Design project · University of Twente

Technical Computer Science

Visualising Movement and Occupancy of The UT-Library

Authors:
Rik van de Haterd
Indy Haverkamp
Pieter van Heijningen
Joris Kuiper
Mathijs Tobé

Supervisor:
Yanqiu Huang

Nov. 10th 2023

Abstract

This report was written during the "Design Project" module of the University of Twente. This design project has been made in collaboration with the UT-library and LISA to improve their data-driven decision-making capabilities. Throughout the years the library staff has placed various trackers and sensors around the library, which are able to track the amount of people who enter the library and the occupancy of the project rooms. However, few applications have yet been created that make use of this vast amount of data, therefore we have created a functional database and dashboard to gather, analyse and utilize this data. This report will describe the entire development life-cycle of the system in detail, starting at the domain analysis and going all the way through requirements specification, testing and system architecture. Finally, we'll discuss the team's collaboration with each other as well as direct stakeholders. A manual for the system is also included.

Contents

1 Introduction

In the ever-evolving landscape of modern data-driven enterprises, the need for effective data collection, storage, and analysis systems has become paramount. The university library stands as an intellectual cornerstone of the university, serving a multitude of roles that extend well beyond merely housing books and study spaces. For both the university and its students, the library represents a hub of knowledge acquisition, research, and academic community building.

Over the years, the library staff has installed multiple sensors throughout the library to monitor the library's foot traffic. However, there has been limited application of this substantial dataset. Therefore, we were asked to design a system that can effectively store and display this vast amount of data.

Our proposed system is a dynamic dashboard coupled with a time series database, and it is underpinned by a Python program designed to gather and insert information from the two critical sensors within the university library. This project aims to address the information needs of the library, with the end goal being to improve their ability to make data-driven decisions. The system could have an impact on many library operations, like opening hours and resource allocation, which can significantly affect students and staff of the university alike.

This report will encompass the whole development life-cycle of the system, adhering to a structured chapter order. Beginning with a domain analysis, we will dissect the context and domain of the system. Subsequently, we will elaborate on the requirements that drive the system's design, followed by an examination of potential risks associated with its implementation. The planning and test plan chapters will outline the project's timeline and the strategies for quality assurance respectively. The report will further delve into system design and detailed system design, providing an insight into the technical intricacies and architectural elements of the system. Additionally, the manual section explains how to operate the systems, ensuring effective utilization. In the evaluation chapter, we assess the collaboration of the team and the potential for future enhancements of the system. The report concludes with a discussion of the system's performance, suggestions for future projects that can build on our work, as well as our conclusions regarding the achieved results of our project.

2 Definitions

Throughout this document, we will be using various terms. Most of them will be explained where needed. This section will shortly define terms that are used throughout the entire document.

LISA (or Library, ICT Services & Archive) is the organization that has provided the case for this project, they will be referred to as the client or as the product owner.

Sensor This will refer to the sensors hanging at the entrance of the University Library at the University of Twente, these sensors measure the people going in and going out. These sensors function as the primary data source of this project.

Dashboard refers to the monitoring dashboard we have been asked to build. The goal of this dashboard is to improve the data-driven decision-making at LISA with regard to the library opening hours, garbage collection, and more.

TSDB (Time Series DataBase) refers to the time series database software that we will be using for this project. Currently, we have settled on InfluxDB.

SensorData API refers to the sensor data UTSP API provided by LISA to us from where we can query the sensor data.

Grafana Grafana is the tool we use to visualize our dashboards.

ASITO ASITO is the company contracted to clean the library. They are, for example, responsible for cleaning out the garbage cans, mopping the floors, and keeping the tables clean.

CFM Campus & Facilities Management (CFM) are responsible for the building the library is in. They decide on the opening hours and ensure repairs are performed for example.

3 Domain analysis

3.1 Data Analysis

The data provided by the IT staff of the UT-library is quite extensive, this section will explain the amount of data that we have on each sensor and what information we can retrieve from it.

3.1.1 The old people counter sensor (Brickstream)

At the main entrance of the UT-library, there is a people-counting sensor which has been active since July 2019. As was found, the reported data turned out to not be entirely accurate. Due to the sensor being not precise enough, more people appear to leave than have ever entered the library (Figure 1. `brickstream_cumulative_sum` goes below `0`). Besides various metadata, it reports the number of people who have left and entered the library since the last measurement. This is done on a set time interval (currently configured at once a minute).

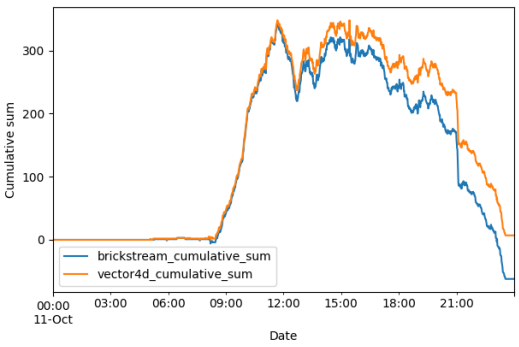


Figure 1: Occupancy sensor differences

3.1.2 The new people counter sensor (Vector4D)

To replace the poorly functioning Brickstream sensor, a new sensor with the same function was mounted around April 2023. This has given us several months of overlap in data between these sensors, which could allow for a correction of the old sensor data based on the new data. As this new sensor is believed to be nearly 100% accurate, it can be used as 'ground truth'. Similarly to the old sensor, it reports the total number of people who entered and exited within a set time interval.

3.2 Problem Analysis

The core problem of the client is that they currently have a lot of sensor data in different forms. But none of it can be easily visualised or analysed. This project should resolve that problem.

3.3 Stakeholder Analysis

For this project, we have identified three direct stakeholders who have a high interest in this project and a lot of influence.

Direct The staff of the UT-Library who need the data to make decisions.

Direct The network engineer from LISA who manages the sensors.

Direct The students who sit at the service desk of the library.

Direct CFM (Campus & Facility Management). They are responsible for building management.

Direct Another stakeholder group is ASITO. They provide the cleaning services.

Indirect UT students. They may be affected by decisions made using our system.

Because the system is primarily meant for making data-driven decisions by the UT-Library staff, we have planned to meet regularly with these supervisors. If it is not possible to meet on a weekly basis, we aim to send at least a weekly update over the mail to inform them.

3.3.1 UT-Library staff

The UT-Library staff will be the ones who are going to use our system the most. They want to use the data from the sensors to improve their decision-making regarding the opening times of the UT-Library or when it has to be cleaned. They will mainly use the dashboard and the final project to understand their data and draw conclusions from it.

Our goal is to ensure that they are sufficiently updated regarding the development of the project and that they are often asked for input on the developments.

3.3.2 Network administrator

The Network Administrator from LISA will be an important stakeholder on the technical side of the project. The administrator maintains the sensors and stores the data. Therefore the administrator has a lot of influence on our project and he should be highly informed on the project.

After this project is done, it is important that the administrator can maintain the codebase and dashboard with relative ease. The most important part is that more sensors can be added and the dashboard can be expanded with new visualisations of the data without having to contact us.

3.3.3 Project Supervisor

Our project supervisor will be helping us to ensure our project fulfils the academic standards that are needed for the design project. She will also be grading us and therefore it is important to meet with her often to discuss our progress. To ensure this we have set bi-weekly appointments with her to check our progress and ensure we stay on track. We will also keep her updated by email on our progress.

3.3.4 Service desk employees

Although one could argue that they are part of the UT-library staff. This group are the students who sit at the entrance of the library. They ensure that it is open and that they can help when needed. When it is decided to change opening hours, it will mean that they have either fewer hours or more hours. Since this impacts their income we have identified them as a direct stakeholder group.

3.3.5 CFM

CFM (Campus & Facility Management) is responsible for the maintenance of the building and decides when the building will be open. Although the library staff is able to request the change in opening hours, CFM has to approve it. The idea behind our dashboard is that LISA will be able to make a stronger case to CFM when to open the library thus making them a direct stakeholder.

3.3.6 ASITO

ASITO is a company that provides cleaning services for the entire university. They ensure that the buildings are clean and trash is taken out. LISA works closely together with them to ensure that the library is clean. However, LISA has noticed that garbage is overflowing some days and would like to use our dashboard to make a better case to ASITO as to when to clean the building. Thus our dashboard will have a direct influence on the services provided by ASITO, making them a direct stakeholder.

3.3.7 UT students

With the dashboard provided by this project, LISA will be making decisions regarding opening hours or resource allocation of the UT library based on the data provided by the dashboard. These changes in available resources and opening hours have a direct impact on students. They might not be able to study when they could normally or opening hours might be more accommodating for them. Thus it directly affects the students, making them an important indirect stakeholder of the project.

4 Requirement specification

4.1 Agile workflow

To ensure quality and flexibility when gathering requirements, the Agile approach to requirement management will be implemented. With this approach, multiple iterative cycles are performed to refine current requirements. Furthermore, new requirements can be added to guarantee the completeness of the system according to our clients' needs.

4.2 Tools used

4.2.1 Trello

To effectively manage and collaborate on the requirements outlined in this specification, we make use of the management tool Trello. Trello facilitates task organization and assignment, enabling coordination among team members, streamlining the workflow, and ensuring that each requirement is appropriately addressed.

4.2.2 Gitlab

To ensure efficient collaboration and maintain a structured codebase, we utilize the University's Gitlab instance. This platform not only provides version control for our code but also offers an array of collaborative features. It enables team members to work cohesively, manage code contributions, and maintain a well-organized and accessible code repository.

4.2.3 InfluxDB

For our data storage needs, we chose to implement InfluxDB. InfluxDB's specialization in time-series data management aligns perfectly with our project's requirements, facilitating efficient data storage and analytics.

4.2.4 Grafana

For the visualization of our project, we implemented Grafana. Grafana allows us to create dynamic and interactive dashboards, enabling real-time visualisation. This dashboard improves our project's overall presentation and provides a clear tool for data analysis and visualisation.

4.3 Requirement formation

The requirements will be formulated according to the SMART criteria, which means they must be: **S**pecific, **M**easurable, **A**cceptable, **R**easonable and **T**ime-bound. This results in requirements which are well formulated and as useful as possible for the project.

4.4 Requirement prioritization

Determining the priority of requirements is essential in building a system efficiently and ensuring a satisfied client. Therefore, the requirements have been prioritized using the MoSCoW strategy[1]: **M**usts, **S**houlds, **C**ould and **W**on't. This can be seen in the next section as indicated by the letters M, S, C and W, respectively.

4.5 Functional requirements

M The data analysis model shall use the data of the new people counter sensor as the ground truth.

M The data analysis model shall provide a way to normalize the historical data of the old people counter.

M The data analysis model shall use the people counter sensor to be used for analysis.

[1] https://www.productplan.com/glossary/moscow-prioritization/

M The data analysis model shall reset the cumulative count of the number of people present every night at 3 a.m. to correct any mistakes.

M The dashboard shall visualize the in and out traffic in 15-minute, 1-hour, and 1-day intervals.

M On the dashboard, the user shall be able to compare weeks from different academic years.

M On the dashboard, the user shall be able to check specific days for traffic and occupancy of the library.

S The dashboard should show enters and exists as separate graph lines.

S The dashboard shall display the total occupancy of the library in real time.

C The data analysis model should predict entrance/exit movement on a daily level, but preferably hourly or 15 minutes.

C The data prediction model should include weather data.

C The data prediction model should include academic year calendar data.

C The system should allow for academic year calendar data to be uploaded as e.g. CSV.

C On the dashboard, the user can check the characteristics of a data point, like the weather or the day in the academic year calendar.

C The dashboard shall visualize the predicted data of the prediction model on the data.

4.6 Non-Functional

M The dashboard shall load data within 5 seconds.

M The dashboard shall be able to switch between 15-minute, 1-hour, and 1-day intervals within 5 seconds.

C Sensor data redundancy should be minimal, i.e. data queried from the UT database should not be stored unnecessarily.

S The database should be efficient in creating graphs (grouping by 15, 30, or 60 minutes) and handling many measurements.

4.7 Out of scope

These won't requirements are set in place to be able to clearly define the scope of the project.

W To reduce data querying and redundancy, databases, and schemes won't be implemented on the UT servers.

4.8 User stories

For the user stories, we have identified two main users of the system. Therefore, the user stories have been divided among these two types of users.

4.8.1 The UT-Library staff

As an employee of the UT-Library

- I want to know when the most people are entering the library by day, hour, quarter etc.
- I want to know when the most people are leaving the library.
- I want to be able to compare the data on different opening hours.
- I want to know how many people were in the library compared to the amount of available seats.

- I want to know how the weather impacts the amount of people coming to the library.
- I want to be able to compare the weeks from the modules across the years.
- I want to know how many people have visited the library during the holidays.
- I want to know how busy it is each quarter on each day.
- I want to know which day of the week is the busiest.
- I want to know how busy it is during certain timeframes.
- I want to know how the student numbers are increasing over the years compared to the occupancy of the library.
- I want to see a projection of how many students can be expected during certain days or weeks.
- I would like to see how people are booking the project rooms across the years to see behavioural changes in booking the rooms.
- I want to be able to edit the dashboard to add more visualisations.

4.8.2 Network administrator

As the network administrator:

- I want to be able to add more sensors to the system
- I want to be able to add different sensors to the system
- I want to be able to edit the dashboard to add more visualisations

5 Risk analysis

5.1 Displaying false data

One of the main risks of our system is that we unintentionally showcase data that turns out to be false. Due to the amount of processing that is needed for combining and correcting different data sources, it is not unreasonable to think a mistake might be made which might not be immediately apparent. If this is the case, it could lead to the library making decisions based on this incorrect data, which could in turn negatively influence the quality of life in the library.
To combat this risk, we will be sure that on every processing step, we analyse the data with the utmost care to be sure that data has not been incorrectly manipulated.

5.2 Human error with updating Excel files

As will be explained in the user manual (Chapter 11) part of the report, there are several Excel files that need to be manually updated once in a while. This mainly concerns academic year data. The manual insertion presents the risk of human error, which could, for example, result in incorrect data visualisations on the dasboard.
To combat this risk, we made sure to provide adequate instructions on how to update the Excel file, together with making the Excel as user-friendly as possible.

5.3 Timeseries Database (TSDB) downtime

Another risk that might affect our system is the server with the database going down. This would result in data being parsed by our system but not inserted into the TSDB. Although the main goal of the system is to help in data-driven decision-making by the UT-library, a gap in the live data is not favorable in our opinion.
To combat this risk, we have added a check where our code requests the last inserted point in the TSDB and then starts inserting from that point. This will mitigate the risk of data not being inserted.

5.4 Other risks

This section covers smaller risks that are there but do not pose a huge problem that warrants them to get their own subsection.

5.4.1 Sensor being offline

If the sensor is offline, there is no way of retrieving data that should have been measured in the meantime. On our end, this will result in a gap in the data. However, this gap should not have too much impact on the visualisation of that day.

5.4.2 Our system being offline

If our system is offline, data will not be inserted. However, due to the check discussed above, if the system is started again it will start inserting data from the last point data was inserted. Thus this risk has minimal impact in the long run.

5.4.3 Change of sensor

If the sensor is changed to a new or different type, a new piece of software has to be written that will parse the output to our desired class. However, since LISA has fairly recently installed the current sensor the risk of this happening in the next 2 to 3 years is minimal. If the sensor needs to be replaced, a small update needs to be made to filters if names and/or ids have changed.

6 Original Planning

This is the original planning we made at the start of the project, in the evaluation part you can find how the actual project ended up being completed.

6.1 Deadlines

The only hard deadlines for the project are as follows:

- September 19th: Project proposal and plan
- October 3rd: Test plan
- October 17th: Design report
- October 31st: Presentation
- November 9th: Final poster session

6.2 Week planning

Generally, we will aim to follow this weekly planning, to be sure that we are on track to finish the project on time:

Table 1: Week plan

Week nr.	**Purpose**
1	Setting up requirements, finding a supervisor
2	Finalising requirements in project proposal
3	Exploring what technologies to use
4	Build parses to easily process the data, test plan
5	Start analysing the data
6	Correcting historical data, presentation
7	Correcting historical data
8	Building the dashboard
9	Building the dashboard, poster
10	Final touches

6.3 Task division of team members

The remaining work will divided equally among all 5 team members, with team members specialising in the following areas:
- Rik will mainly be working on requesting and processing weather and academic calendar data. Which will help with the predictive model.
- Indy will focus on processing the data retrieved from the library's API into a usable format
- Pieter will focus on retrieving the data from the library's API
- Joris will focus on inserting the data into our database and setting up the general infrastructure.
- Mathijs will focus on correcting the data from the brickstream counter.

7 Testing plan

7.1 Introduction

This section of the project was initially written in the earlier weeks of the design project and the language reflects as such. The content, however, is still relevant to our final project and this section has been modified to ensure this.

7.2 In scope

The scope defined in this document will focus on the following functional and non-functional requirements as have been identified in the project proposal. These requirements are also here classified according to their MoSCoW classification.

7.2.1 Functional Requirements

M The data analysis model shall use the data of the new people counter sensor as the ground truth.

M The data analysis model shall provide a way to normalize the historical data of the old people counter.

M The data analysis model shall use the people counter sensor to be used for analysis.

