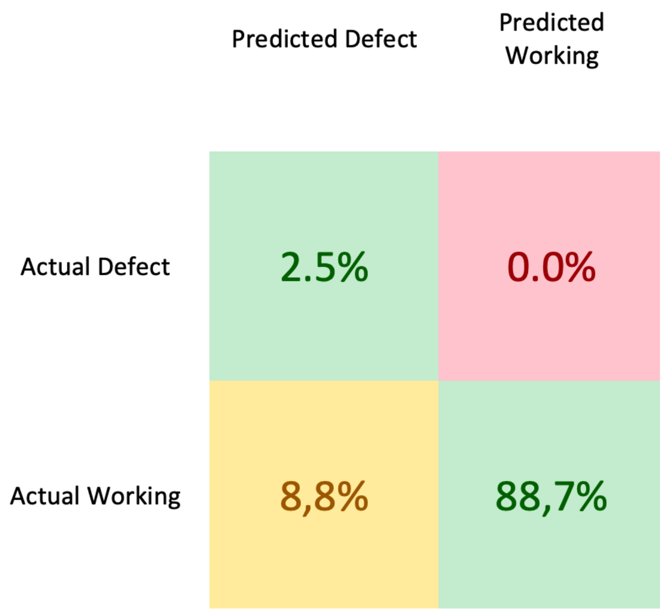
In chapter *6 Modelling* was made clear that the Statistical Modelling technique will be applied for this project. In the evaluation phase, the results are validated against the goals made in the Business Understanding phase. Also, a reflection on the process is made, and the next steps are determined.

7.1 Model Validation

Validating the quality of the statistical model is essential. If the model approves a defective chip, this could cause reputational damage to Malvern Panalytical and a loss of money. Every successive production phase adds much value to the chip.

The model is validated in two ways, first by visualising the results and looking together with the tester. The hypothesis was: "The the tester would make the same decisions as the model would". The result of this test was positive; the model indeed disapproved wrong curves.

A blind test is performed to remove any biases the tester may have had during the first test. The model labels 18 wafers which are 1962 chips and is validated against 18 wafers labelled by the domain expert. By comparing the results of the model and the domain expert, a confusion matrix could be made. This matrix is made assuming that the domain expert makes no errors.



	Predicted Defect	Predicted Working
Actual Defect	2.5%	0.0%
Actual Working	8,8%	88,7%

Figure 29: Confusion Matrix

The confusion matrix in *Figure 29* works as follows. If a chip is rejected by the tester and the model classifies the chip as working, the chip ends up in the upper-right corner. Conversely, if the model disapproves the chip and the tester approves the chip, the chip ends up in the lower-left corner.

At first glance, 8.8% of wrongly rejected chips seem inadequate, which would involve that 8.8% more chips are thrown away unnecessarily. However, this number is more nuanced; it could be that the tester made mistakes. The labels of the tester cannot be validated because the test data is at wafer-level. Another reason can be that the label of the tester is based on other tests, while the model only looks at the DAC curves. For instance, if a chip is rejected based on a power test and the DAC tests are fine, it will end up in the wrongly rejected box.

The number in the upper right box is 0%. This number is a good result which means that you could make the model work independently without running the risk that it will pass wrong chips. Even if this would be slightly above zero, it doesn't have to be a problem, as long as the costs remain below the saved working time. If there are wrongly approved chips, they will be detected in the PCB tests.

Besides the wafer-level labels of the tester, there is also a dataset of labels from the PCB test phase. The advantage of these labels is that the chips at this stage are measured. So there is no human randomness factor in the assessment of a chip works or not.

With these labels, it is possible to investigate the false negatives in Figure 29. The labels show that the model could prevent 10% of the defect labelled chips in phase 2. This number is calculated by counting the disapproved chips from the model in phase 1 and the measured as defect chips in phase 2, divided by the total amount of broken chips in phase 2. These numbers are based on a set of 31 wafers.

On the other hand, it is also possible to calculate the unfairly labelled as defect chips by the model. These false positives are around 7%. Again the number is likely lower in reality because the working / not working label is based on all test and not solely on the DACs. Also preventing a defect chip on the wafer to become a PCB safe 10 times more then is lost with throwing a wrongly disapproved chip away. In practice, this means there is already a cost reduction if nine chips are wrongly thrown away, and there is 1 correctly disapproved.

Another thing to measure the performance of the model is to see if a particular filter has more false positives then other filters and if a chip closer to the average is more likely to be a correct classification then a chip further away.

The hypothesis is that a curve that is far outside the average has a higher chance of actually being wrong than a curve that is closer to the average. In *Figure 30,* the distance to the average is compared to whether it was the right classification or a wrong one. Label 4 represents the distance to the average height and label 5, the distance to the average gradient. The graph does not show a clear correlation between distance and the right classification.

Another question is whether certain filters are more error-prone than others. Therefore, in figure *Figure 31* also labels 2 (gate filter), and 3 (ascending/descending) are added. These labels are plotted against the number of incorrect and correct classifications. This graph does not give a clear conclusion. At most can be concluded that label 5 is more wrong than right, label 4 is near-random and label 2 and 3 are most of the times right.

It remains challenging to draw a proper conclusion from both graphs. This is difficult because the numbers will most likely look different in reality due to the earlier mentioned problem that the labels from the PCB test are not specific enough. Take the following example. A chip has 27 DACs. And the model will label in the extreme case all 27 DACs as faulty, while in reality only one DAC is broken. In that case, 26 DACs wrongly get the conclusion that they are correctly classified.