M The data analysis model shall reset the cumulative count of the number of people present every night at 3 a.m. to correct any mistakes.

M On the dashboard, the user shall be able to check specific days for traffic and occupancy of the library.

S The dashboard should show enters and exists as separate graph lines.

S The dashboard shall display the total occupancy of the library in real time.

C The data analysis model should predict entrance/exit movement on a daily level, but preferably hourly or 15 minutes.

C The data prediction model should include weather data.

C The data prediction model should include academic year calendar data.

C The system should allow for academic year calendar data to be uploaded as e.g. CSV.

C On the dashboard, the user can check the characteristics of a data point, like the weather or the day in the academic year calendar.

C The dashboard shall visualize the predicted data of the prediction model on the data.

7.2.2 Non-Functional Requirements

C Sensor data redundancy should be minimal, i.e. data queried from the UT database should not be stored unnecessarily.

7.3 Out of scope

For this testing plan, we have listed below the functional and non-functional requirements we will not be testing. This mainly concerns the visual aspects of the project because these are heavily susceptible to change if LISA and/or our supervisor desire such.

7.3.1 Functional Requirements

M The dashboard shall visualize the in and out traffic in 15-minute, 1-hour, and 1-day intervals. *We will not be testing this since the graphs can and will be changed during the project and intervals might also change if the client desires as such.*

M On the dashboard, the user shall be able to compare weeks from different academic years. *This requirement is left out because we will only be testing if the academic year relation is working.*

7.3.2 Non-Functional Requirements

M The dashboard shall load data within 5 seconds. *This will be handled by our TSDB and thus we cannot test this*

M The dashboard shall be able to switch between 15-minute, 1-hour, and 1-day intervals within 5 seconds. *This is something that is dependent on the software we will use to visualize our data and thus testing this is not relevant.*

S The database should be efficient in creating graphs (grouping by 15, 30, or 60 minutes) and handling many measurements. *This is also handled by our TSDB and thus not something we can impact or change.*

7.4 Quality Objective

The overall objective of our plan is to ensure that our software works as intended, does not show unexpected behaviour and catches most edge cases while trying to complete the requirements as discussed earlier.

7.5 Test Methodology

7.5.1 Overview

For our project, we have chosen to adopt an agile approach to testing and developing in general. This is because of the lifespan of our project (total of ten weeks) and having to be able to react quickly to any and all changes.
However, the time we have for our project does allow us to define sprints as is normal with Agile. We have decided to *focus* on specific parts of the project for a duration of time and prioritize work for that part of the project.

7.5.2 Test Levels

Unit testing Testing all parts of the code separately to ensure the correctness of the functions.

Integration testing Testing that the separate parts of the project can work together as intended.

System testing Testing that ensures the entire system works as has been laid out in the (non-)functional requirements.

Data testing Testing to ensure the data is not malformed or could have unexpected formatting that could cause the project to fail execution

User testing Testing the visual part of the requirements with the end user to ensure the user can easily understand the dashboard.

7.5.3 Test Completeness

To ensure our tests are complete we will be utilizing the following criteria:

- 90-100% unit test coverage.
- All manual and automated tests executed.
- All methods have documentation and the entire system is well documented.
- 100% Test Case Pass Rate

7.5.4 Test Coverage

Test coverage is an important metric we will be using for our unit testing. It is important to reach a high percentage such that most code is tested. However, a high test coverage might show the false impression that all code is tested properly while this is not the case.

For example, code can be tested by a simple test case to test expected behaviour, however, it might still fail with an edge case that has no test. Therefore, test coverage should not be regarded as the main metric.

7.5.5 Manual and Automated Tests

This metric will ensure that all tests have been executed. The specific tests will be discussed later in this document. By ensuring that all tests are executed, most behaviour (unexpected and expected) can be tested against to ensure the correctness of the program.

Similar to as discussed with the test coverage metric, the amount or percentage for the manual and automated tests executed does not ensure that all behaviour or possibilities are tested against. Thus it is also important here to identify the various scenarios to ensure that most edge cases are caught.

7.5.6 Documentation

Another important metric is the documentation of the project. As it is a possibility that another project in the future will build upon this project or that it needs maintenance, it is important that the project is well documented.

To ensure this, the aim is to have all important methods and classes in the code base to have documentation. Although there is no clear way of measuring proper documentation, the aim is to have the documentation be clear to someone without any knowledge of the details of this project.

Furthermore, the documentation should also clarify the flow of data between parts of the project and show how the code base is structured through the use of UML diagrams.

7.5.7 Test Case Pass Rate

This metric is fairly straightforward. All tests we write should also pass. If a test is designed to fail and does, then it passes with regards to the Test Case Pass Rate.

7.6 Resource & Environment Needs

7.6.1 Testing Tools

To achieve the goals of our testing, the following tools will be utilized.

- Python unittest for unit testing
- Python unittest for integration testing
- System and user testing will be performed manually

7.6.2 Test Environment

For the test environment, Table 2 shows the minimum requirements for the test environment. We have decided to only list Linux as the operating system for testing due to it being the most commonly used operating system for servers and being the base image for our docker container. However, we will also test everything with Windows 10 or higher to ensure that the software works across various operating systems.

Table 2: Environment Requirements

Category	**Software/Tool**	**Minimum version**
Operating System	Ubuntu	LTS 22.04
Programming language	Python	3.11
TSDB	InfluxDB	2.7.1
Dashboard	Grafana	10.2.0

7.7 Unit Testing

The unit testing will be done with the unittest framework that is provided by Python. This framework offers lightweight testing and offers the capabilities to perform dependency injection where needed.

The precise unit tests can be found in the test package of our source code and the aim is to achieve a coverage between 90% and 100%.

7.8 Integration Testing

For integration testing, the unittest framework will be also used. The manner of testing will differ greatly. Where unit tests focus on testing separate parts of the code base, the integration tests will focus on testing the separate parts together to ensure interoperability.

Integration tests will also increase the focus on edge cases in the data sources due to the project combining multiple sources into one data point that is inserted into the TSDB.

7.9 System Testing

This part of the test plan will list the system tests that will be made. For each test, it is required to have an active connection with the API provided by LISA and a proper connection to the TSDB. Between each test, the buckets containing the data of InfluxDB need to be reset.
The focus of the system tests will be on ensuring that the entire system works as expected. This list will expand as more tests are created. 3 shows a summary of all the system tests we have identified.

7.9.1 System test to ensure all manners of data insertion work together

This is a test that can manually be performed when major changes are made to the code base. This will ensure that the data is still correctly being inserted into our TSDB.

Table 3: Types of system tests.

Description	**Expected result**	**Status**
Run `/jobs/sync_peoplecounters.py`	Script runs without any errors and InfluxDB contains the latest sensor data	Pass or fail
Run `/jobs/sync_weather.py`	Script runs without any errors and InfluxDB contains historic and forecast weather data	Pass or fail
Run `/jobs/sync_daily_max.py`	Script runs without any errors and the data in InfluxDB has the max counts of the past days	Pass or fail

7.10 User Testing

To ensure that the product satisfies the client's demands, frequent meetings will be organised to present the current state of the dashboard and collect feedback to improve the dashboard.

Due to the dashboard being developed in InfluxDB and Grafana, most UI/UX elements cannot be modified or changed. The testing will mostly focus on the types of graphs, the colour scheme and the representation of the data. It is important that the client can easily digest the data that is displayed on the dashboard such that they can improve their decision-making. Therefore there is no strict procedure set for testing the interaction of the user with our dashboard. During the frequent meetings, iterations of the dashboard will be shown to the client and feedback will be collected.

8 System Design Structure

This chapter delves into the core components of our general system design structure. It provides an explanation on these elements and sheds light on how they shape the architecture of our system.

Object-oriented programming (OOP) is the first area of focus. The OOP approach is central to our design philosophy as it helps in managing complex and large-scale systems and promotes structured software development practices.

The discussion then shifts to the module structure. This subsection covers the organization of the system and how various components interact. An effective module structure enhances the maintainability, scalability, and overall manageability of our system.

Following the module structure, we examine the database structure. A well-designed database structure is essential to our system, as we need to interact with millions of data entries.

The chapter concludes with an overview of the dashboard. This dashboard is the main product of our system which gives LISA access to the functionality and data. The design and functionality of the dashboard significantly impact the usability and effectiveness of our system.

By gaining an in-depth understanding of these components and their interplay within the general system design structure, this chapter sets the stage for a more comprehensive look into the technical intricacies of our system.

8.1 Code Structure

8.1.1 Object-oriented programming

We built the system loosely following the Object-oriented programming (OOP) paradigm, as Python is not designed to be an Object-oriented programming language. We chose this approach for its ability to enhance code organization and maintainability. OOP breaks down complex systems into manageable objects, which makes development and maintenance easier. As we are working with physical sensors and their states, this can be transformed for OOP easily.

8.1.2 System structure

Figure 2 shows the general outline of our Python application. Functionality is split into three parts: common, jobs, and providers.

In order to experiment with the available data and train models, utilities are available in `common` to transform measurements from minute basis to many different forms: grouping by an interval, filling missing minutes with 0, combining duplicate minutes, coarsening measurements, combining datasets, calculating the cumulative sum, and more. These utilities have to handle millions of data points in memory, for which they are optimized.

`jobs` contains scripts that are run periodically using cron. These scripts are designed such that they can be run on an empty database and sync all historic measurements as well.

The most important part of our system is `providers`. We split the functionality of our system into five components. These providers and expose few public methods, such as returning data for a specific timestamp. Each provider should be contained in its part of the system and work on its own. This way each provider can be improved and updated independently.

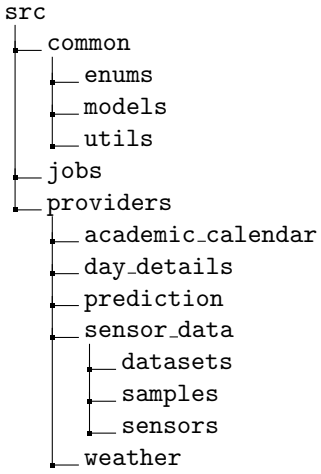


Figure 2: Source code tree

8.2 Database Structure

Our database structure, due to being a time series database has a quite simple structure. Essentially it is a giant key-value map, with the key being a timestamp, and the value being a float, integer, string or boolean. These are all stored in a single "bucket" (database), which contains a number of different types of 'measurements' (tables) with 'fields' (columns).

- `people_counters`, which contains measurements for each minute of how many people entered and exited the library in the form of the fields `enters` and `exits`

- `people_counters_computed`, which contains pre-computed maximum occupancy levels for each day in a `daily_max` field. This is done to improve the performance of the historic data dashboard. Additionally, the `daily_max_prediction` field stores the daily maximums we predict.
- `weather`, which contains hourly data as fields `rain` and `temperature`, both in separate fields

Each row or entry in this key-value map can have additional details 'tags' (metadata), which can, for example, be the ID of a sensor. The fields in the `people_counters` measurement also contain quite a number of meta-data points:

- `academic_day`, the number of days since the start of the academic year
- `academic_year`, the academic year the data point falls under
- `device_id`, the device id of the people counter sensor
- `exam_week`, whether or not this data point was during an exam week
- `holiday`, contains the name of the holiday if the data point was during a holiday, otherwise "Normal".
- `quartile`, which academic quartile this data point falls under. Q5 is the summer vacation.
- `week`, which week of the year this data point falls under

8.3 Dashboard Overview

In consultation with the stakeholders, we decided to deliver two dashboards. Namely a live dashboard, which can show the latest data and updates every couple of minutes. And a historical dashboard, which displays and compares data from multiple years back.

8.3.1 Live dashboard

Figure 3: Screenshot of the live dashboard

In figure 3 a screenshot of the live dashboard can be found. This contains a graph showing the occupancy over time in the library, along with temperature and rain information.
Secondly, a graph showing the total amount of movement per hour, in terms of people exiting and entering the library. This can create some insights on when for example garbage should be cleaned.
Finally, there is also a simple gauge, to simply display the current occupancy.
The above two graphs are configurable with the menu on top, where a specific time range can be selected.
This is a dashboard that can be easily displayed in for example a large monitor in the library, as it continuously updates to showcase the latest data.

8.3.2 Historical Dashboard

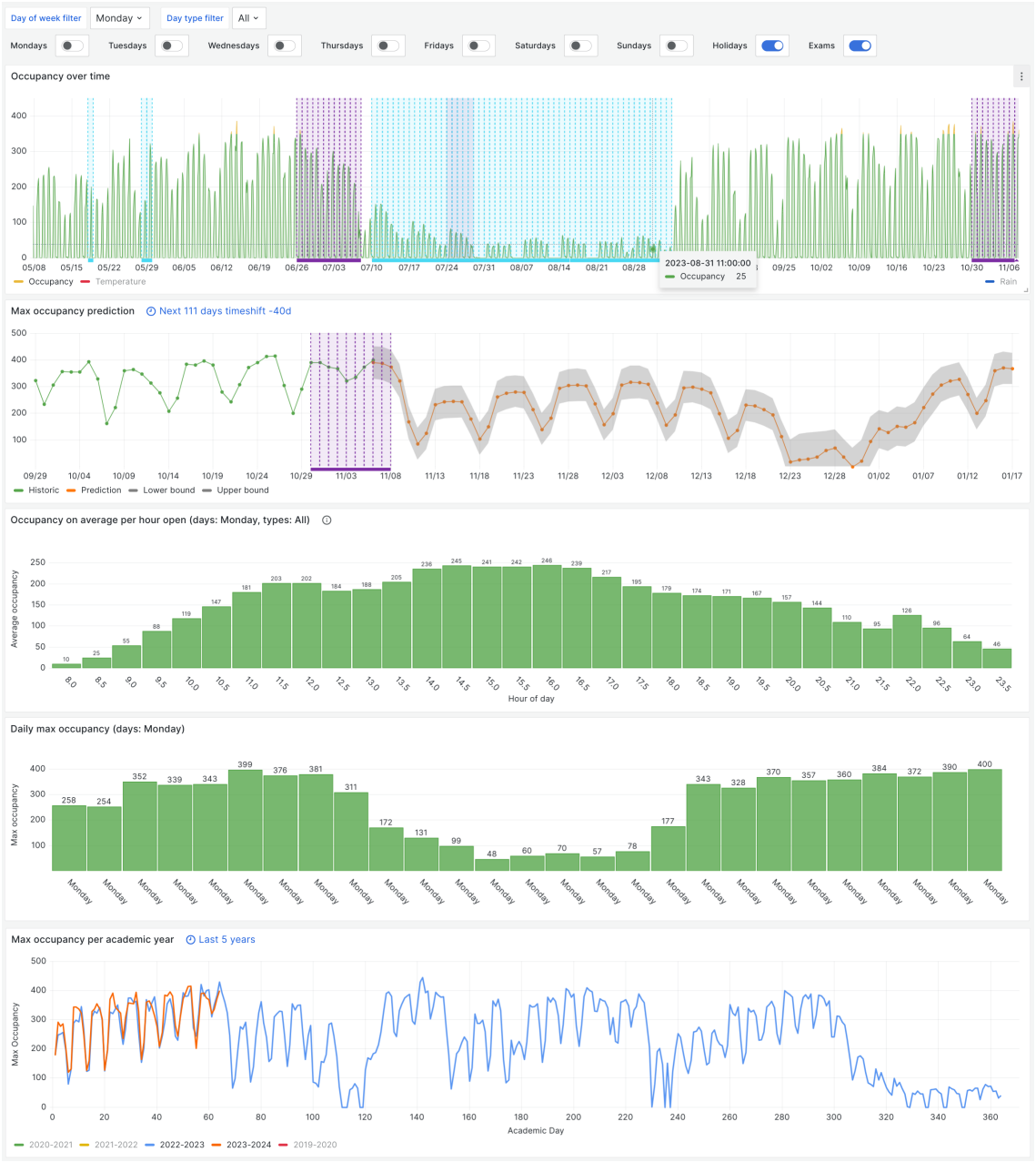


Figure 4: Screenshot of the historical dashboard

In figure 4 a screenshot of the historical dashboard can be found. As mentioned before the goal of this dashboard is to provide insights into the growth and long-term development of the occupancy in the library. This dashboard consists of the following graphs:

Occupancy over time graph Similar to what is displayed in the live dashboard, however, this one shows data from 6 months back by default. Additionally, it can also highlight which days are part of exam weeks and holidays.

Max occupancy prediction This graph shows the predicted maximum occupancy of 40 days in the future. Together with the maximum occupancy of the past 40 days. Mainly the prediction part is important here, as it provides insights to be able to make decisions beforehand.

Occupancy on average per hour open Over the selected time range, this graph averages the occupancy of each hour over all days. This creates insights into when most people are usually in the library, which can assist in deciding the opening hours schedule.

Daily max occupancy This is a customizable graph which has the unique feature of comparing selected weekdays against each other. Which weekdays are compared can be selected at the top of the dashboard. This can help create insights into how busy weekdays are compared to weekends for example.

Max occupancy per academic year This is a graph that compares the different academic years against each other. With this, it can easily be seen if, for example, this year is more busy compared to the previous year.

9 Detailed System Design

This section describes parts of our system in detail and justifies technical design choices. Our system consists of five important parts, of which the first four are part of our Python application:

1. Loading and correcting measurements from different data sources.
2. Extending measurements with details such as weather and academic data (seasonality).
3. Predicting using these measurements and details.
4. Synchronizing measurements to our TSDB.
5. Visualizing data on a dashboard.

Figure 21 in Appendix B contains a class diagram of the components that will be discussed in the following sections and their interactions within our system.

9.1 SensorData API

The SensorData platform is the API provided by LISA which can be used to read raw measurements from datasets. Each dataset contains measurements of a single type of sensor.

9.1.1 Datasets

Relevant datasets for us are the Brickstream and Vector4D datasets. Additionally, we are provided with a Brickstream CSV export from 2019-07 to 2020-4 not recorded on this new platform.

Each dataset is represented by a class. Each class implements its own parser to transform raw samples to `PeopleCounterSample`s. The most basic dataset implementation for a CSV dataset is as follows:

```
class CsvBrickstreamDataset(CSVDatasetAbstract[PeopleCounterSample]):
    CSV_FILE_PATH = "/dataset.csv"

    def __init__(self, sensor_id: str):
        super().__init__(self.CSV_FILE_PATH)
        self.sensor_id = sensor_id

    def parse(self, obj: dict, tz: tzinfo = ZoneInfo("UTC")) ->
    ↪ Optional[PeopleCounterSample]:
        device_id = obj['SensorId']

        if device_id != self.sensor_id:
            return None

        dt = iso8601.parse_date(obj["StartDateUTC"]) \
            .astimezone(ZoneInfo("UTC")).astimezone(tz)

        sample = PeopleCounterSample(
            timestamp=dt,
            exits=obj['exits'],
            enters=obj['enters'],
            tags={
                "device_id": device_id,
            }
        )

        return sample
```

Here, parse accepts a `dict` type argument, which is a line of the CSV parsed into a dictionary, with column names as keys. The `CSVDatasetAbstract` has a `get(start, end)` method which the dataset implementation inherits. This method implements opening and reading the actual file, and calling `parse()` for each line in the CSV. By designing the datasets like this, a dataset can be instantiated with a specific sensor ID and serve as an API for a specific sensor. A similar structure is used for datasets from the SensorData platform. Instead of a `CSVDatasetAbstract`, `BrickstreamDataset` and `Vector4DDataset` extend a `SensorPlatformDatasetAbstract`. This abstract reads raw lines from the SensorData platform, instead of from a CSV file, but can be interfaced with in a similar way.

9.1.2 Configuring sensors

Since over time sensors may eventually get replaced and data sources may change, our system is designed such that sensors can be defined in terms of location/position. Multiple data sources can be configured to be used for different time ranges. For example, the people counter sensor in the library is called `UbEntrancePeopleCounter`. An example of a sensor definition can be seen below. The sensor extends `SensorAbstract` and defines the sample type as `PeopleCounterSample`.

```
class UbEntrancePeopleCounter(SensorAbstract[PeopleCounterSample]):
    NEW_SENSOR_ID = "vrijhof-ub-ingang"

    csv_old_sensor = CsvBrickstreamDataset(sensor_id="1001",
    ↪  new_sensor_id=NEW_SENSOR_ID, apply_corrections=True)  # static
    old_sensor = BrickstreamDataset(sensor_id="utwente-ub",
    ↪  new_sensor_id=NEW_SENSOR_ID, apply_corrections=True)  # static
    new_sensor = Vector4DDataset(sensor_id="00:21:AC:04:23:2E",
    ↪  new_sensor_id=NEW_SENSOR_ID)  # static

    def __init__(self):
        super().__init__({
            datetime(2019, 9, 2, tzinfo=ZoneInfo("Europe/Amsterdam")):
            ↪  self.csv_old_sensor,
            datetime(2020, 6, 5, tzinfo=ZoneInfo("Europe/Amsterdam")):
            ↪  self.old_sensor,
            datetime(2023, 4, 2, tzinfo=ZoneInfo("Europe/Amsterdam")):
            ↪  self.new_sensor,
        })
```

Here, the `UbEntrancePeopleCounter` inherits a method `get(from_datetime, until_datetime)` that automatically joins measurements from these datasets, based on the dates defined in the constructor. These dates serve as the start dates of a dataset. The different sensor ID across datasets is replaced by a single ID `vrijhof-ub-ingang`, abstracting these different sensors from the dashboard. Additionally, each dataset can be configured to perform corrections (discussed further in Section 9.1.6) if the dataset implements it.

Now that we can fetch all data automatically from the right datasets, the only code needed to read data from the `UbEntrancePeopleCounter` would be:

```
sensor_instance = UbEntrancePeopleCounter()
samples = sensor_instance.get_from(datetime(2023, 1, 1), datetime(2023, 10, 1))
```

9.1.3 Data models/Sensor samples

As described above, every sensor and dataset should define what type of sample they work with and return. For example, a `PeopleCounterSample` has the following simplified structure:

```python
class PeopleCounterSample:

    def __init__(self,
                 timestamp: datetime,
                 exits: int,
                 enters: int,
                 tags: PeopleCountTags = None):
        self.timestamp = timestamp
        self.exits = exits
        self.enters = enters
        self.tags = tags if tags else {}

    @staticmethod
    def create_from_json(json_input: dict):
        return PeopleCounterSample(
            iso8601.parse_date(json_input["_time"]),
            json_input["exits"],
            json_input["enters"],
            tags={
                "device_id": json_input["device_id"],
            }
        )
```

This format is similar to how data is stored in the TSDB, such that data from the TSDB can easily be loaded into samples again using `create_from_json()`. Similarly, weather data is also stored in its own measurement as discussed in Section 8.2.

```python
class WeatherData:

    def __init__(self, timestamp: datetime, temperature: float, rain: float,
    ↪  tags=None):
        self.timestamp = timestamp
        self.temperature = temperature
        self.rain = rain
        self.tags = tags if tags else {}

    @staticmethod
    def create_from_json(json_input: dict):
        return WeatherData(
            iso8601.parse_date(json_input["_time"]),
            json_input["temperature"],
            json_input["rain"],
            tags={
                "type": json_input["type"]
            }
        )
```

9.1.4 Data cleaning utilities

The sensor data we work with has to be cleaned or streamlined. For example in the range 2023-01-01 and 2023-10-01, there are `63` minutes without measurement. Additionally, `72` measurements fall in the same minute (possibly due to a sensor glitch).

`fill_and_combine_missing_minutes()` This utility first fills a list of `PeopleCounterSamples` based on a provided date range, such that for each minute in this range a sample exits. If a minute is missing from the provided list, `0` is used as the default value for enters and exits. Next,

the enters and exits of duplicate minutes are combined into one sample. This step also trims every sample's timestamp to have minute precision and ensures all samples are sorted by timestamp.

`populate_samples()` This utility links sensor data to weather data, converting `PeopleCounterSample`s into `PopulatedCounterSample`s. Linking data is not necessary when simply inserting new data into the TSDB, but it makes working with the data on a local machine easier.

```
class PopulatedPeopleCounterSample(PeopleCounterSample):

    def __init__(self,
                 timestamp: datetime,
                 exits: int,
                 enters: int,
                 weather_data: WeatherData = None,
                 tags: dict = None):
        super().__init__(timestamp, exits, enters, tags=tags)
        self.weather_data = weather_data

    def populate_weather_data(self):
        self.weather_data = weather_instance.get(self.timestamp)
```

9.1.5 Data transformation utilities

In order to work with large datasets efficiently, several utilities were written. These utilities are used to understand the data, compute extra characteristics/features, and make training easier when working with local data. This way the full datasets can be used while our TSDB structure is not final or does not contain all data yet.

`group_measurements_by()` This utility can group measurements by interval (*minute, hour, day, week, month, year, or day of week*). Additionally, it can coarsen `PopulatedPeopleCounterSamples` (e.g. from minutely to quarterly), making it easier to test adjustments to prediction. Entrances, exits, and rain are summed, while temperature is averaged. This function also calculates the `total_enters`, `total_exits`, and `cumulative_sum_max` for each group, and `cumulative_sum`, `cumulative_enters`, and `cumulative_exits` for each sample. These cumulative values are from the start of the interval (e.g. day) up to the minute it is calculated for, but can be configured to never reset at the beginning of an interval.

`datasets_to_map()` This utility can transform the output of multiple `group_measurements_by()` calls into a dictionary with the minute as key (`dict[minute, dict[name, SamplesGroup]]`) and name with `SamplesGroup` as value. An example call would be

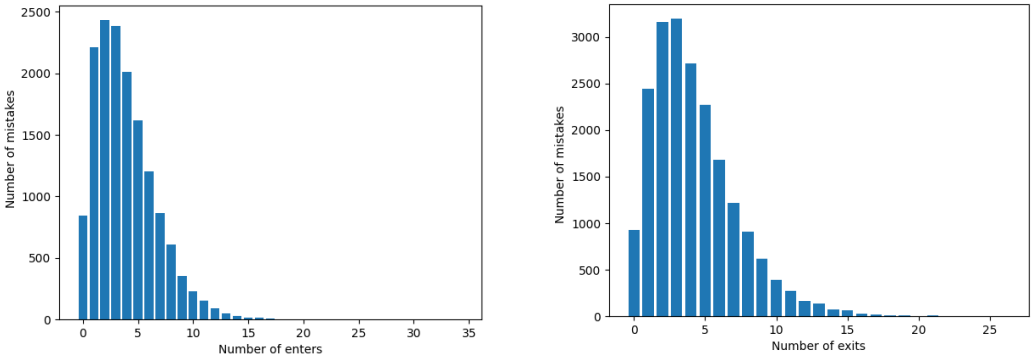
```
map = datasets_to_map({
    "old": old_grouped_samples,
    "new": new_grouped_samples,
})

# -> {datetime(2023, 1, 1, 1, 1): {"new": SamplesGroup, "old": SamplesGroup}}
```

`map_to_dataframe()` This utility transforms the previously created map to a Pandas `DataFrame`. It flattens the nested class structures to one level. The resulting `DataFrame` has a column `ds` (the timestamp), and duplicate columns for each dataset. The example above would result in columns `ds`, `old_enters`, `old_exits`, `old_cumulative_sum`, ..., `new_enters`, `new_exits`, `new_cumulative_sum`, ... This dataframe can now be easily used when experimenting with predictions.

9.1.6 Data correction

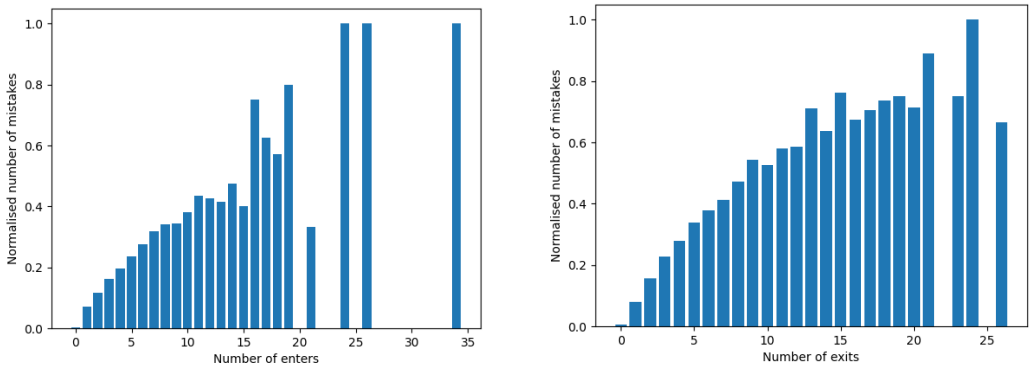
As discussed before in section 3.1, there is a big discrepancy between the data from the old Brickstream sensor and the new Vector4D sensor: the Brickstream sensor is less accurate. To make sure the old data is as correct as possible, we need to apply corrections to the data. The expectation was that mistakes would be made more often in minutes when there was a lot of movement (15 people entering compared to 3). To confirm this, we have counted the number of mistakes grouped by the number of enters and exits in that minute. This results in Figure 5. This distribution itself is not very useful, as it most likely only shows that there are more small groups leaving and entering the library.



(a) Enters (b) Exits

Figure 5: Number of times the old sensor differs from the new sensor

Only once we normalise the data by dividing by the total number of occurrences of these groups we can clearly see in Figure 6 that errors are more common in minutes with more activity proportionally, both for enters and exits.



(a) Enters (b) Exits

Figure 6: Number of times the old sensor differs from the new sensor (normalised)

The old sensor always records more exits and enters than the new sensor, indicating that we should only *reduce* the exit or enter counts. If, at the end of a day, the occupancy is `<0` we reduce the number of exits, and if `>0` the number of enters. Our algorithm for correcting the old data works as follows:

1. Determine offset at the end of the day. If negative, reduce exits, else reduce enters.
2. Order all measurements of the day by number of exits/enters respectively.
3. Subtract `1` from each minute, starting at the highest value, until the offset would be `0`.

Offsets were observed to often be between `-60` and `+10`. Considering there are 1440 minutes in a day, this algorithm will only modify the highest 10 to 60 measurements. If we were to subtract more than `1` from a minute (i.e. subtract based on the area below the normalised graph), we might create sudden gaps or spikes in the data when we plot the graph.

Instead, we try to smooth out the correction over as many minutes as possible (thus only subtracting 1). An example of this correction applied to sensor data can be seen in Figure 7. Here, `old_corrected_cumulative_sum` is the graph which has this algorithm applied. This is more in line with the new sensor's `new_cumulative_sum`. Note that the new sensor is not corrected to be exactly `0` at the end of the day, causing a slight difference at the end of the day.

Corrections like this can be implemented on a dataset level. For the Brickstream dataset, we overwrite the `get()` method, and provide an extra `applyCorrections` toggle to the constructor:

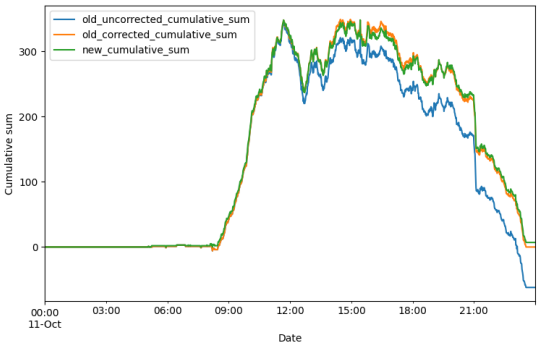


Figure 7: Occupancy sensor correction

```
# BrickstreamDataset

    def get(self, start_datetime: datetime, until_datetime: datetime) ->
    ↪ List[PeopleCounterSample]:
        samples = super().get(start_datetime, until_datetime)
        if self.apply_corrections:
            BrickstreamDataset.distribute_differences(samples)
        return samples
```

9.2 Weather Data

Our system uses temperature and rain data measured in Enschede, which is visualised on the dashboard (Figure 3, Occupancy over time), and can be used as a regressor for prediction. Historic data is sourced from the KNMI[2] and forecast data from Open-Meteo[3]. Neither for our dashboard nor prediction, it matters if weather data is historic or a forecast. By structuring historic and forecast data in a similar way, we can easily join the two datasets (a date range is provided which is first filled with KNMI data and then extended with Open-Meteo data) and store it in our database.

9.2.1 Historic weather data

The acquisition of hourly historic weather data is done through the KNMI website. To access this data, a POST request is made to the KNMI API, making sure to send along the correct parameters. Upon retrieval, the data is processed to increase efficiency. The raw JSON data is put into a Python dictionary, with the date and hour being the key and the weather data being the value. This key choice is made because a sensor measurement has a timestamp, thus the corresponding weather data can easily be retrieved. Transforming the JSON data into a dictionary structure increases the average lookup speed from `O(n)` to `O(1)`, as it does not have to loop through the entire data set. This speed increase is essential to our system, as millions of data points need to be processed. Furthermore, to reduce redundant API calls, the processed data is then stored in a cache on the file system. If weather data is missing from the cache (i.e. the last data is from more than two days ago), new data is fetched and the cache is updated. This caching mechanism eliminates the need to repeatedly access and retrieve the data from the KNMI website, improving speed and resource-efficient data management within the project.

9.2.2 Weather forecast data

This historic weather data is combined with forecast data. Our system makes use of the Open-Meteo API. Upon boot, the forecast is retrieved from Open-Meteo and loaded in memory, which is then updated hourly. Considering its relatively compact size (only a few days of data), forecast

[2]https://www.daggegevens.knmi.nl/klimatologie/daggegevens
[3]https://open-meteo.com/en/docs

data is not cached, reducing unnecessary complexity. The data provided by KNMI is one day behind, so to bridge this gap our system also uses data from Open-Meteo for these missing days. The data is subsequently processed into a dictionary structured similarly to the historical data to ensure efficient and fast access.

9.3 Academic Calendar Data

To compare academic years, it is not sufficient to compare identical dates. Although the date is the same, the academic context could be very different. For example, in the academic year 2020-2021 the 21st of December is a regular university day, whereas in 2021-2022 it falls in the holiday period. To take into account this difference, our system integrates the different academic calendars dating back to 2019-2020. By considering these calendars all dates can be compared with the right academic context.