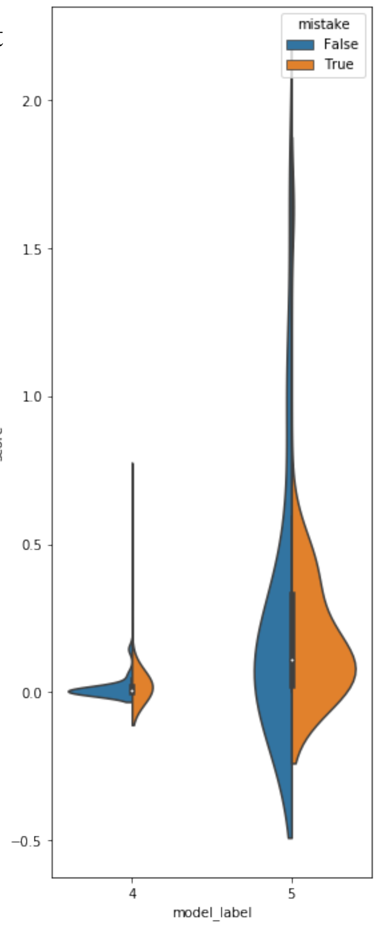


Figure 30: Relation Label to Prediction

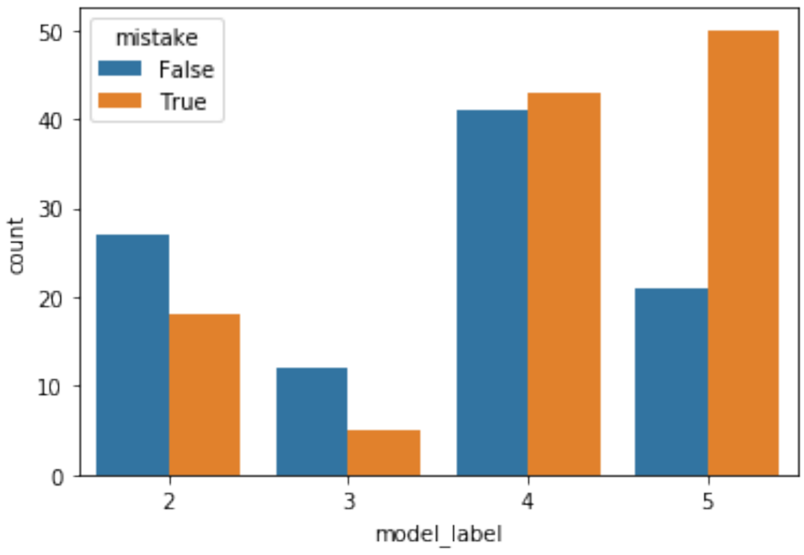


Figure 31: Labels compared to success/mistake

8 Deployment

In many software projects, the deployment phase is hard. Thing work on local machines of developers but on others, it doesn't. Also, the problem of specific components that require preinstalled software can be a hassle. This software runs inside Docker [25] container to solve these compatibility problems.

8.1 Docker

First, what is Docker and which problems does it solve. The application is developed on a macOS with a specific python version and mongoDB version installed. If someone else wants to run the software, he or she likely has a different operating system and python versions may also differ. The differences in the operating environment may cause unwanted behaviour or may result that the application does not work at all. In the era before Docker, a virtual machine could be a solution. With a virtual machine, you can make sure that both environments are precisely the same. But the downside of a virtual machine is that it occupies a lot of memory space and it is difficult to scale. Docker solves these issues by eliminating the guest OS. For reference compare *Figure 32 and Figure 33*.

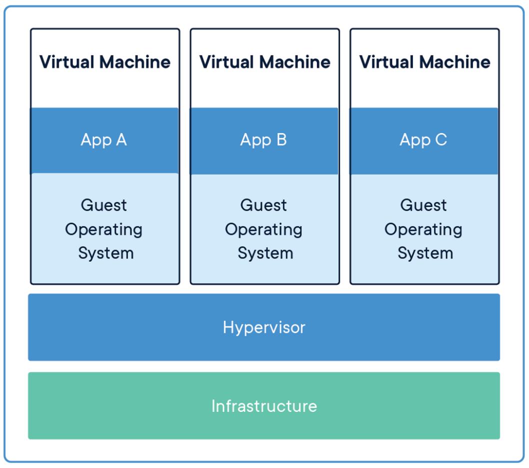


Figure 32: Example Virtual Machine – from Docker

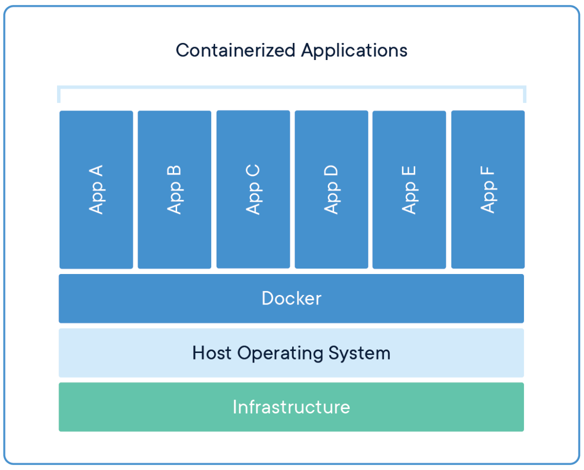


Figure 33: Example Docker Container – from Docker

An additional advantage of Docker is that the user who runs the application does not have to configure and install additional software besides the docker deamon. All the required packages and software with the right versions are automatically installed within the docker container.

This application has a docker-compose file which contains all the necessary configuration to run the entire pipeline. The only thing the end-user has to change is its data directory. For a complete manual to run the software, consult Appendix 2: Insatallation Manual

9 Discussion and Advice

The data is now easily accessible through a web interface and a database. The accessibility of the data makes other data science-related studies on this data quicker. However, the testing process can still be improved. In the current situation, the labels coming from the manual review of DAC tests are not detailed enough. There is a label that the chip is disapproved but not why it is disapproved. This label can also indicate that the chip failed on tests other than the DAC test. Further specifying this label can help to assess this model better.

Another point for improvement could be to standardise the manual test process. The current test process depends on a person's opinion. If this person leaves the company and is replaced by someone else, the labels will most likely change slightly based on the test results. This randomness can also be seen in figures *Figure 31*; label 4 is almost half the time right and the other half the time wrong. When I show the wrong predictions of the model to the tester, he tells me that he would also reject them, but that they were probably overlooked. On the other hand, it can be argued that the test process is not good because the chip did work in the end. To prevent this, I advise doing an extensive study on the effect of a broken chip on DAC curves. This study has never been done so far due to the cost of placing a chip on a PCB. This study could result in a more optimised model which leads to a further increase in cost savings because fewer chips need to be thrown away incorrectly.