9.3.1 Academic calendar

Since there is no standardized format for the academic calendar provided by the UT, it is necessary to manually transform this into a format our system can use. Therefore, we have created an Excel template file. Excel is an excellent choice for this project, as the LISA staff are already familiar with the software. Furthermore, reading Excel files in Python is straightforward using Pandas[4]. This data is transformed into a list, and from this list the system can easily retrieve all row information. The Excel file requires date intervals (start and end dates) to decrease the amount of manual labour. From this, we can gather all necessary information, without requiring someone to fill in every day of the UT calendar. A more detailed description of the Excel van be found in Chapter 11 in the Excel manual.

9.3.2 Holidays

To incorporate holidays, our system uses two sources. We use the Python holidays[5] library that contains all public holidays of The Netherlands. This library automatically determines the public holidays in a specific year. For all non-standard holidays, a LISA staff member needs to insert the dates in the academic calendar Excel file under the 'bridging days' column.

9.4 Recurring Tasks

Cron jobs are tasks performed at various intervals or situations through a piece of software called cron. It is built for Unix-based operating systems such as Linux. Through the crontab file, tasks are specified and when to run these tasks. We decided to utilize cron for our project because it is lightweight, specializes in repeating tasks, and is easy to configure.
To keep the database updated with the most recent data, there are a number of cron jobs that need to be run at set intervals according to the crontab file. These jobs are, however, designed such that they can be run on an empty database and retroactively insert missing data. A class diagram of all of the following jobs can be found in Appendix 22. Each utilises the InfluxDB classes and the provider related to their task.

9.4.1 Syncing sensor data - `/jobs/sync_peoplecounters.py`

Runs every 5 minutes - This updates the database with the newest sensor data. It does this by first querying the database for the last inserted datapoint and then using that date-time to retrieve all datapoints from the SensorData API from that moment until now, the datapoints are then parsed and stored as the `enters` and `exits` fields in the `people_counters` measurement.

9.4.2 Syncing weather data - `/jobs/sync_weather.py`

Runs every hour - This updates the database with historical weather and forecast data, from both the KNMI and Open-Meteo respectively. Historic measurements are marked with a tag `type=Historic`, while forecast has `type=Forecast`. Upon execution, the database is queried for

[4] https://pandas.pydata.org/
[5] https://pypi.org/project/holidays/

the last historic measurement, which is used as a start date to insert new data. Forecast datapoints are overwritten with historical data once available. Weather data is stored as the `temperature` and `rain` fields in the `weather` measurement.

9.4.3 Syncing daily max - `/jobs/sync_daily_max.py`

Runs every day at 04:05 - This calculates the max occupancy of days and updates the daily maxes prediction. The database is queried for the last stored daily max, which is used as a starting point. Next, a query is performed to calculate the daily maxes, which are stored as the `daily_max` field in the `people_counters_computed` measurement. We immediately use these new maxes to retrain our prediction model and predict the next 40 daily maxes. These are stored as `daily_max_prediction` in `people_counters_computed`

9.5 Prediction

As hinted at a few times before, we predicted the daily maximum occupancy in our system and provided this on the dashboard. Every time we inserted new daily maximums, we retrained the model and predicted all upcoming days again. To predict this, we thus provided our prediction implementation with all previous daily maximums and a date till which it should predict. These predicted values, along with upper and lower prediction confidence bounds, were stored along the `daily_max`es in our TSDB. In Chapter 10 we further discuss how we have implemented prediction for these maximum values. Technically, each day from now on, the model should get more accurate!

9.6 Technical Challenges

9.6.1 Requirements for optimal parsing performance

One of the core, and often recurring, issues we encountered during the project was that everything related to the data insertion needed to be as fast as possible. This was due to the fact that the entire dataset since 2019 contains millions of data points, so if there was an operation that needed to be performed on each data point that takes 1 millisecond, applying said operation to the entire data set could already easily take an hour.

Therefore, with everything related to parsing and combining the different data sources, we had to make sure everything was performant through the use of caching and in-memory maps. This ended up adding unexpected complexity and difficulty to the project. Some solutions we had to come up with were:

- Bulk-inserting data points into InfluxDB.
- Local file caches for the SensorData API and weather APIs.
- Ensuring data such as academic calendar is loaded into memory on boot.
- Ensuring we don't create file-handles in loops.
- Making use of key-value maps (`dicts`).
- Performing in-place operations where possible.

9.6.2 Relatively slow data API

The way that we retrieved the sensor data was through an API provided by LISA called SensorData. Since this API was not designed for large-scale data retrieval it also created some challenges to overcome.

The first issue was not being able to query data from a dataset which only contained the sensor from the library: we received all measurements of the same sensor type and then filtered out the specific sensor IDs which were not relevant to us. For example, for the Vector4D dataset, all of the data was about 15GB, while the relevant data for us was merely 200MB.

Secondly, the rate at which data could be downloaded from the SensorData platform was limited to 5MB/s, which compounded the previous issue. To solve both of these issues we decided to utilise a cache file, to which we wrote the measurements we received from the SensorData platform. This allowed us to retrieve and filter all the historical data once, which could then later be appended

with new measurements and loaded directly from the disk. This allowed us to work with the entire data set much more easily and quickly during development.

9.6.3 Issues with time zones

A common issue we encountered during the project was handling time zones. Various data sources had different timezone settings, which resulted in problems across different components of the project. These issues we encountered were:

- The Brickstream and Vector4D datasets had measurements with different timestamp formatting standards within datasets.
- Brickstream measurements contained date, time, and timezone data in separate fields, which required very specific parsing.
- Using Python's built-in `datetime.strptime()` method often resulted in missing timezone data.
- Prediction using Prophet could strip dates of their time zones.
- To query a whole day in local time, the influx query needed to know the computer's timezone.

These issues are resolved, however, it is important to note that the effects of these issues are often only noticeable when data is visualised on the dashboard, either looking shifted or the start of a day being displayed as 01:00 or 02:00. To reduce the occurrences of these issues, we agreed on handling timestamps as UTC in most cases as is best practice when dealing with time zones, and restricting the use of naive `datetime`s (`datetime`s with no timezone information); many functions refuse to work if a `datetime` without timezone is provided.

10 Prediction

As defined by one of the 'could' requirements of the MoSCoW prioritization strategy, the system could include some form of prediction. It was not a priority for this design project and thus we decided to investigate potential research in an exploratory way to assist further research.
Given the many data points we have received access to from people counter sensors in the past 4 years, a time-series prediction model could be used to predict how many people will be in the library for the upcoming weeks, or years. We have decided to make an attempt to envision the number of enters and exits for the people in the library. More importantly, our client was interested in the maximum occupancy per day for people in the library. This data could be used to show to ASITO (the cleaning company), or to higher ranks within the university to indicate whether the library should be opened during holidays. The library has always fought for the opening hours during holidays like Christmas since some people do not have the luxury of a place to study/relax. The predictions and data visualization give the library staff a solid foundation for supporting their statements.

10.1 Choosing a time-series forecasting model

In- and out movements at the UT-library can be considered a traffic flow problem. To specify how to create a forecasting model, some non-quantitative parameters will have to be defined:

1. Outside of the opening hours, there is no movement at the library entrance.
2. During opening hours, people can enter or leave at any given time, with data points given at a frequency of 1 minute.
3. During exam weeks, traffic flow is significantly higher both in weekdays and weekends
4. Weekends are significantly lower in library occupation than weekdays, given that it is not an exam week.
5. During holidays where there are no classes (Christmas, Summer holiday, etc.), library occupation is much, much lower.

According to Shuvo et al., 2021, who have done a thorough analysis of various time-series forecasting models, for traffic flow prediction, Prophet has the ability to create a model with an accuracy of 90%. But, in order to receive such a high accuracy, the data has to be following a seasonal pattern. Our time-series data can be observed to have high seasonal traits:

1. *Yearly seasonality*, The academic year is divided into 4 main quartiles that have a huge influence on when people put their most effort into studying, and hence, visiting the library. Besides this, there is also the summer holiday which has a huge negative influence on the number of people visiting the library.
2. *Weekly seasonality*, As discussed before, we observed that the weekends are less occupied than weekdays. Also, there seems to be an upward occupation trend toward Thursday, lowering on Friday again.
3. *Daily seasonality*, We observed that the biggest peak of library occupation is around 2 PM on most days. Before this, it usually linearly increases. Afterwards, it slowly decreases. If not separating the opening times from the model, occupation is 0 during closing times.

When looking at the time-series forecasting model FacebookProphet, it also takes the importance of 'trend' into account. Based on the data points, Prophet determines (various) trend lines and uses seasonality components, holidays, and additional regressors into account to determine a prediction value (yhat). All time-series forecasting models that we considered for the project, with their ups and downsides are discussed now.

10.1.1 ARIMA

For forecasting time-series data, ARIMA is another often-used approach (Shuvo et al., 2021; Subashini et al., 2019). ARIMA stands for Auto-Regressive Integrated Moving Average and uses *past values* and *past errors* for predicting at time t. ARIMA needs to be standardized such that the model becomes independent of seasonal trends and becomes stationary. According to Subashini et al., 2019, ARIMA is limited in some aspects:

1. The time interval between data points needs to be the same throughout the data
2. A day with not-assigned values is not allowed
3. Seasonality with multiple periods, such as year and week, is difficult to handle.
4. Parameter tuning needs to be done by an expert.

10.1.2 Prophet

Prophet is created for business analysts and people who want to make many forecasts, potentially without training in time series methods. Time-series forecasting is a difficult endeavor and analysts who can produce high-quality forecasts are rare since forecasting is a skill that requires substantial experience (Taylor and Letham, 2017). Prophet is created to be adjusted without any underlying knowledge of the model, with easy-to-tune parameters. So for people who know their data and the variables, it is easy to create a forecast.

To overcome ARIMA's shortcomings, Prophet has ways to counter all of these (Taylor and Letham, 2017). For the time-series data we are using for prediction, the time interval between data points is usually always the same (1 minute). However, when excluding the closing time minutes from the data, the interval between data points can differ (over 8 hours). As we will explain later, some days are excluded from analysis since the patterns during these days are outliers when compared to an 'average' day. Prophet is specialized for modeling time-series data with seasonalities and includes relatively simple parameters that can be tuned.

Prophet is an additive model consisting of four components:

$$T(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{1}$$

$T(t)$ describes the predicted value (y) of the additive model, on time point t. $g(t)$ describes the trend in the data, and is the most general, foundational value of the model. $s(t)$ are the seasonal effects on time t, and describe how daily, weekly, or yearly effects have an influence on the model. When looking at parameter tuning later, the seasonal effects can also be set to *multiplicative*, which implies that it will be multiplied with the trend value at time t. $h(t)$ is the holidays or events component, since these can have a big influence on the prediction too. ϵ_t stands for the fluctuations that cannot be explained by the model. ϵ_t follows a normal distribution, where the mean and standard deviation are derived from the data.

10.1.3 NeuralProphet

NeuralProphet, as explained by the developers, is the bridge between traditional time-series forecasting methods like Prophet and Deep Learning methods, like LSTM or DeepAR ("Overview of the NeuralProphet Model - NeuralProphet documentation", n.d.). In comparison with Prophet, NeuralProphet does better in terms of adapting to the local context of predictions and the support for auto-regression and covariates. However, during development, it turned out that there was barely any correlation between a value y_t and $y_{t-2}, y_{t-3} \forall t$, given the auto-correlation graph produced by all data (see Figure 8). Furthermore, NeuralProphet yielded far lower accuracy scores than Prophet for our data, given that we do not use the concept of auto-regression and covariates. Hence, a choice to use the traditional Prophet was made.

10.2 Variables affecting prediction

We have identified quite a number of variables that have a significant effect on the prediction model.

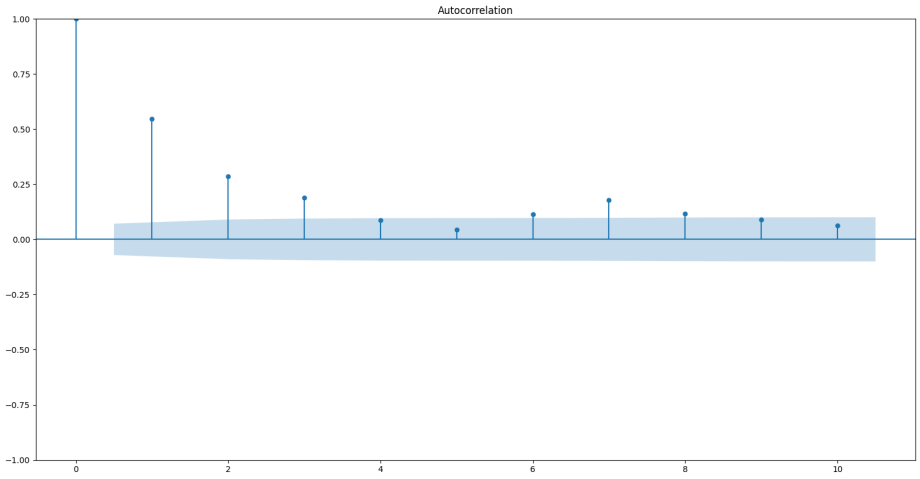


Figure 8: Auto-correlation graph created on the residuals graph of a NeuralProphet prediction, on the maximum occupancy per day.

Codependency of enters/exits Logically, the number of people entering the library and the number of people exiting the library are co-dependent on one another. The sum of exits of a day can for example never be higher than the sum of enters at any point in time. This, as is discussed in further sections, can cause some complications when predicting these values separately from each other.

Opening hours As we found out, the opening hours of the library change quite frequently, besides more restricted opening hours during the weekends, the library also often changes the opening hours based on the proximity to exam weeks. On top of that, there are often "special" days. For example for open days or festivals on campus which impact the opening hours.

Holidays Besides the aforementioned special days, there are also quite a number of public holidays in the Netherlands where the library is sometimes closed or has reduced opening hours. This has significant effects on the prediction model if not handled correctly. Since Prophet is an additive model, and has the function $h(t)$ for adding a holiday value to the final prediction value, we can tackle this by adding all holidays to the model, so Prophet can decide what impacts holidays have on the model (usually leading into fewer people visiting the library, hence giving it a negative value).

Weekdays vs. weekends Not every week is equal for students, many students tend to visit their parents or choose not to study during the weekend, which results in both less movement and a lower maximum occupancy during the weekend. Even Friday is affected by the same effect. This is a reasonable conclusion to assume, as by simply looking at any typical week this effect can be seen. However, it is not sure that this behaviour does indeed result in the observed effect and we suggest further research into this behaviour.

Number of students One variable that we ended up not included in either the prediction or the data was the number of students attending the university at that time. This was mainly done because we did not have access to a reliable source of student numbers at any given moment. Although yearly numbers can be accessed, numbers on a weekly or even daily scale are not possible to get. However, it can potentially have some effect, as there might be a significant number of students who drop out somewhere during the academic year. Also, market research on what kind of students usually attend the library is a recommendation. If hypothetically all studies have an equal percentage of students at the university, this probably means that in the library, not the same number of students of each study attend. Maybe psychology students are more likely to join

the library than computer science students. Of course, not all studies have the same number of students, so when this changes, this may have an impact on the library occupancy.

Weather Our dataset includes weather data for the region in and around the University of Twente, in the form of rain and temperature hourly. Because movement from students can be classified as utilitarian movement, it is less impacted by the weather Jonkeren, 2020 thus we decided to not include this in the prediction model due to it raising the complexity significantly.

10.3 What to predict

The available data columns after the pre-processing steps done by our team are displayed below. We are using data from 2019-09-01 until 2020-03-01 and 2021-09-01 until now. This time range excludes weird data

The variable names speak for themselves. Data is available each minute, which means that in the 2.5 years of data points we use for training, we have approximately 1.3 million minutes available. However, nearly one-third of this data should be excluded, since the library is closed during those minutes.

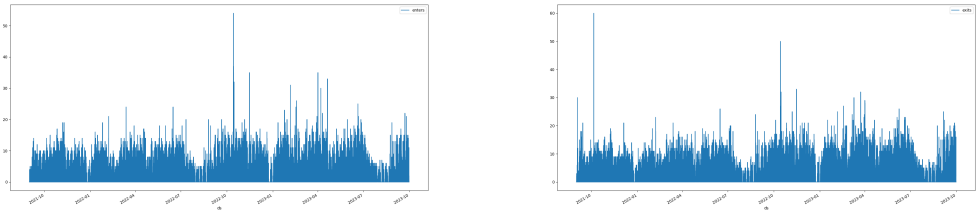
```
>>> import pandas as pd
>>> df_loaded = pd.read_csv('csvs/prediction_data.csv', parse_dates=['ds'])
>>> df_loaded.columns
Index(['cumulative_sum_max', 'cumulative_sum', 'cumulative_enters',
       'cumulative_exits', 'enters', 'exits', 'day_academic_day',
       'day_holiday', 'weather_temperature', 'weather_rain', 'ds'],
      dtype='object')
```

The `ds` column is defined as the date string and is required by Prophet or NeuralProphet. The value to predict must be defined as `y`. Given the columns, we can set `y` to the desired prediction value. The most self-evident variables for this would be `enters`, `exits` or `cumulative_sum_max`

10.3.1 Movement: Enters and exits

To predict *enters* or *exits* as defined in the pandas data frame, we need an excellent time-series prediction model since values per one single minute are always integers, and usually in a range from 0 to 20, but with outliers sometimes going up to 54. The average number of people entering per minute when this value is at least 1, is 2.6. For *exits*, the values are comparable.

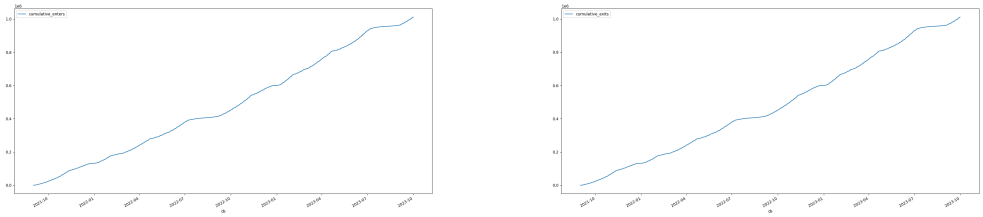
Since the enters and exits graphs specified in the above manner do not follow a regular pattern that Prophet finds easy to predict, we are switching to *cumulative_enters* and *cumulative_exits*. See Figure 9 for plots of enters and exits per minute. When comparing this with Figure 10, we can see more clarity in its seasonal trends. The gradient of the line is lower where the number of enters or exits is lower, and higher where the number of enters and exits is higher.



(a) Enters (b) Exits

Figure 9: Enters and exits per minute over a span of 25 months.

Since we observe that the cumulative enters and exits follow patterns that can be expressed in Fourier terms, this is perfect for a model like Prophet, which models seasonalities in such. You can see such patterns in the Appendix, in Figure 26.



(a) Enters (b) Exits

Figure 10: Cumulative sum of enters and exits per minute over a span of 25 months.

10.3.2 Occupancy: Maximum number of people on a day

Our client was really interested in receiving a prediction for the maximum occupation per day. This way, they get a good insight into how many people will join the library on that day and hence can arrange certain occupation measures. The raw maximum occupancy data is shown in Figure 11.

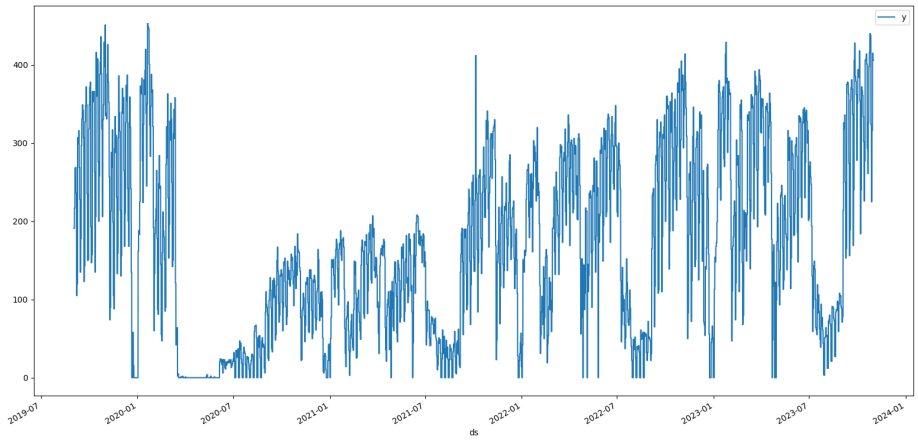


Figure 11: Maximum occupancy values of the past 4 years.

Without the COVID19 period, which took place from 2020-03 until 2021-06 for the UT-library, data follows obvious seasonal trends. The downside to this prediction approach, is that there are less data points available, since you only have 1 data point per day.

10.3.3 Timeline

Let us start this timeline section, which explains what choices were made throughout the weeks of development, with mentioning that the client's needs came first in any case. After the old data was corrected, there was enough data to start on the prediction phase. The first step was to look into a time-series forecasting model which required little to no previous experience in prediction. The model should also adapt to seasonalities, and should be able to handle regressors, like temperature data. NeuralProphet was decided as the model to use. In collaboration with the client, we discussed that it should be possible to predict enters and exits on a minute scale, so we focused on that. After issues with NeuralProphet and the prediction of enters and exits, we decided that we should focus on a broader picture, like the maximum occupancy per day. The UT-library would be very happy with this, since an overall view of the upcoming months would be shown. After issues generating a prediction in NeuralProphet for these very few data points (800), we switched to Prophet to see whether or not this would suffice. The prediction of enters and exits was also done with Prophet,

but this lead to bad results. This all lead to the pure focus on the maximum occupancy per day, and understanding the variables, to lay down a foundational basis for upcoming research projects that want to focus on this prediction aspect.

10.4 Parameter tuning

According to the documentation of Prophet, "Diagnostics", 2023, the parameters of the model as shown in Table 4 are open to being tuned. We used a hyperparameter tuning method that uses cross-validation splits in order to obtain the most accurate `changepoint_prior_scale` and `seasonality_prior_scale`. Values that were tested can be found below.

```
param_grid = {
    'changepoint_prior_scale': [0.001, 0.01, 0.1, 0.5],
    'seasonality_prior_scale': [0.01, 0.1, 1.0, 10.0],
}
```

For the results, the Prophet model was fit in the following way:

```
from prophet import Prophet

holidays = load_holidays()

model = Prophet(holidays=holidays,
            daily_seasonality=False,
            weekly_seasonality=True,
            yearly_seasonality=True,
            changepoint_prior_scale=0.01,
            seasonality_prior_scale=0.1,
            seasonality_mode='multiplicative')
```

10.5 Preprocessing

The data was loaded from a CSV file. Since both Prophet and NeuralProphet need a *ds* column with date strings, and an *y* column with the value to predict, this had to be arranged in Python. As shown in Figure 11, the COVID-19 period is not similar to more recent values, and hence, these are also removed.

```
import pandas as pd

# Read CSV file
df_raw = pd.read_csv('csvs/prediction_data.csv', parse_dates=['ds'])

# Convert the date strings to datetimes that Python understands.
df_raw['ds'] = pd.to_datetime(df_raw['ds'], utc=True).dt.tz_convert(None)

# 'cumulative_sum_max' can also be 'cumulative_enters' or 'cumulative_exits',
↪  depending on what values to predict.
df_raw['y'] = df_raw['cumulative_sum_max']
df = df_raw[['ds', 'y']]

# Remove the COVID-19 period where data is not representing current behavior,
↪  also remove the Christmas break from 2019, where library opening hours were
↪  very different from what they are now.
df = df.loc[(df['ds'] <= datetime.date(2019, 12, 15))
```

Table 4: Prediction parameters and their function.

Status	Parameter	Explanation
Likely to tune	`changepoint_prior_scale`	Determines the flexibility of the trend. Default is 0.05, and anything in between [0.001, 0.5] is okay.
	`seasonality_prior_scale`	Controls the flexibility of the seasonality, similar to `changepoint_prior_scale`, a larger value allows the seasonality to fit large fluctuations, and a small value shrinks the magnitude of the seasonality. Reasonable tuning range is [0.1, 10]
	`holidays_prior_scale`	Controls flexibility to fit holiday effects, also on a range of [0.1, 10]
	`seasonality_mode`	Either additive or multiplicative, in relation to the trend
Maybe tune	`changepoint_range`	Proportion of the history in which the trend is allowed to change
Unlikely to tune	`growth`	Either linear or logistic, but automatically decided by Prophet
	`changepoints`	To manually specify the trend changing points
	`n_changepoints`	The number of automatically placed trend changing points
	`yearly_seasonality`	Automatically decided whether to use or not if there is enough data
	`weekly_seasonality`	Same as `yearly_seasonality`
	`daily_seasonality`	Same as `yearly_seasonality`

```
15                  | ((df['ds'] >= datetime.date(2020, 1, 15))
16                  & (df['ds'] <= datetime.date(2020, 3, 1)))
17                  | (df['ds'] >= datetime.date(2021, 7, 1))]
```

Based on the data, Prophet makes up a trend and seasonalities. These can be seen in Figure 12. Also, the values that Prophet assigns to the given holidays can be seen in this figure. The holiday and events dates that were loaded into the Prophet model can be found in `prediction/prophet_test/load_holidays`.

10.6 Results

Findings on the prediction for traffic flow, and findings on the prediction for the maximum occupancy will be shown in this section. The many challenges and doubts that we encountered during research and the prediction phase will be covered in the 'Discussion and Future Work' section.

10.6.1 Maximum occupancy

At the start of the prediction phase, NeuralProphet was used. However, after careful parameter tuning and the usage of autoregression, the prediction of 365 days was way off, since it did not follow a regular yearly pattern, see Figure 13. The parameters for NeuralProphet here were similar to those of regular non-neural Prophet. Why NeuralProphet is so off in this prediction, remains a question with an unknown answer.

When using regular Prophet, with parameters as shown earlier, seasonalities were defined properly by the model. Leading to the result as shown in Figure 14.

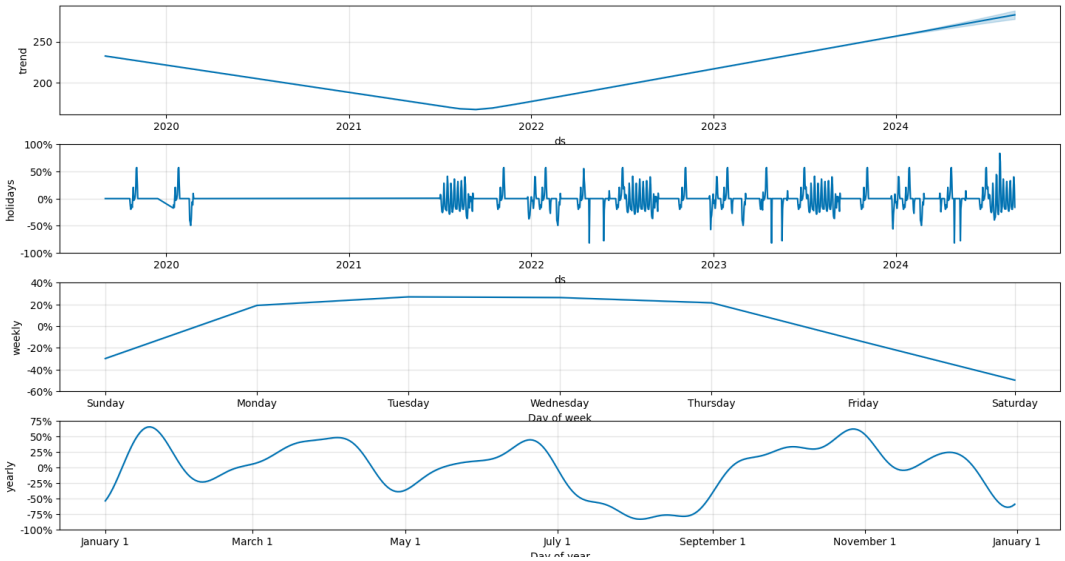


Figure 12: Individual prediction components created by Prophet.

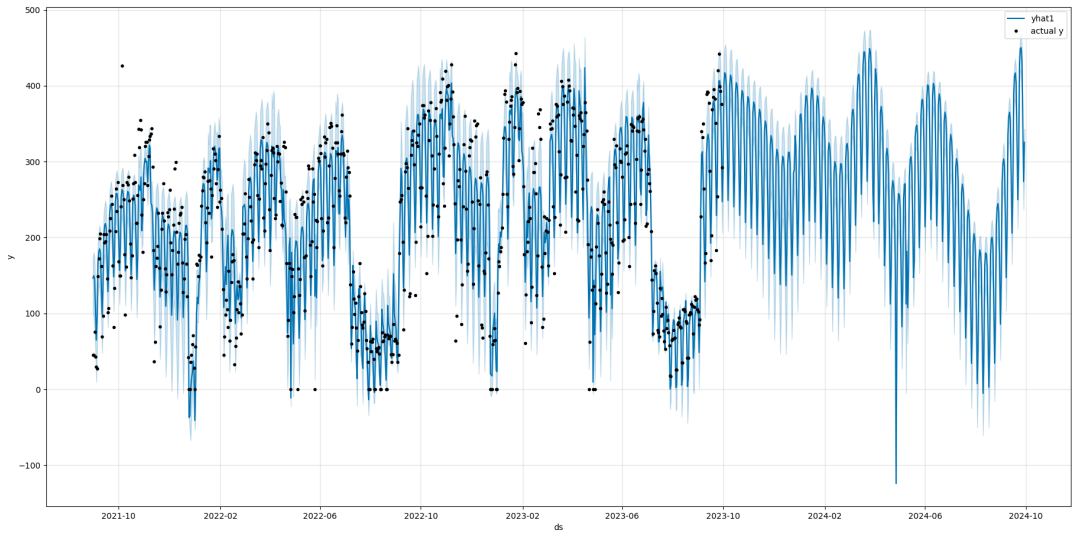


Figure 13: Maximum occupancy prediction done by NeuralProphet. The black dots are historical max occupancy values, and the blue line is the actual prediction.

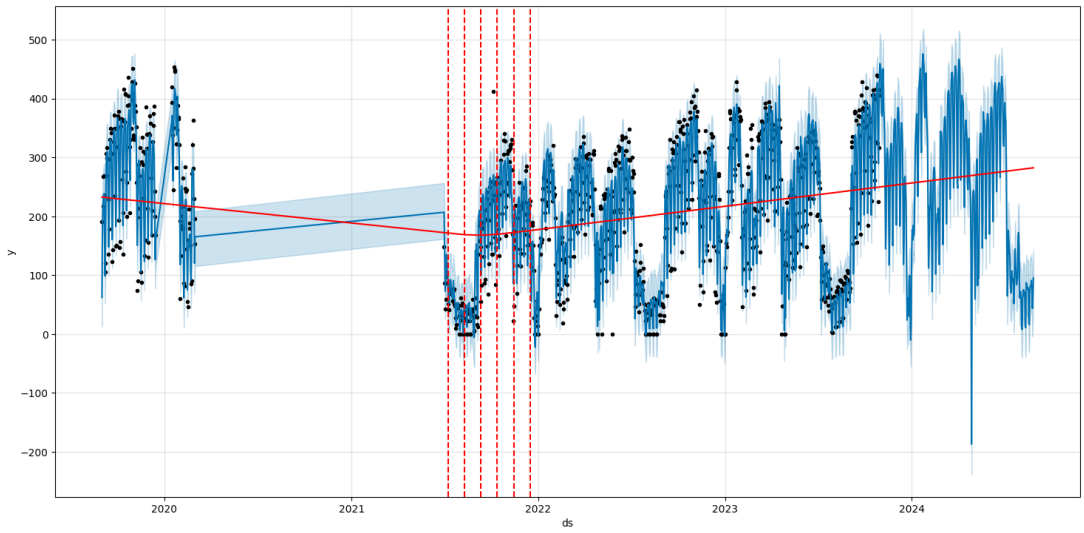
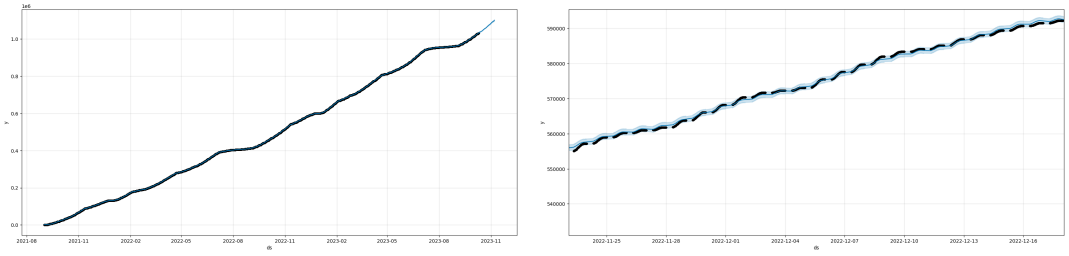


Figure 14: Maximum occupancy prediction done by Prophet. The black dots are historical max occupancy values, and the blue line is the actual prediction.

10.6.2 Enters and exits

The prediction of enters and exits was done with the data as shown in Figure 10. Only results on enters are shown in this section, and no diagnostics were produced, since the main focus shifted to maximum occupancy. The prediction is shown in Figure 15. Closing times were removed from the model, since values are always 0 here. When converting these upward going graphs back to daily data, we can see a comparison between actual values and predicted values. These can be seen in Figure 16. As you can see, is Prophet predicting values that are too high during the weekends, and values that are too low during the weekdays. See the Discussion for a possible explanation.



(a) Enters (b) Exits

Figure 15: Prediction of the cumulative sum of enters per minute, with a closer view in the right figure.