Altogether the model reaches the business goal of Malvern Panalytical. It is capable of identifying faulty chips. This model can work standalone instead of a domain expert since it is proven that it does not work worse. And thus can safe a lot of labour time. However, the model can also work together with a human operator to form a perfect synergy. Since identifying the faulty curves is the most time-consuming process. The human operator can then make the final decision on approving or disapproving the chips.

While the current model is capable of classifying chips, it would be even better to predict failure. It can happen that at a particular test phase, a chip look good, but in a later test phase, it fails. With the current dataset of Malvern Panalytical it is likely possible to predict failure.

It is also interesting to build a model based on all test data. Training such a model can be done with the current because a good/faulty label will suffice if all data is used. A model that is trained based on all test results is likely to be much more accurate than the current model.

10 Conclusion

This project was carried out for Malvern Panalytical to answer the question: "How can Malvern Panalytical automatically label faulty DACs with historical measurement data? Several components have been researched and developed to answer this question.

The first part of the research focused on interpretation and investigation of the data. Data Science is still in its infancy at Malvern Panalytical. The data is readily available but difficult to process because it is stored in text files. That is why the first step was to build a tool that collects the data and puts it inside a database. This tool ensures that future data research can be more easily done because popular data science tools such as Matlab and Pandas have good integration with MongoDB and cannot handle text files.

Furthermore, it turned out that there is much manual work involved with the current test methods. It became clear that the data had to be put in excel files, and graphs had to be generated manually. This manual process takes much time. Also detecting deviating DAC curves was difficult because the graphs do not give any help. Furthermore, it became clear from the data that averages per wafer could vary considerably and therefore a chip with the same value on wafer A was approved and not on wafer B because it could simply not be seen with the bare eye. This problem led to the construction of a tool that collects the data automatically and plots it in a clear way. It uses the results of the Statistical Model to give the tester an indication that a line is deviating. It can then later be decided whether to rely on the results of the model blindly or to let the tester make the final decision if a chip is working or not. In any case, the tool reduces the number of randomness in the made decisions and safes manual labour.

The model itself is built to emulate the decisions made by the tester. The disadvantage of this is that the decisions of the tester are based on intuition, and not on research. Therefore, my advice is to do extensive research on the influence of a broken DAC on the DAC curves. With this information, a better model can be developed. The lack of detailed labels also makes it difficult to give a well-founded assessment of the results of the model. Nevertheless, in most cases, the model performs better than the human tester, but the most significant advantage is the time gain. It is no longer necessary to manually identify deviations chip by chip. This process can now work fully automatic.

All in all, it is possible to label the wrong DACs automatically. However, an even better result can be achieved by gaining more insight into the DAC curves. That is why my advice is to put several chips marked as "Broken" on PCB after all, to verify whether they are broken. It is also advisable to give a more detailed description with the current labels instead of working or not working. The more elaborate labels will make assessing the performance of the model more reliable.

11 BIBLIOGRAPHY

[1] S. C, "The CRISP-DM model: the new blueprint for data mining,," 2000.

[2] A. Hadoop. [Online]. Available: https://hadoop.apache.org/.

[3] S. Collet, "Creativ Data," [Online]. Available: https://creativedata.atlassian.net/wiki/spaces/SAP/pages/61177860/Python+-+Read+Write+files+from+HDFS.

[4] S. Balint, "The Small FIles Problem," [Online]. Available: https://blog.cloudera.com/the-small-files-problem/.

[5] MySQL. [Online]. Available: https://www.mysql.com/.

[6] MatLab, "MySQL with Matlab," [Online]. Available: https://www.mathworks.com/help/database/ug/mysql-jdbc-windows.html.

[7] MySQL, "MySQL pyton connector," [Online]. Available: https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html.

[8] MongoDB. [Online]. Available: https://www.mongodb.com/.

[9] W. Schools, "MOngoDB Insert Document," [Online]. Available: https://www.w3schools.com/python/python_mongodb_insert.asp.

[10] S. Bear, "SOAP vs REST," [Online]. Available: https://smartbear.com/blog/test-and-monitor/soap-vs-rest-whats-the-difference/.

[11] Django. [Online]. Available: https://www.djangoproject.com/.

[12] Flask. [Online]. Available: https://flask.palletsprojects.com/.

[13] M. Solutions, "Fask vs Django," [Online]. Available: http://www.mindfiresolutions.com/blog/2018/05/flask-vs-django/#:~:text=Django%20is%20a%20full%2Dstack%20web%20framework%2C%20whereas%20Flask%20is,by%20providing%20the%20required%20functionality..

[14] D. S. 2019. [Online]. Available: https://insights.stackoverflow.com/survey/2019#technology.

[15] A. Material. [Online]. Available: https://material.angular.io/.

[16] SASS. [Online]. Available: https://sass-lang.com/.

[17] PyTest. [Online]. Available: https://docs.pytest.org/en/stable/.

[18] Cypress. [Online]. Available: https://www.cypress.io/.

[19] Selenium, "Selenium Web Driver," [Online]. Available: https://www.selenium.dev/.

[20] S. Learn, "Guassian Mixture," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html.

[21] S. Learn, "Mean Shift," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html.

[22] S. Learn, "Mean Shift," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html.

[23] S. Learn, "OPTICS," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html.

[24] O. C. Carrasco, "Towards Datascience," [Online]. Available: https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95.

[25] Docker. [Online]. Available: https://www.docker.com/.

[26] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining.

[27] HBO-i, "Methoden Toolkit HBO-i," [Online]. Available: https://onderzoek.hbo-i.nl/index.php/Methoden_Toolkit_HBO-i.

[28] M. S. Brown, "Dummies," [Online]. Available: https://www.dummies.com/programming/big-data/phase-1-of-the-crisp-dm-process-model-business-understanding/.