Based on the maximum occupancy prediction, cross prediction was done, and a root mean square error for each day in the future was calculated. Based on this, we conclude that the accuracy of the model is too insufficient to rely on. See Figure 17. The figure shows that when predicting 25 days in the future, the average error will be around 50 people. We have modeled this in the Grafana dashboard with a `lower_bound` and `upper_bound`.

```
from prophet.diagnostics import cross_validation, performance_metrics
import matplotlib.pyplot as plt

df_cv = cross_validation(m, initial='365 days', period='45 days', horizon = '90
    days')
plot_cross_validation_metric(df_cv, metric='rmse')
```

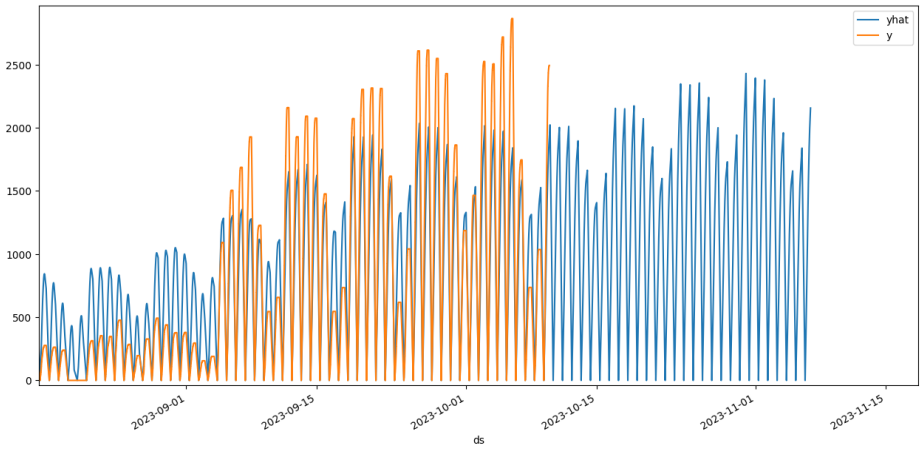


Figure 16: The predicted enters converted back to a daily graph, resetting to 0 at midnight. With the orange graph being the actual values and blue the predicted values.

```
plt.show()
```

10.7 Discussion and future work

In this section we go over the limitations that were found during the prediction phase. Based on this, some future work is suggested.

Predicting enters and exits Shuvo et al., 2021 have shown that Prophet can achieve high accuracies for usual traffic-flow datasets. However, classical traffic flow problems only deal with one prediction value, and do not measure or care about the in **and** out traffic. In an ideal situation, one would predict the enters and exits, and subtract them on a certain time interval, like 1 minute, 5 minutes, or 15 minutes, to get an actual indication of movement on that time interval. However, since there is no way for an exits model to be aware of how many enters would take place on that certain time interval, this can lead to both models predicting considerably different activity for a day, that would be impossible in the real world: a person who enters will always leave. To solve this problem of codependency between enters and exits, research could be done to figure out what forecasting models can adapt to this.

Weekends and weekdays As shown earlier in the report, occupancy and traffic flow in weekends differs dramatically from weekdays. Also, no classes are taking place, so the concept of 'breaks' does not exist in weekends. Students go home in the weekends, so there is overall less occupancy. A recommendation would be to completely separate weekends and weekdays into two different prediction models, and only merge them later. When doing this in Prophet or NeuralProphet, results were worse, so a different model could be explored for this purpose.

Analysis of individual days Only later in the prediction phase, it occurred to us that an average day follows a certain pattern that can be explained by many variables. For example, university breaks, like the lunch break between 12:30 and 13:45 have a significant impact on the occupancy in the library. A thorough analysis on day-to-day behavior and analysis on when people decide to go to the library would be recommended. It is also important to keep in mind that people usually leave their stuff in the library when going on a break, so even though occupancy shows lower values, not all tables are available. This may be an important variable to consider when displaying this information to students.

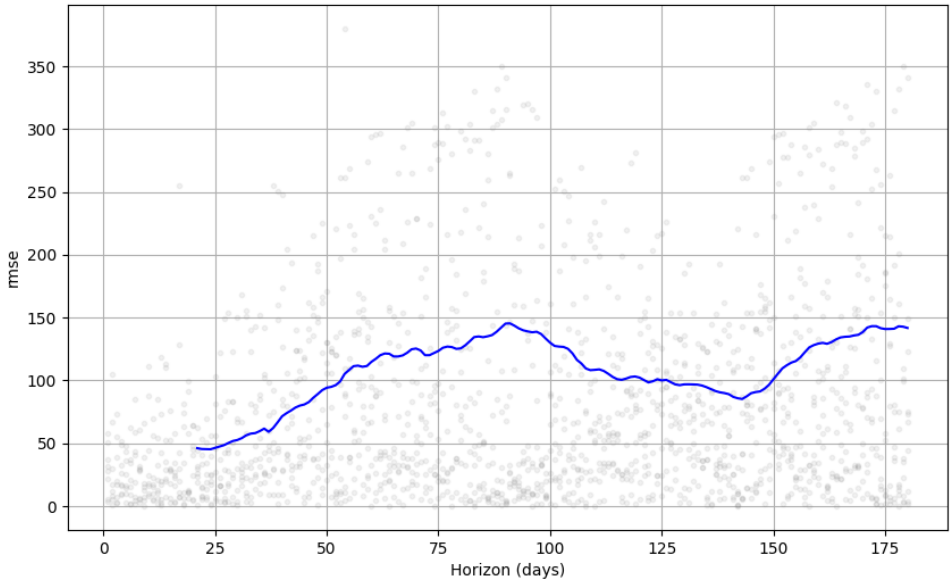


Figure 17: The root mean square error displayed on the cross validation splits on the maximum accuracy prediction model.

More data and regression The data we have now did not allow any form of regression. Regressors also require data in the future, so having 'exits' as regressor in the 'enters' prediction is impossible to do. Having data such as the number of opening hours per day, the number of students currently studying at the university, or the number of students that have tests upcoming, could be valuable for prediction.

General Recommendation The variables as shown in Section 10.2 can be used as a baseline for any prediction model regarding this data. We would suggest to further improve on Prophet if the reader is not an expert in data prediction, but understands the data. Other models, as explained by van der Bijl et al., 2022 should also be looked at, for hourly and daily-aggregated data points. Overall, other, more complex models should be tried on the data, like TBATS or LSTM models.

11 Manual

11.1 Installation instructions

There are various ways to install the software for this project. It can be done by cloning the repository from the UTwente Gitlab and following the instructions as found in the README.md, or it can be done by deploying the Docker container. Regardless of the chosen way of installation, a Grafana and an InfluxDB instance are required to make this project work as intended without any errors.

11.1.1 Docker

The recommended way of building and deploying the project is through a Docker container. The benefit of using Docker is that you do not have to worry about the dependencies and it can be deployed on any machine. The project comes provided with a dockerfile and a docker-compose.yaml file that provides the necessary building blocks to deploy the project with Docker.

Local deployment To deploy and build the Docker container, the docker-compose.yaml can be used. The first step is to fill in the missing environment variables. These are needed to ensure proper connection to the various data sources we use. Once these values are provided, running `docker compose up -d ubvision` in your terminal will start building the docker image and start the needed dependencies.

Environment variables A small issue that one might encounter is that the environment variables are not properly set up for the cron environment to use. Although the docker container's endpoint should normally handle this, it will not be always the case. It is therefore recommended to run a small command in the docker container's terminal/shell such that this will be the case. The command is `printenv >> /etc/environment`. This ensures that cron will be able to use the same environment variables as have been set during the setup.

Linux deployment Since the Gitlab repository contains a pipeline that builds the docker image for you, a simple deployment to any server can be done. First, you have to pull the image from its registry. The location of the registry can be found in the repository. After pulling, the environment variables can be set up and the container can be started. Once the container is started, please check what is described in the paragraph above about environment variables.

Kubernetes To deploy the software on a Kubernetes cluster, it is recommended to fill in the variables through a secrets yaml file. The project does not come with a Kubernetes helm chart but if needed, help can be given in setting this up.

11.1.2 Grafana & InfluxDB

Grafana and InfluxDB are two dependencies needed to run this project. By default, InfluxDB is set up through the docker-compose.yaml file when trying to build and deploy the main docker image for the project. However, due to the project needing an InfluxDB API key, it is highly recommended to first start the InfluxDB image. Once an API key is retrieved, this value can be added to the UBvision docker settings such that the project can connect with the time series database. Once the installation is done, create a bucket and use that as the `INFLUXDB_BUCKET_COMBINED`.

Grafana is used for visualizing the data and can be deployed through the docker-compose file as well. Because our client (LISA) already uses Grafana, it is not listed as a dependency for this project.

To set up Grafana locally, run `docker compose up -d grafana` and once it is running set a password. Once Grafana is set up, add the InfluxDB instance as a data source. To create the dashboards, copy the JSON configuration for each dashboard you want from the dashboards folder found in the project repository. Then import these in Grafana with the InfluxDB datasource. For the final step, set the dashboard into read-only mode by going to the settings for the dashboard (the cog-wheel) and clicking on read-only. Refresh the dashboard and you should be done.

11.1.3 Local files

The project uses a few local files for data insertion. These files have been added to a persistent volume mount. It is recommended to map this volume to an easily accessible folder such that changes can be made with relative ease. If new dates are added then they will automatically be added to the database.

11.2 Excel Manual

To utilize academic calendar data, our system uses an Excel file that ought to be updated at the start of every academic year. To update the Excel file, expand the table downwards and fill the columns. Every row represents a quartile, corresponding to the academic year. Furthermore, the rows are filled with quartile information such as start and end dates, exam weeks and holidays. Public holidays do not need to be inserted manually, as our system already collects them automatically. The Excel file contains the following columns that need to be filled:

Table 5: Excel sheet columns and their description.

Column name	**Description**
Academic year	The academic year of the calendar
Start date	Start date of the academic calendar
End date	End date of the academic calendar
Quartile	The quartile number (One through five)
Start quartile week	Start date of the quartile
End quartile week	End date of the quartile
Start exam week	Start date of the exam week
End exam week	End date of the exam week
Start holiday week	Start date of a holiday week (Holiday periods that last for a week or longer)
End holiday week	End date of a holiday week
Bridging days	All dates that do not fit into a holiday week interval, but are holidays at the UT

To update the Bridging days column, simply expand the table downwards and fill in the bridging days that do not occur in a holiday week period. Note that bridging days can be named differently in the academic calendar. For instance, in the 2022-2023 academic calendar, the 'Kick-In' holidays should be inserted into the bridging days table.

11.3 Using the dashboard

Once you are logged into Grafana using the provided credentials. You have two dashboards to choose from, which can be found by opening the menu on the left and selecting the dashboards page. From there you will be able to select either the live or the historic dashboard.

11.3.1 Selecting the time range

From any dashboard, the time range in which the graphs show data can be modified with the dropdown on the top-right. As can be seen in Figure 18

Once you open the menu, either, one of the pre-defined time ranges can be selected on the right. Or an exact time range can be manually set on the left. This last one can be done by either

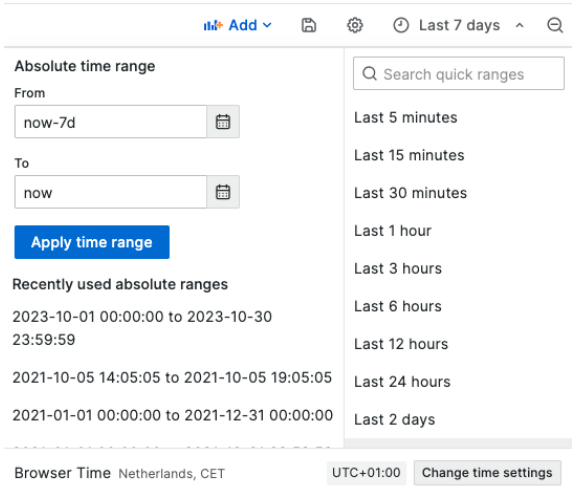


Figure 18: Time range selection menu

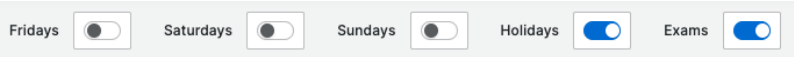


Figure 19: Special day highlight menu

clicking the calendar icon, or for example inputting "now-7d" which would represent the current date minus 7 days, as can be seen in Figure 18.

11.3.2 Highlighting special days

Weekdays, Holidays and exam weeks can be highlighted in the graph by enabling the toggles on the top of the dashboard (Figure 19)

11.3.3 Extra tips

Zooming into graphs Some graphs can be zoomed in if you for example want to see a particular day in more detail. Double-clicking allows you to zoom out again gradually.

Disabling parts of a graph Some graphs have multiple overlapping lines, if you only want to see a particular line, you can click on it on the bottom legend of the graph to toggle everything else off. Furthermore, holding control while clicking the various graph lines in the legend allows you to for example only disable one of the lines

Do you want to save your changes? Although the dashboard should be set in read-only mode, sometimes grafana will ask you to save the changes to the graphs once you try to close the page. We think it would be best to generally not do this if you don't know what you have changed. As it might cause something to break because of it.

12 Evaluation

12.1 Design process

The first point we want to evaluate is our design process. Although we had an initial idea of what to do, it was very chaotic and our ideas changed constantly. This made it difficult for some to keep track of what was going on and what the current plan was. This resulted in various design philosophies being used which resulted in only more chaos. We also noticed that the scope of our project kept expanding during the first weeks.
Luckily as the project progressed, more structure was introduced and we were able to set a realistic goal and reduce the tasks back to a manageable amount. For future projects, we have learned that we should first more thoroughly investigate what our ideas and options are after specifying the scope.

12.2 Client and supervisor communication

12.2.1 Client meetings

Meetings with the client were scheduled on a weekly basis, as we wanted continuous input from them to ensure that the end result would be satisfactory. At the start of the project, the goals of these meetings were to gather the functional and non-functional requirements of the projects. Alongside the requirement gathering, we also used these meetings to communicate with LISA what we needed from them in terms of sensor data. Halfway through the project the focus shifted more towards feedback sessions and showcasing our progress. For these meetings, we prepared presentations and certain drafts of the product, where LISA could give feedback on certain design choices. This feedback could include certain visual changes to the dashboard or more functional feedback such as suggesting different graphs.

12.2.2 Supervisor meetings

Meetings with our supervisor Yanqiu were scheduled on a bi-weekly basis at the request of our supervisor. The first meeting was mostly about getting to know each other and the project, and determining that she was willing to be our supervisor. Fortunately for us, this was not an issue and we could soon begin planning more functional meetings. These first meetings were focused on assessing the academic level of our project and discussing our project proposal. After these initial meetings, we could start working on the project and showcasing our progress similarly to our client meetings. In these meetings, we could also discuss the more technical details of our system, mostly regarding the prediction model. These meetings were very useful for us. The field of data science and prediction models is very huge and the expertise of our supervisor was very helpful.

12.3 Team collaboration

Another challenge in the project was collaborating together with a relatively large group of 5 people.
As explained before, for this we used a number of tools, like Trello, Gitlab, and Google Drive. However these tools are not perfect, so we still encountered some issues. For example, Trello only works well if everyone uses it religiously. However, during our chaotic start this got lost in the noise and resulted in the Trello not being updated that often.
At the start of the project, it was admittedly a bit chaotic, back then we had to decide on what technologies to use to build our dashboard, so while a few team members were experimenting with a number of database and dashboard systems, other team members were a bit left in the dark. This did result in some friction among the team.
Furthermore, some small issues in collaboration in the project emerged further on in the project when dividing tasks, sometimes no one felt "responsible" for a task, and sometimes work was not equally divided among all team members.
However, overall we managed to push through, tasks were properly divided, communication improved and we successfully finished the project and delivered a product that the client was satisfied with.

12.4 Week planning

This section includes the definitive planning of the team. The team could work on visualising, correcting and predicting the data in parallel, thus the planning shifted greatly.

Table 6: Tasks throughout the module weeks

Week nr.	Description of tasks
1	Setting up requirements, finding a supervisor
2	Finalising requirements in project proposal
3	Exploring what technologies to use
4	Build parses to easily process the data, test plan
5	Insert data into the database, gather external data and build initial dashboard
6	Correcting historical data, start on prediction and expanding dashboard
7	Expanding dashboard and work on prediction
8	Work on report, fix final bugs and prediction
9	Work on report, make poster and presentation
10	Final touches

12.5 Responsibilities

Throughout the project, the actual responsibilities ended up deviating a bit from the initial planning we made. As we discovered tasks were sometimes better suited to other team members due to having other expertise.

12.5.1 Rik

As planned, Rik worked mainly on connecting weather and academic calendar data. He tested different sources, assembled academic data from various sources, and determining formats in which this data can be used in our system. Finally, he also worked on a significant part of the final report, the poster and chair presentation.

12.5.2 Indy

As planned, Indy focused on processing the data retrieved from LISA's SensorData platform into usable formats and provided transformation utilities. Furthermore, he also sorted out discrepancies in data, streamlined and optimised most of the codebase, implemented Brickstream data correction, optimised graphs and worked on parts of the report.

12.5.3 Pieter

As planned, Pieter focused on retrieving the data from LISA's SensorData platform. Furthermore, he also worked on performance tuning, creating graphs, bug-fixing code and worked on large parts of the report.

12.5.4 Joris

As planned, Joris focused on inserting the data in our database and ensuring that the needed infrastructure was properly set up to be used by our project. Furthermore, he also worked on creating graphs in the Grafana dashboard, communicated and presented to the client and worked on various parts of the report.

12.5.5 Mathijs

Mathijs ended up focusing mainly on prediction and worked on the prediction parts of the report. This was because it turned out that prediction was more tricky and difficult than initially thought.