[29] C. .NET. [Online]. Available: https://dotnet.microsoft.com/apps/aspnet.

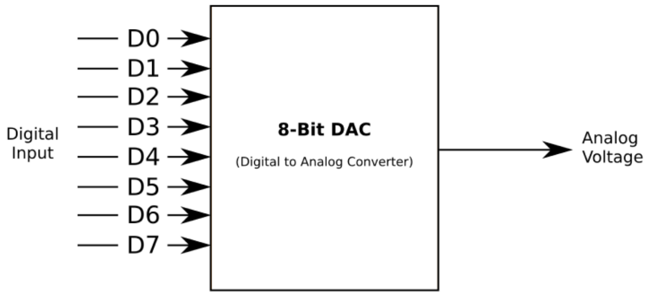
12 APPENDICES

APPENDIX 1: WHAT IS A DAC

Digital to Analog Converter

It is impossible to build an accurate model that can predict failures of DAC (Digital to Analog Converter), without knowing anything about this component. As the names says a DAC converts a digitals signal to an analog signal. The digital signal represents a sequence of discrete numbers. An analog signal represents continuous values. The number of inputs of a DAC is always 2^n, where n is the number of inputs. The DAC has one analog output.

For the conversion every digital signal is mapped to a voltage range. Where 0V being the minimum and the max voltage represents the max digital input.



Why do you need a DAC?

Everything around us is analog. Computers are digital. If you produce a result with a computer and want to make it understandable to the outside world, you have to convert it to an Analog signal. For example,

a song created on a keyboard is saved binary on a computer. To listen to this song the binary values have to be converted to an analog signal that we can hear as music.

APPENDIX 2: INSATALLATION MANUAL

prerequisites

- Windows 10 Pro/Enterprise/Education or a Unix based system
- The Docker-compose.yml from the git repository

Data Directory

Make sure that the data directory contains the following folders:

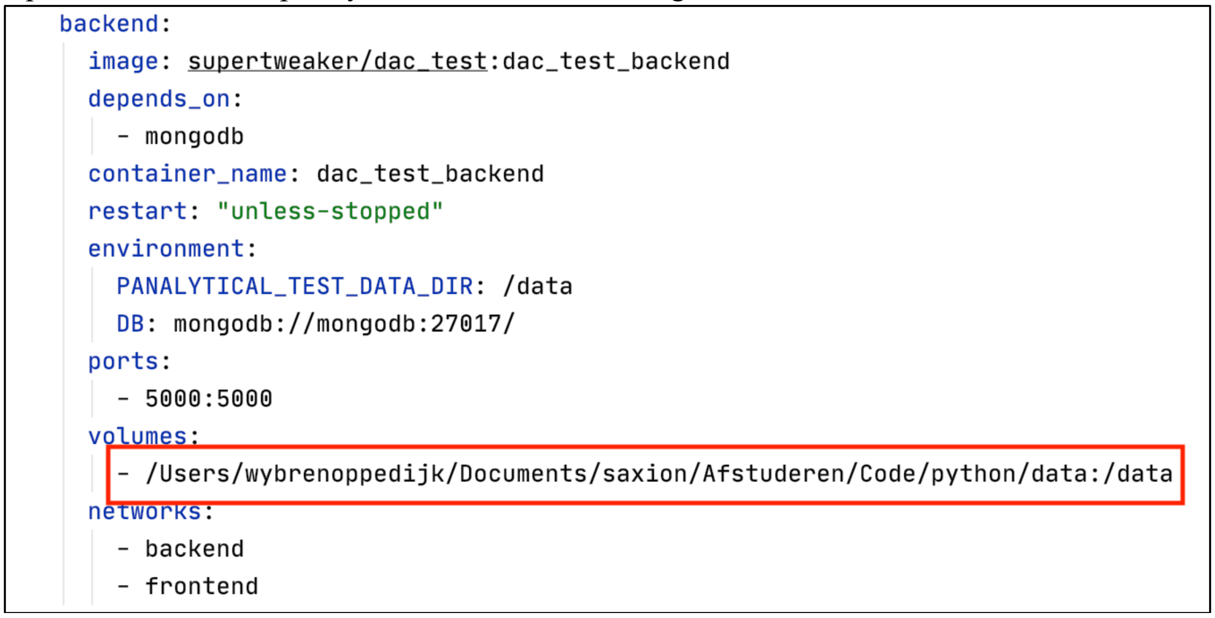
- wafer
- pcb
- detector

Docker

Create an account on https://hub.docker.com/. Make sure that you get access to the private repository "supertweaker/dac_test".

Install docker from: https://www.docker.com/. Make sure during the installation that docker is added to the PATH so that you can use it in the terminal.

Open the docker-compose.yml and edit the following line:



```
backend:
  image: supertweaker/dac_test:dac_test_backend
  depends_on:
    - mongodb
  container_name: dac_test_backend
  restart: "unless-stopped"
  environment:
    PANALYTICAL_TEST_DATA_DIR: /data
    DB: mongodb://mongodb:27017/
  ports:
    - 5000:5000
  volumes:
    - /Users/wybrenoppedijk/Documents/saxion/Afstuderen/Code/python/data:/data
  networks:
    - backend
    - frontend
```

Change the directory to the data directory on your machine, make sure that you leave the "*:/data*" at the end.

Open the terminal and make sure that the docker deamon is running. *cd* to the directory that contains the docker-compose.yml file. Inside this directory execute the "*docker-compose up*" command. This start downloading the correct images, after a couple of minutes the application is accessible on http://localhost. Dependent on the amount of data and the amount of CPU cores the initial run can take a while. This can be more then 30 minutes.

APPENDIX 3: API DOCUMENTATION

Functionality	Returns list of chip id of a particular stage
URL	`/api/stage/<string:stage>/chips?skip=0&limit=1`
Example Return	

```
[
    [
        {
            "totalCount": 265,
            "chipIDs": [
                "W1060A7"
            ]
        }
    ]
]
```

Functionality	Returns list of wafersIDs
URL	`/api/stage/wafer/wafers?skip=0&limit=1`
Example Return	

```
[
    [
        {
            "totalCount": 36,
            "wafers": [
                "V2CXAJH"
            ]
        }
    ]
]
```

Functionality	Returns list of test values.
URL	`/api/stage/<string:stage>/test_name/<string:test_name>`
Example Return	

```
[
    {
        "_id": {
            "$oid": "5ea030d0a0fd9a3d96896460"
        },
        "chipID": "V2CXAJHD2",
        "label": 2,
        "testName": "Cas",
        "value": [
            0.1644714018895878,
            0.1650207496227718,
            0.166057019210369,
        ],
        "wafer": "V2CXAJH"
    },
    …
]
```

Functionality	Returns all tests of test stage object.
URL	`/api/stage/<string:stage>/test_names`

Example Return

```
[
    [
        {
            "_id": "null",
            "testNames": [
                "Cas",
                "DACDiscH",
                "DACDiscL",
                "DACtest",
                "Delay",
                "Disc",
                "DiscLS",
                "Fbk",
                "Gnd",
                "Ikrum",
                "Preamp",
                "Rpz",
                "Shaper",
                "Shapertest",
                "TPBufferIn",
                "TPBufferOut",
                "Threshold0",
                "Threshold1",
                "Threshold2",
                "Threshold3",
                "Threshold4",
                "Threshold5",
                "Threshold6",
                "Threshold7",
                "Tpref",
                "TprefA",
                "TprefB"
            ]
        }
    ]
]
```

Functionality	Returns all tests of test stage object.
URL	`/api/stage/<string:stage>/search/<string:query>`

Example Return

```
[
    {
        "totalCount": 1,
        "wafers": [
            "W1221",
        ]
    }
]
```

Functionality	Returns all tests of test stage object.
URL	`/api/stage/<string:stage>/search/<string:query>`
Example Return	

```
[
    {
        "totalCount": 1,
        "wafers": [
            "W1221",
        ]
    }
]
```

Functionality	Returns all tests of test stage object.
URL	`/api/stage/<string:stage>/<string:id>/failed_test_names`
Example Return	

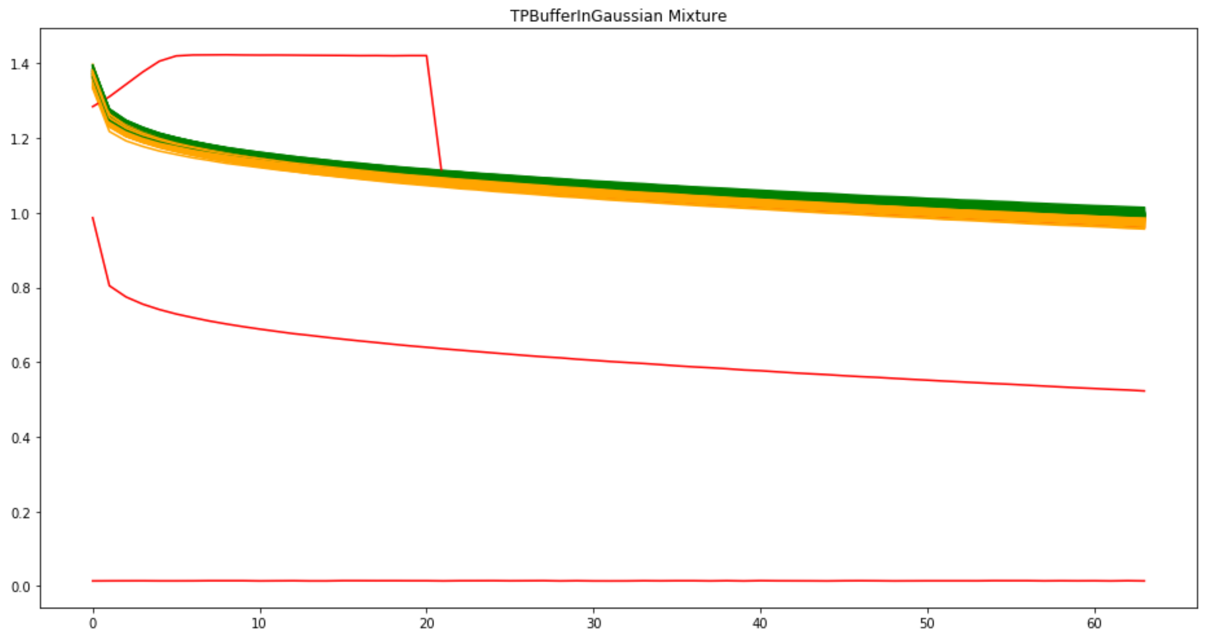
```
[
    [
        {
            "_id": "null",
            "testNames": [
                "TPBufferOut",
                "Threshold3",
                "Threshold6"
            ]
        }
    ]
]
```