12.6 Future work & improvements

As this project ended up mainly focusing on combining and correcting all sensor data from a number of sensors into a single database, we have identified future work and improvements that can be done to build on top of this.

Our own system can be improved in several ways. First, there is data redundancy in the way academic day details are stored. Each `enters` and `exits` field in our TSDB share the same tags each minute that may be better of stored separately. Later, this is copied to `daily_max`, which is prone to errors, and makes it harder to correct this information if it turns out to be incorrectly inserted in the academic calendar Excel file.

Secondly, adding various new visualizations for the dashboard. Although we already offer a variety of graphs to show the data, research can be done on how to improve this. We worked closely with LISA in identifying the current sets of graphs such that they understood it the best, however, we do acknowledge there is a lot of room for new insights and ideas.

This could potentially be in the form of a research project, which uses advanced machine learning and is able to more accurately predict the number of students that will enter the library in the future. Although we did some initial work in this field, another project could expand on our work to investigate this further.

Another (design) project that could be created on top of this is integrating the data with a more sensors, like sensors for detecting when the garbage bins are full, desk occupancy and room sensors. This could allow for more accurate date predictions and new ways of visualising this data. Currently, our project only focuses on the entrance sensor, however, more kinds of sensors are located in the UT-library and more are being added.

Finally, we feel that another project which focuses on visualizing the occupancy of the UT-library for the students could be beneficial for everyone.

12.7 Conclusion

Although at times chaotic, the team worked well together and there was great chemistry. Team members could always discuss personal and project related issues, and the group would accommodate for them accordingly. All of the module deliverables, be it optional or obligatory, were handed in on time without major crunch or stress. This resulted in a relaxed but efficient work atmosphere where everyone could contribute optimally.
As for the goals of the 'Design project' module, we are very satisfied. We have delivered a working product that includes all requirements set in Sections 4.5 and 4.6. Another important aspect is the client of course. LISA was also very pleased with our product and already wants to use the project in its current state. We believe that this project is a great addition to LISA's arsenal and can aid them greatly with their data-driven decision making goals. Furthermore, the project is a great basis for future design and research projects alike, as discussed in Section 12.6.

13 Closing words and acknowledgements

We enjoyed doing this project however, it would not have been possible for us to complete it without the help of the following people:
Olga Steen, Lisalotte van der Tas, Lorna Jutton, and Jeroen van Ingen Schenau for giving us this project case and providing valuable feedback and insights into our work.
Our supervisor Yanqiu Huang for being there for us when we had questions regarding predictions and other valuable feedback.
And finally our Design Project module teachers for guiding us through this project.

References

Diagnostics. (2023, October). Retrieved October 6, 2023, from http://facebook.github.io/prophet/docs/diagnostics.html

Jonkeren, O. (2020). *De invloed van het weer op de personenmobiliteit*. Ministerie van Infrastructuur en Waterstaat.

Overview of the NeuralProphet Model - NeuralProphet documentation. (n.d.). Retrieved October 6, 2023, from https://neuralprophet.com/science-behind/model-overview.html

Shuvo, M., Zubair, M., Purnota, A., Hossain, S., & Hossain, M. I. (2021). Traffic Forecasting using Time-Series Analysis. https://doi.org/10.1109/ICICT50816.2021.9358682

Subashini, A., K, S., Saranya, S., & Harsha, U. (2019). Forecasting Website Traffic Using Prophet Time Series Model. *International Research Journal of Multidisciplinary Technovation*, *1*, 56–63. https://doi.org/10.34256/irjmt1917

Taylor, S. J., & Letham, B. (2017, September). *Forecasting at scale* (tech. rep. No. e3190v2) (ISSN: 2167-9843). PeerJ Inc. https://doi.org/10.7287/peerj.preprints.3190v2

van der Bijl, B., Gijsbertsen, B., van Loon, S., Reurich, Y., de Valk, T., Koch, T., & Dugundji, E. (2022). A Comparison of Approaches for the Time Series Forecasting of Motorway Traffic Flow Rate at Hourly and Daily Aggregation Levels. *Procedia Computer Science*, *201*, 213–222. https://doi.org/10.1016/j.procs.2022.03.030

A System Design Diagram

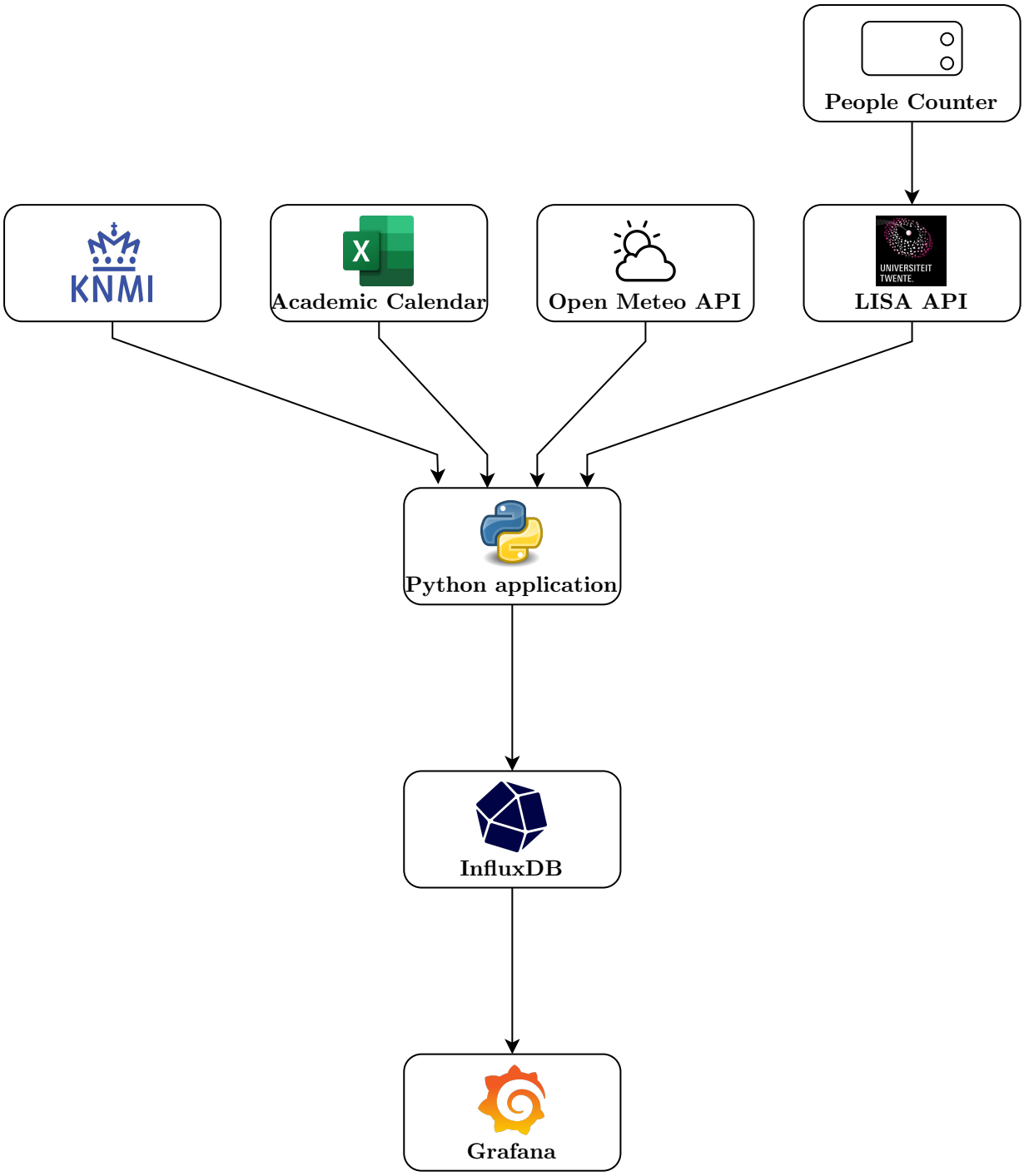
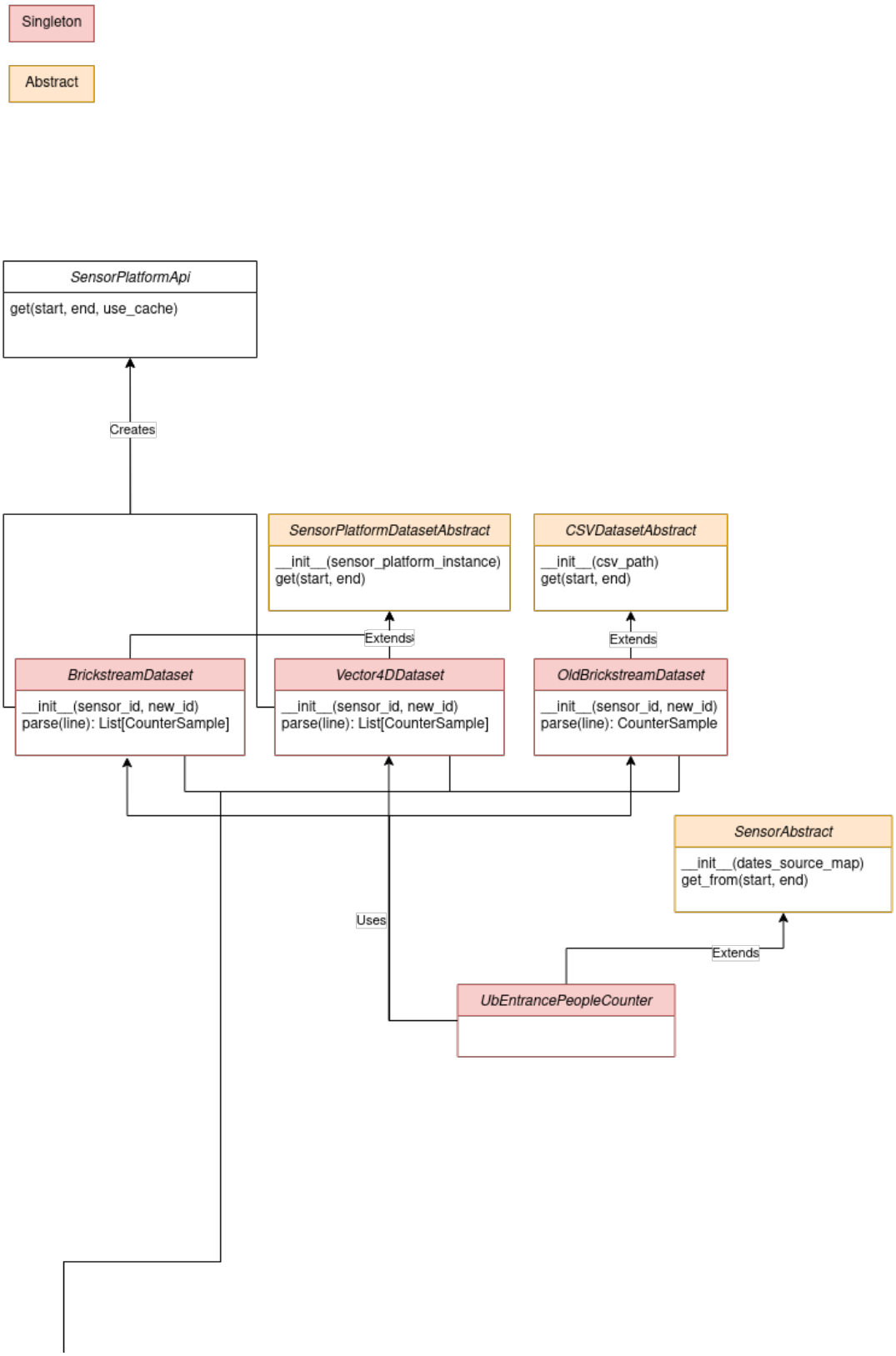