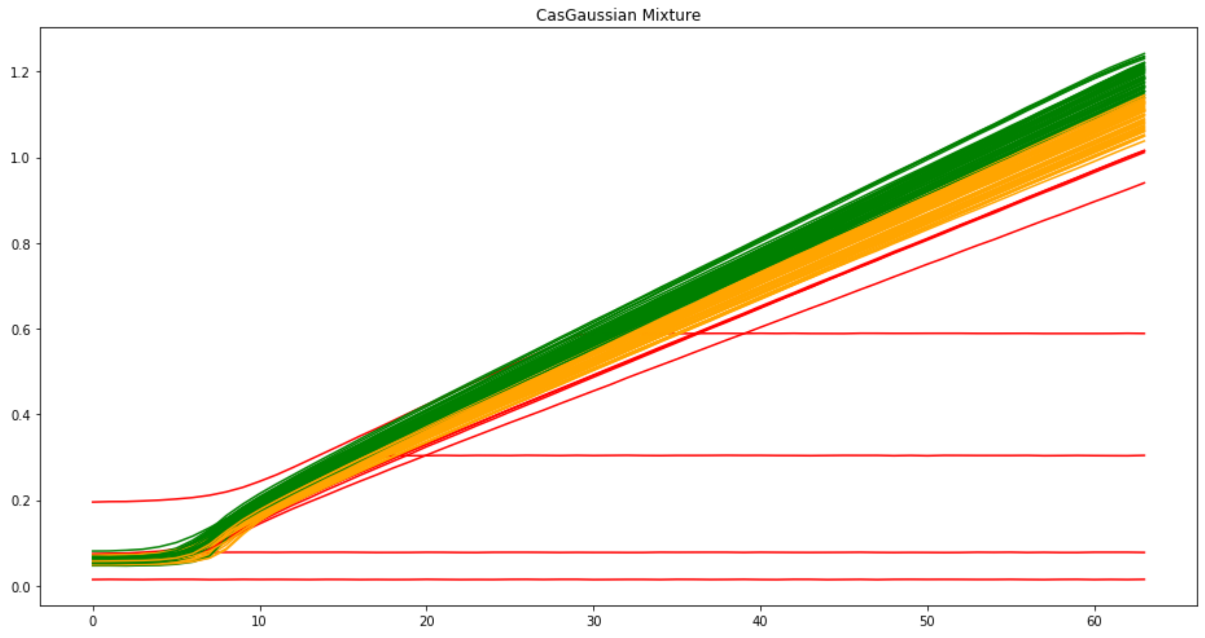
Functionality	Returns all tests of test stage object.
URL	`/api/stage/<string:stage>/<string:id>/failed`
Example Return	

```
[
    [
        {
            "failure": {
                "label": 2,
                "testName": "Fbk"
            },
            "chipID": "W1221A7"
        },
        {
            "failure": {
                "label": 4,
                "testName": "Delay"
            },
            "chipID": "W1221C2"
        }
    ]
]
```

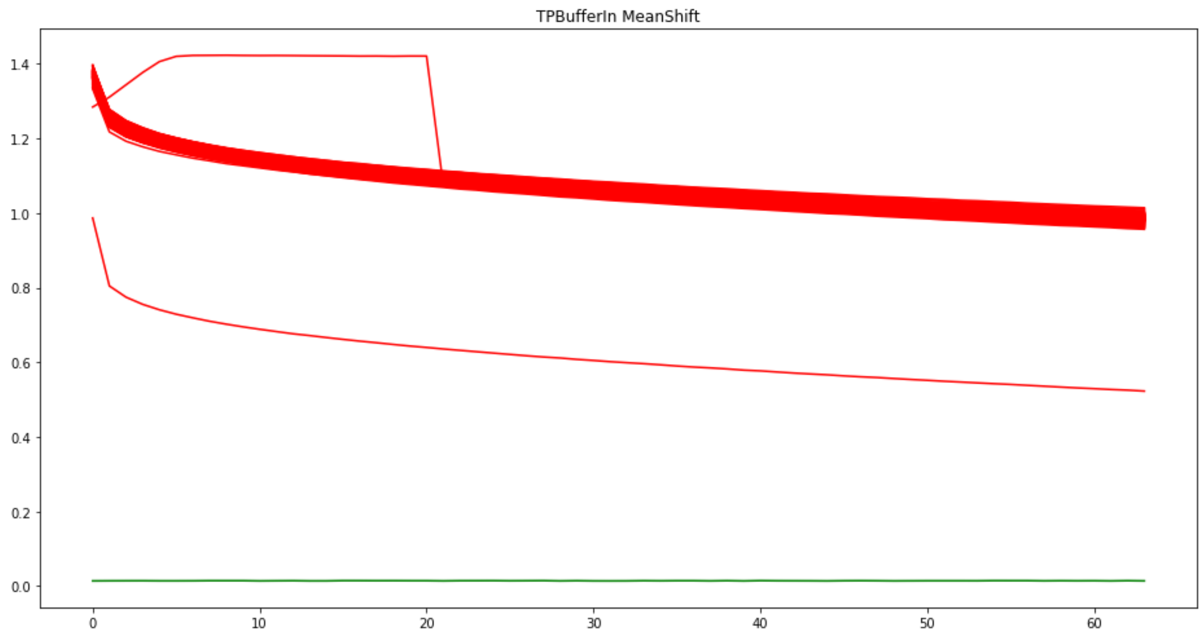
APPENDIX 4: CLUSTER ALGORITHM PLOTS

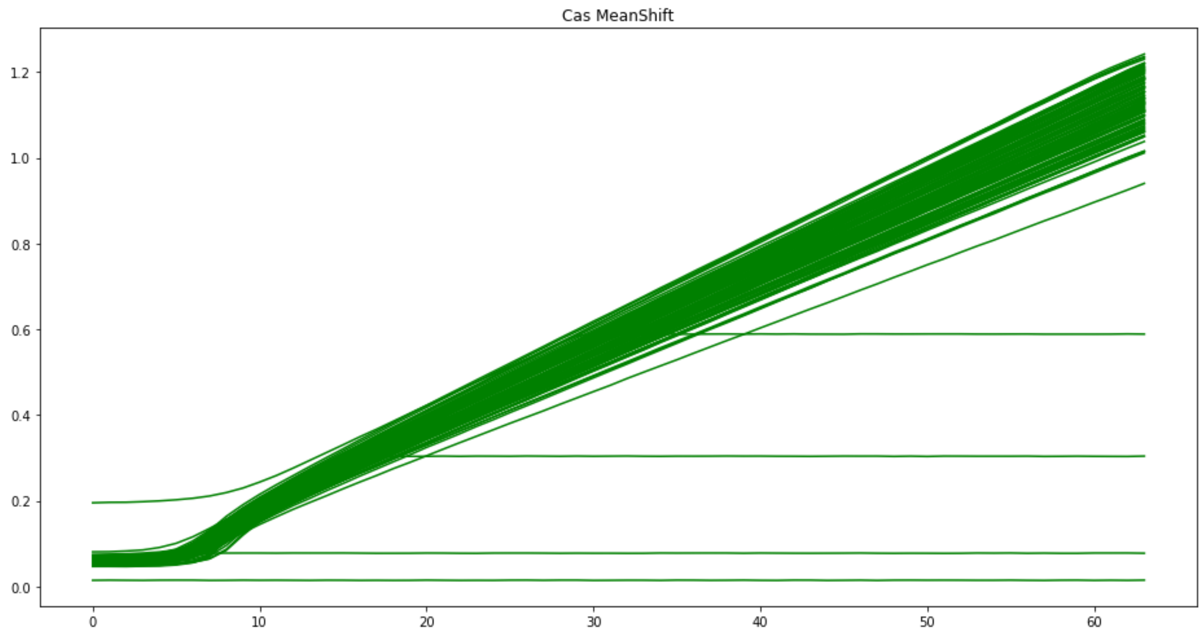
Gaussian Mixture



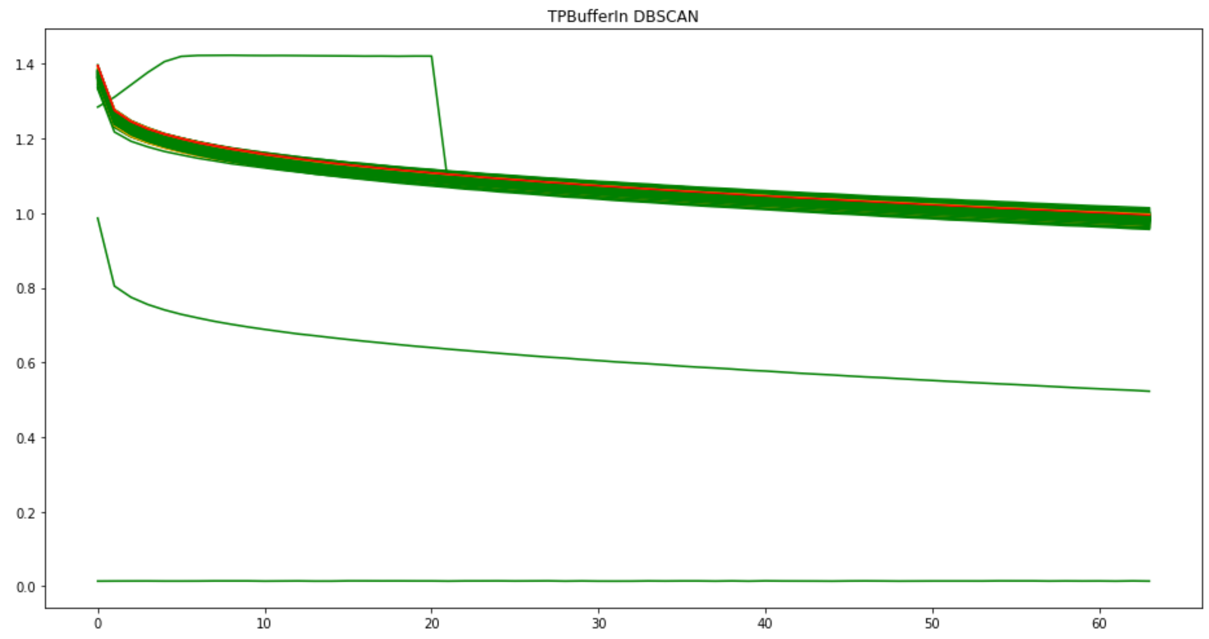


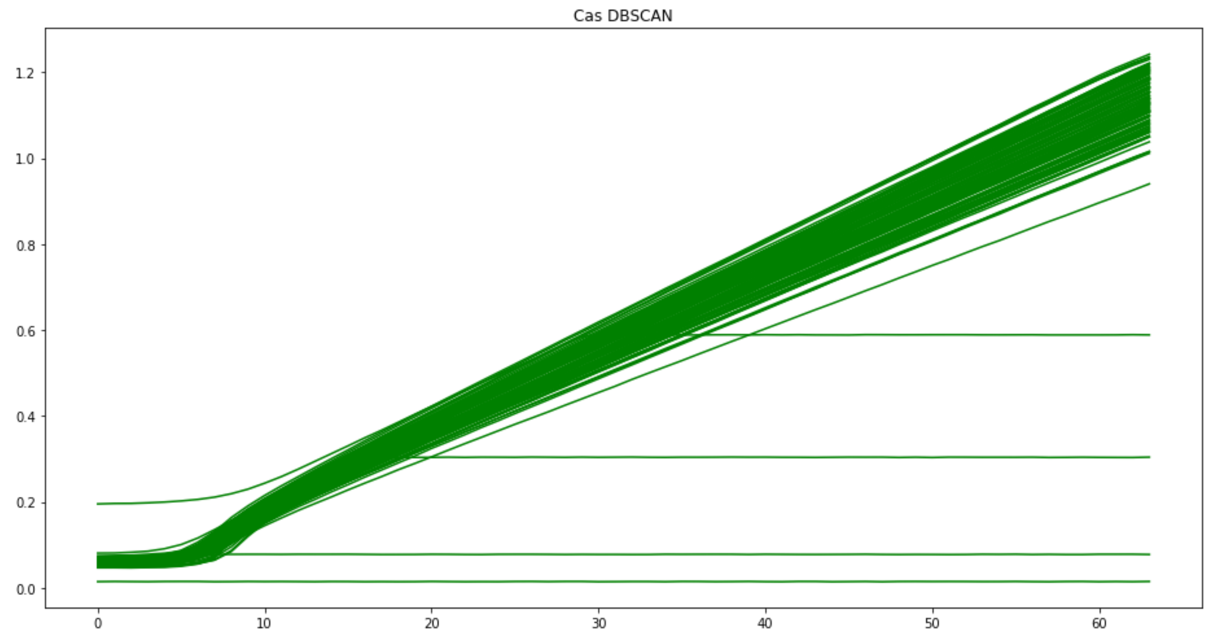
Mean Shift


TPBufferIn MeanShift
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60

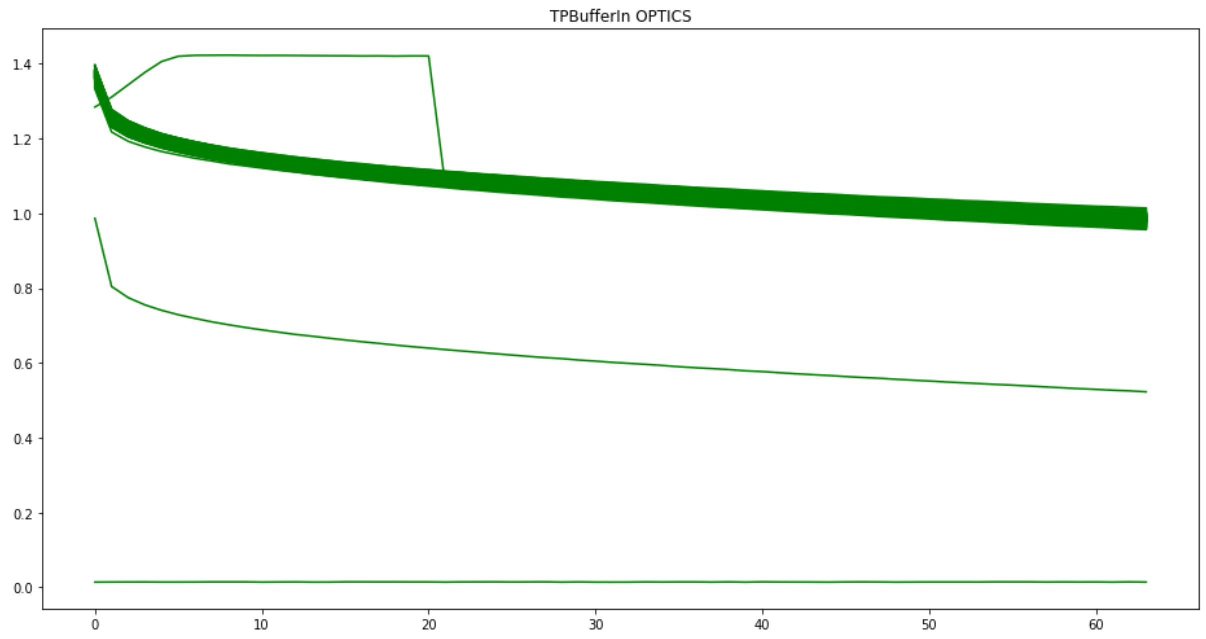

Cas MeanShift
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60

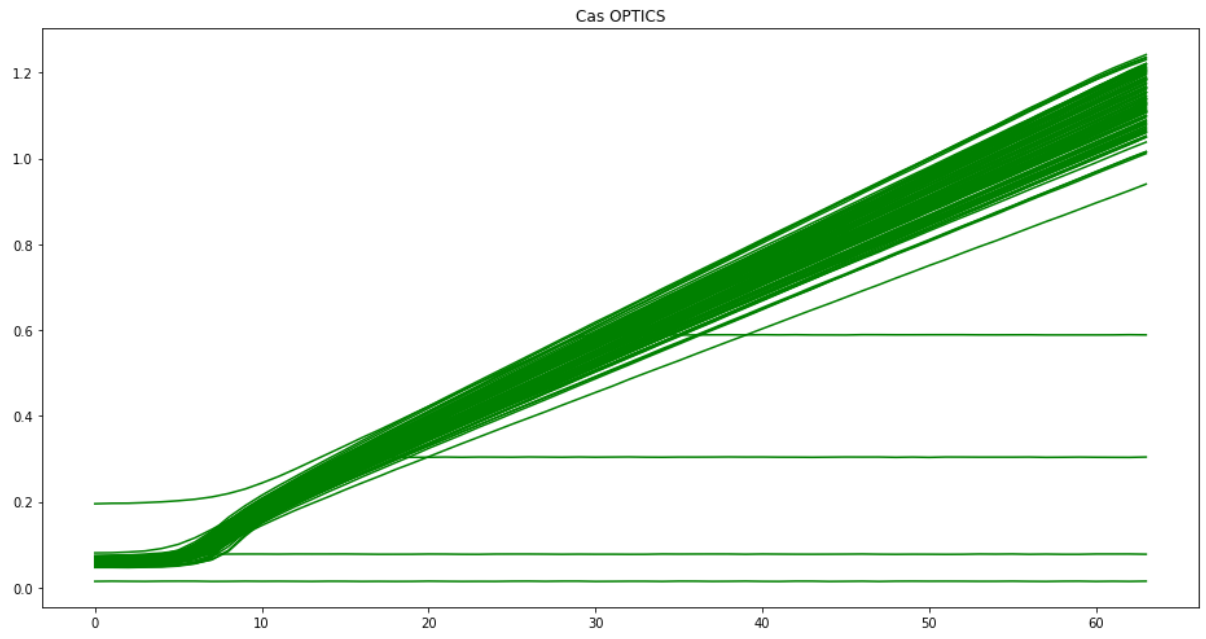
DBSCAN


TPBufferIn DBSCAN
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
10
20
30
40
50
60


Cas DBSCAN
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
10
20
30
40
50
60

OPTICS


TPBufferIn OPTICS
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60


Cas OPTICS
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60

APPENDIX 5: VERSION CONTROL

Version 1

Submission date:	6-5-2020
Updates:	*Initial version*

Version 2

Submission date:	13-04-2020
Updates:	*Spell improvements* *Explanation how scrum is used* *Explanation how Crisp DM is used* *Included information about Gaussian Mixture*

Version 3

Submission date:	19-5-2020
Updates:	*Explanation about statistical model* *Chapter about Docker* *Added the chapter about evaluation and the model assessment.* *Added abstract*

Version 3

Submission date:	15-06-2020
Updates:	*Fix undescriptive figure captions* *Explanation about different cluster algorithms* *Included research about different database technologies* *Added proof to lose statements* *Added missing literature and references* *Fixed better explanation about missing label problem* *Added conclusion*