Figure 20: System Design Diagram

B Class Diagrams



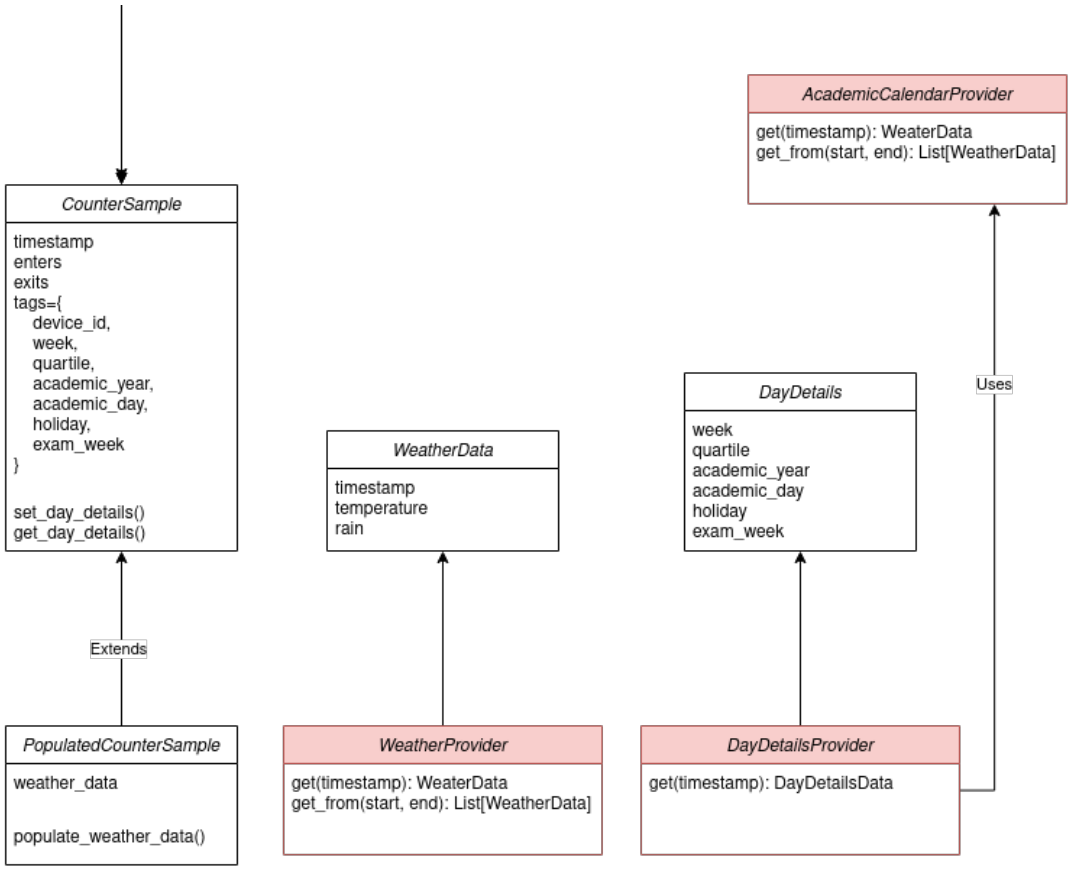


Figure 21: Providers Class Diagram

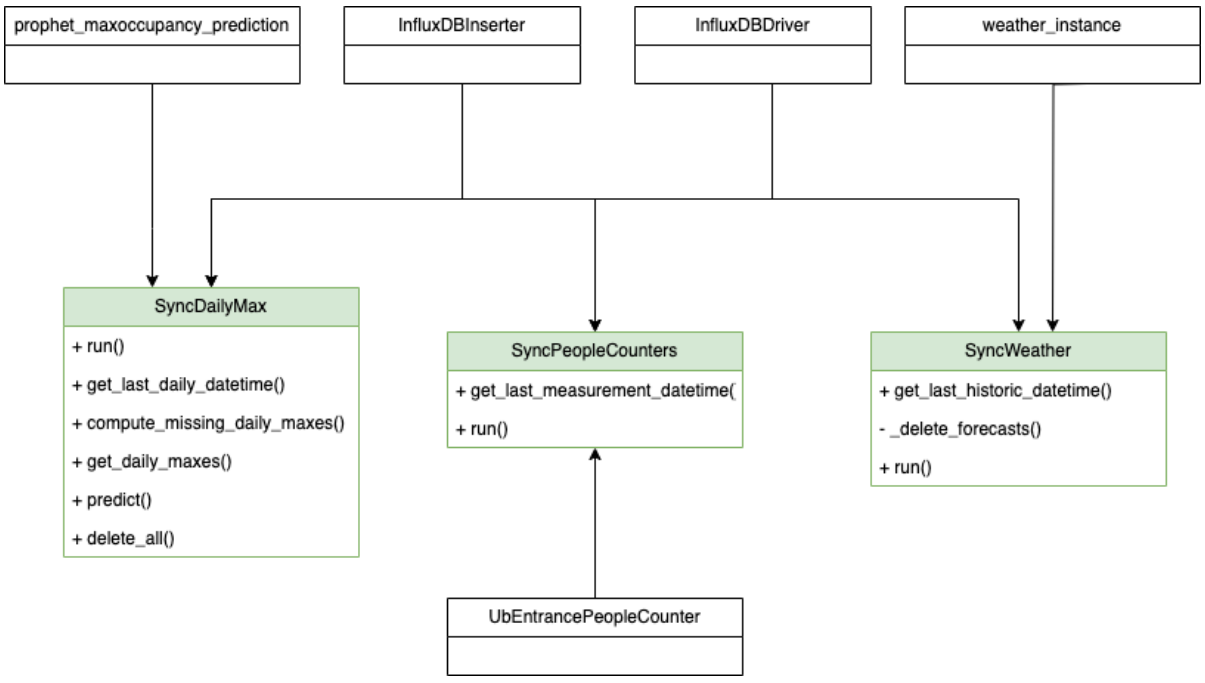


Figure 22: Jobs class diagram

C Sequence Diagrams

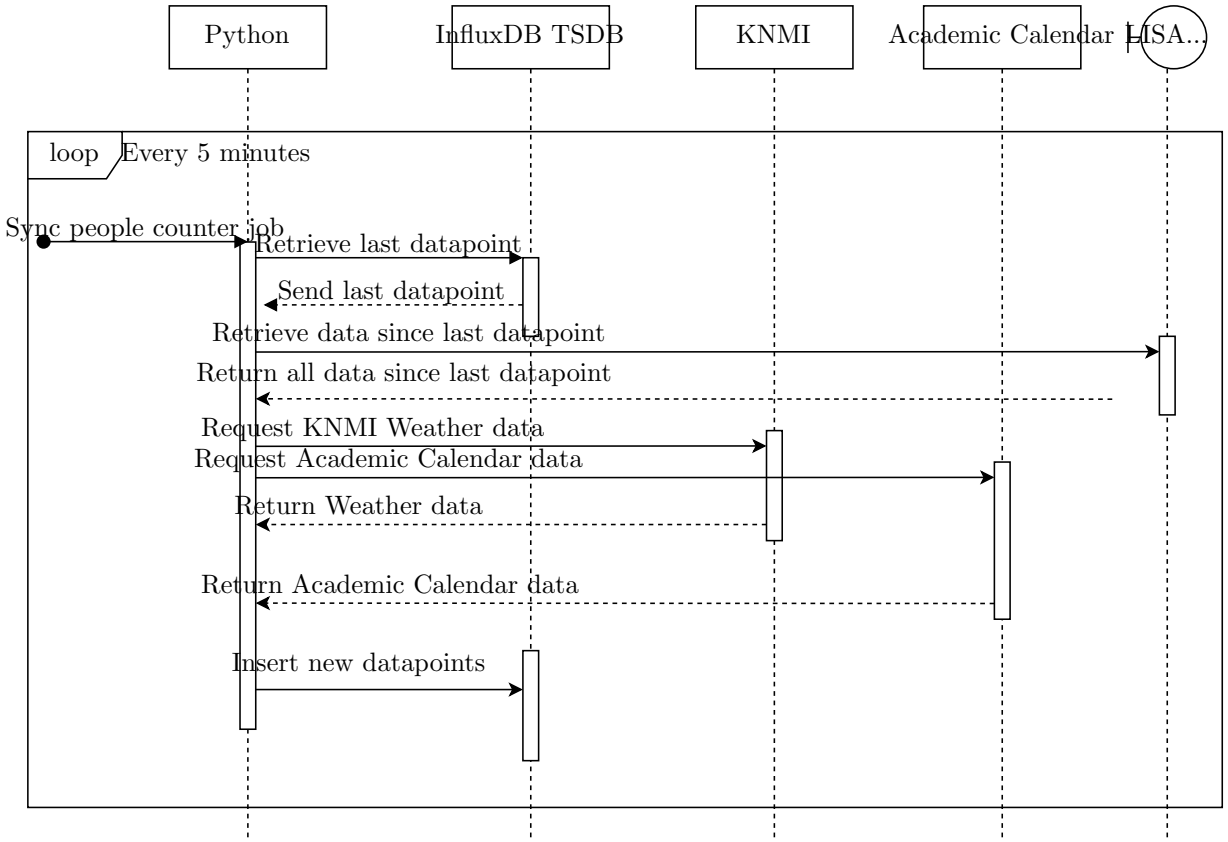


Figure 23: Sequence diagram for inserting new datapoints

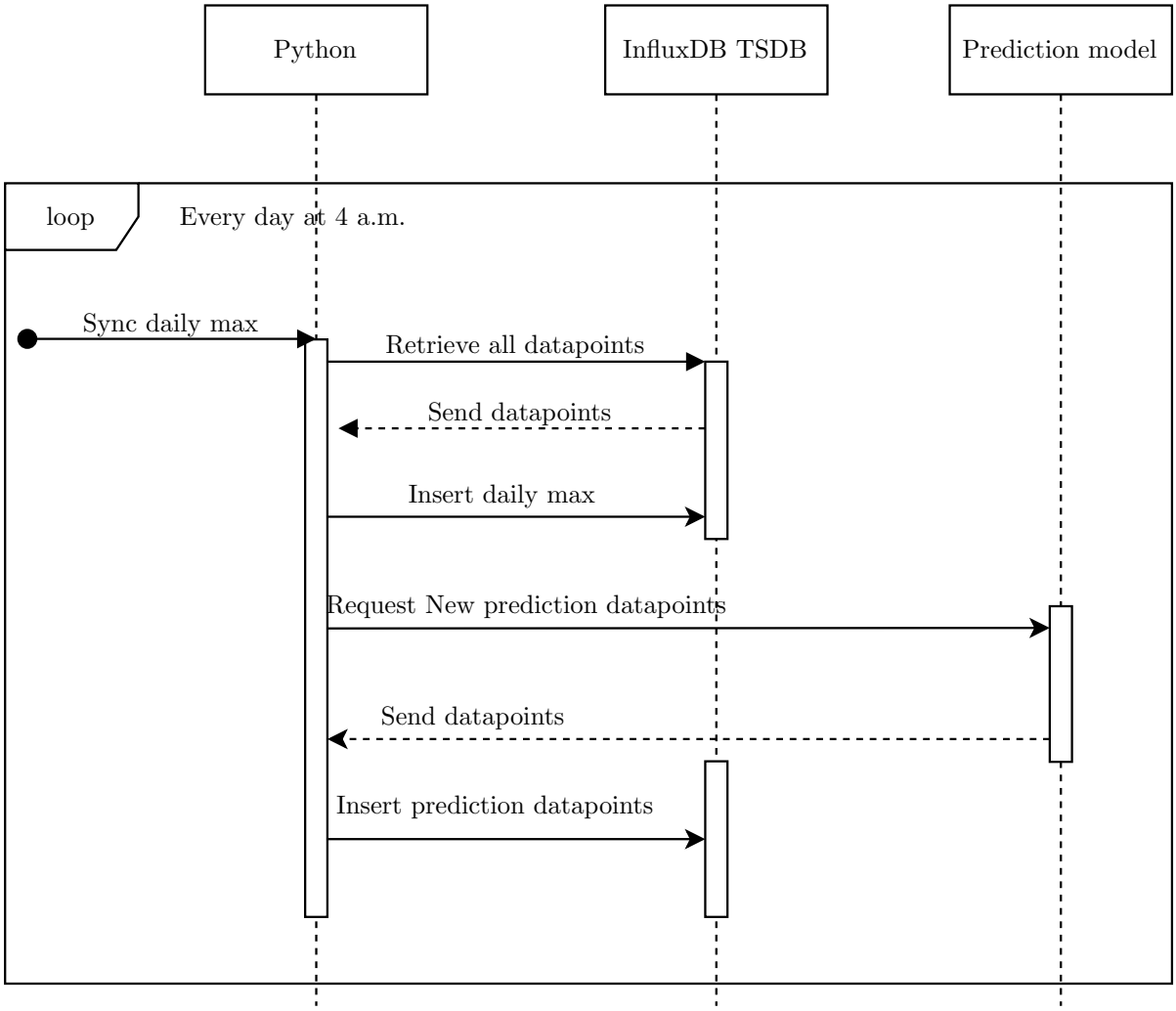


Figure 24: Sequence diagram for inserting daily max values

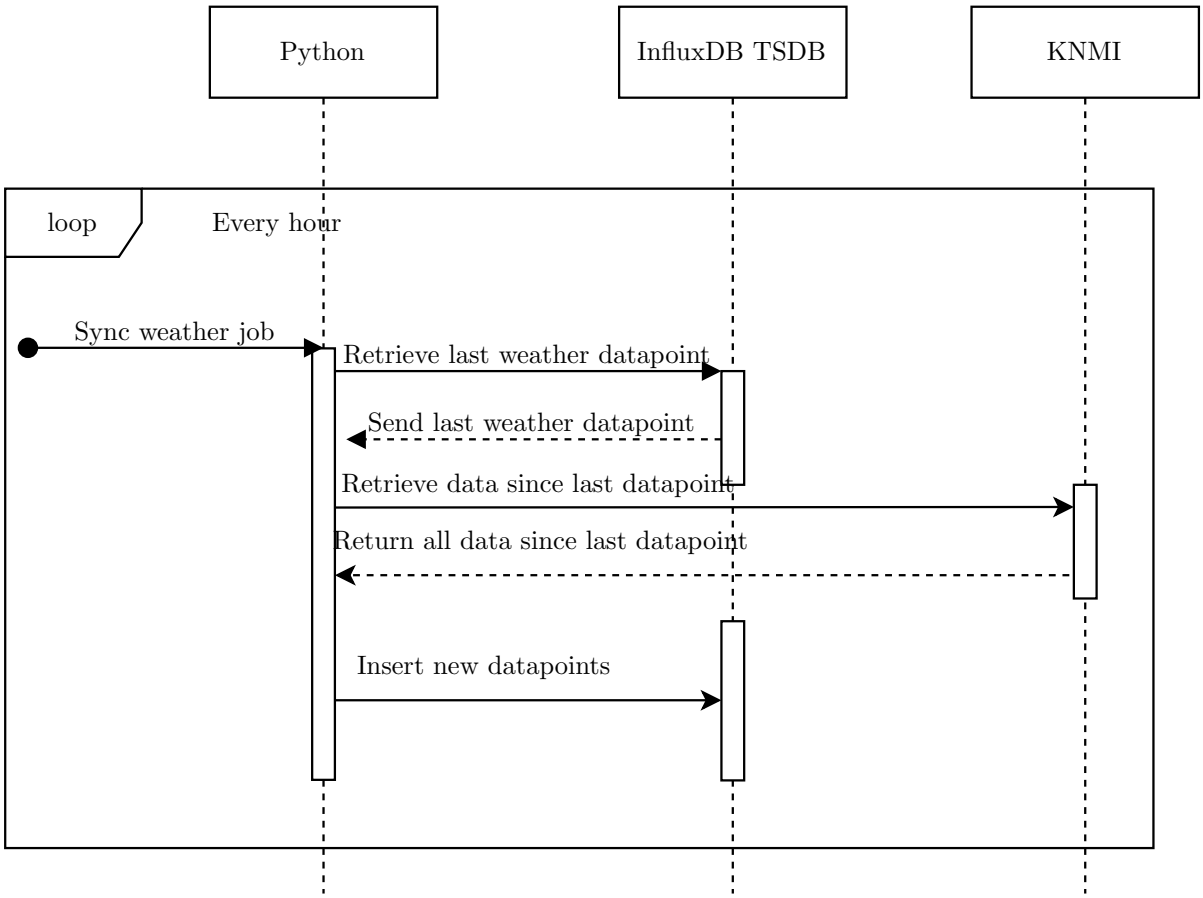


Figure 25: Sequence diagram for inserting weather data

D Prediction Graphs

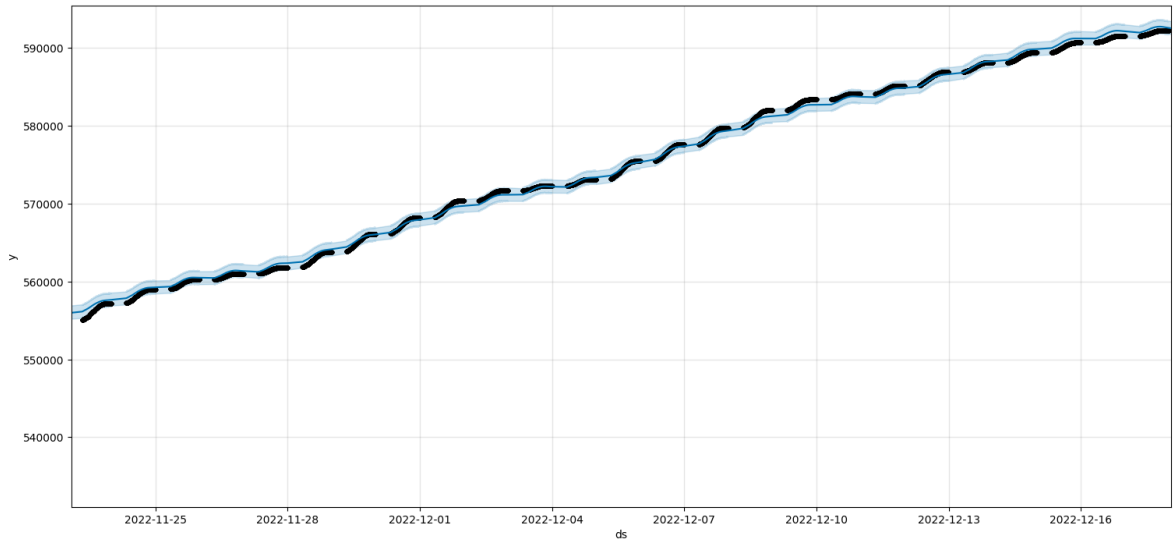


Figure 26: A prediction by Prophet on the cumulative enters, zoomed in on an arbitrary window

E Poster

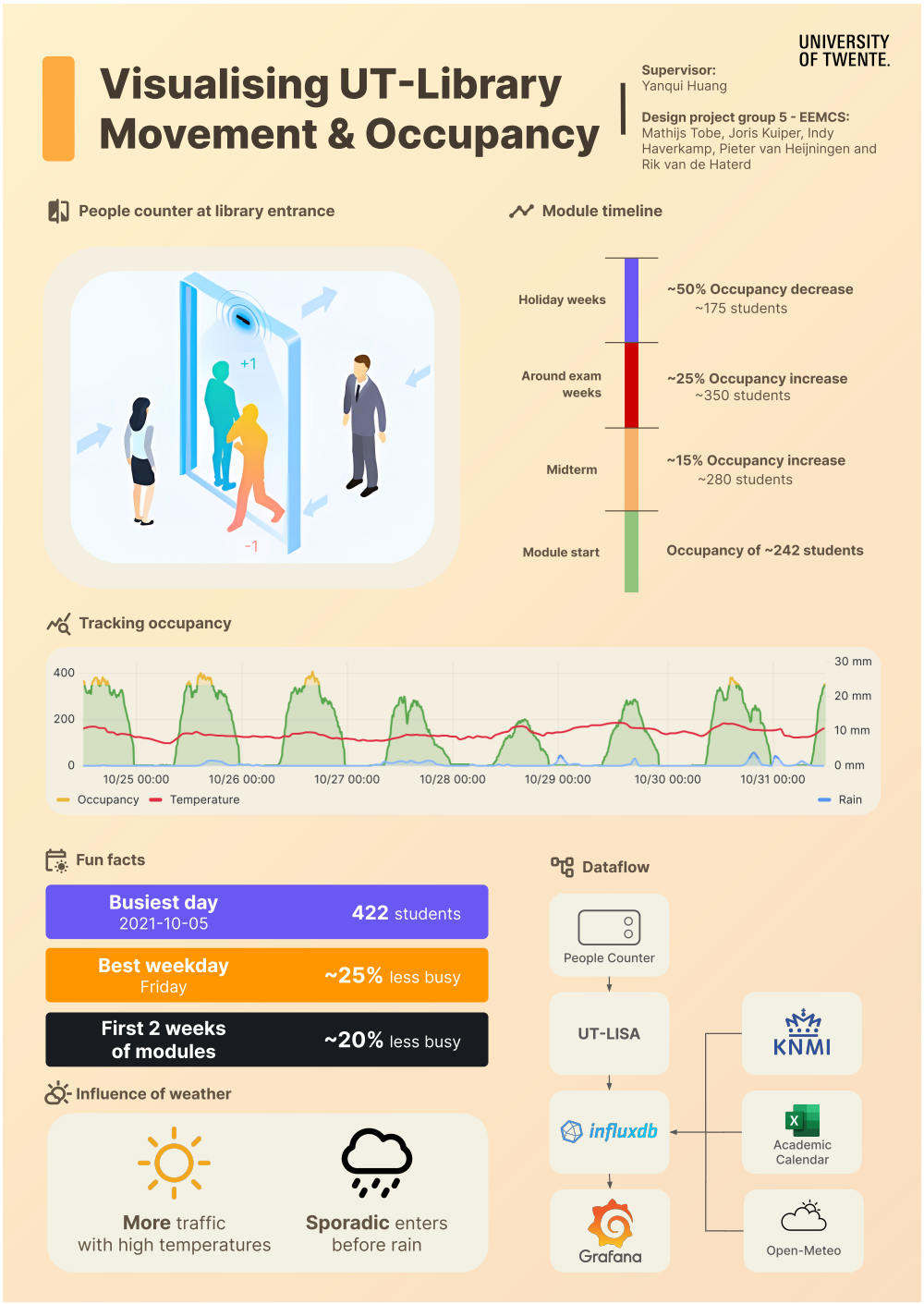


Figure 27: Poster