BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Computer Science

Anomaly detection in data for data cleansing

By: David Zelenay

Student Number: 000000000000

Supervisor: Degree First Name Surname

Vienna, May 14, 2022

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Vienna, May 14, 2022 Signature

Kurzfassung

TODO Add Kurzfassung

Change this

Abstract

TODO add Abstract

Change this

Contents

1 Overview

Check Punctation!

Capitalize Nouns for Titles

1.1 Introduction

With the growing popularity of Internet of Things (IoT) and digitizing business processes there is a growing amount of data available for analysis. In order to utilize the data from the IoT sensors it needs to be preprocessed. One step of preprocessing is data cleaning (also referred as data cleansing). The main goal of data cleansing is to increase the data quality and furthermore to detect and remove anomalies in the data. The quality requirements for the data can differ depending on the use case. Anomalies in sensor data are datapoints which do not picture the reality. For example an anomaly of a temperature sensor would be if the sensor reads 0 °C and the real temperature is 23 °C. This paper specifically focuses on the removal of outliers for water level sensors. The data is provided by FloodAlert [1], which provides a service to warn people about floods, for their area of interest.

1.2 Research Question

What are common methods to detect outliers for time series data?
How can outlier detection methods be compared, with a focus on water level data?
Based on water levels from different rivers, which method is able to classify outliers most reliably?

Add story to research questions or Bullet Points

1.3 Research Method

This thesis will provide an overview and comparison of different approaches to detect outliers. It will focus on time series data, especially water level measurements of rivers. To introduce the topic a general overview about data quality, data cleansing / cleaning and outlier types is provided. For the theoretical pars of the chapters literature research was conducted. After gathering knowledge on different outlier detection approaches they were implemented in Python. To test the performance a suitable performance metric needed to be chosen. To use real world data to classify outliers, the water levels from different measurement stations were taken. In the end the performance of the different approaches to detect outliers are compared.

2 Data Quality

2.1 Features of Data Quality

This section will provide a few example key features of data quality.

Completeness

Data completeness describes the wholeness of data. If there are certain aspects of data missing the data is not complete. For example if each datapoint of a sensor includes the date, time and production speed, the data is not complete, if one of those features is missing or not entire, this datapoint is not complete. [2, 3]

Accuracy

The accuracy of data describes the exactness. Example for possible data which decrease the accuracy are outliers or time shifts. Usually the accuracy of data is harder to measure than the completeness, consistency, structure or documentation. Due to the heterogeneity of sensor data (regarding numerical values like production speed or temperature, not categorical values like on/off) for each datapoint it is difficult to detect which values are genuine and which are sensor errors and therefore outliers. [2]

Man kann hier auch den Rechenfehler von Operationen in Computern anführen. Bei bestimmten Operationen wird darauf im Code Rücksicht genommen.

Consistency

One example for consistency would be, if the data interval is equal. For example there should be a datapoint every ten seconds. As soon as two datapoints are more than ten seconds apart from each other the data is not consistent anymore. [2]

Structure & Documentation

If the structure of the data is not homogeneous, it is very difficult to analyze in an automated way. As a result the data either needs to be structured from the beginning or a process needs to be fabricated to structure the data automatically. Furthermore documentation is required in order to structure and preprocess data. Documentation of data might include data format (Comma Separated Values (CSV), parquet [4], Java Script Object Notation (JSON)), date format (e.g. ISO 8601 with UTC offset), valid value spans (e.g. temperature is only valid if it is between 100 and 400 °C) [2]

2.2 Improving Data Quality

This section will describe methods to improve data quality, based on the features elaborated in section 2.1.

Completeness

The most common methods to increase data completeness are statistical and deep learning based approaches. The goal of these methods are to fill in the missing values of a dataset. An example for a statistical method is DynaMMo [5]. For Artificial Neuronal Networks (ANNs) Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) can be used to predict missing data. [3]

Accuracy

One approach to increase the accuracy of data is to define constraints for each value. E.g. When a machine cannot produce more than ten pieces per second, because it is physically not possible, the value could be limited to less or equal than ten. However limiting the values to a specific range might hide the fact that the machine has an error and is producing faulty products at a rate of 15 pieces per second. This is one of the reasons why more sophisticated outlier detection methods are used. [3]

Consistency

To facilitate consistent data, statistical smoothing or forecasting methods can be used. Examples methods are: AutoRegressive Integrated Moving Average (ARIMA) or Gaussian Process (GP). ANNs can also be used to unify the time series interval between datapoints. [3]

Structure & Documentation

The process of structuring heterogeneous and messy data is called data wrangling. In order to unify the structure of the data at least some documentation is required. Therefore the documentation of the data is fundamental in order to analyse or further process it.

2.3 Data Cleaning & Cleansing Approaches

There are two main methods when it comes to data cleaning or cleansing. Ignoring faulty data or replacing it with a representative value. This paper will use the term data cleaning to describe the process of ignoring or deleting incorrect data and the term data cleansing to portray the process of replacing invalid data with representative values. Faulty, incorrect, invalid or wrong data is data which is inaccurate, incomplete or inconsistent.

"cleaning or cleansing" würde ich hervorheben, weil ich zuerst "data cleaning" und "cleansing" gelesen habe.

Example sensor data: (Valid values for `production_speed` range from 0.00 to 2.00 meter(s) per minute)

`ID`	`timestamp`	`production_speed` (meter/minute)	`machine_running`
0	2021-12-01T12:00:00.000	1.56	True
1	2021-12-01T12:01:00.000	1.58	True
2	2021-12-01T12:02:00.000	3.50	True
3	2021-12-01T12:03:00.000	1.50	False
4	2021-12-01T12:04:00.000	1.50	True
5	2021-12-01T12:05:00.000	1.49	True

Table 1: Example of IoT sensor data

2.4 Data Cleaning

As already mentioned the approach for data cleaning is to ignore or delete faulty data. Depending on the use case either the entire datapoint needs to be ignored or just one value. The process of data cleaning will be shown with the example data pictured in Table 1. The first incorrect datapoint has the `ID` 2. This row is incorrect, because the `production_speed` exceeds the maximum value of 2.00. Depending on the use case (e.g. summary of how long the machine has been running) it can make sense to just ignore the row `production_speed` and keep the value for `machine_running`. The second appearance of a faulty datapoint has the `ID` 2. This datapoint is incorrect since `machine_running` is False but the value of `production_speed` is not 0.00. In this case it does not make sense to keep either of those values for further analysis, because it is impossible to determine which of the two columns are incorrect. A possible result after the data cleaning is shown in Table 2

`ID`	`timestamp`	`production_speed` (meter/minute)	`machine_running`
0	2021-12-01T12:00:00.000	1.56	True
1	2021-12-01T12:01:00.000	1.58	True
2	2021-12-01T12:02:00.000		True
3	2021-12-01T12:03:00.000		
4	2021-12-01T12:04:00.000	1.50	True
5	2021-12-01T12:05:00.000	1.49	True

Table 2: Example of IoT sensor data after cleaning

2.5 Data Cleansing

Data cleansing pursues a different approach. Incorrect data is not ignored, but substituted by a representative value. For example for the datapoint with the `ID` 2 there are several strategies that could be followed. For example the outlier value 3.50 could be replaced with the upper limit of the valid range, in this example 2.00, the value could also be replaced with the last valid value, in this example 1.58, or the value could be replaced with the average of the last n Values, for example with $\frac{1.56+1.58}{2} = 1.57$. For the datapoint with the `ID` 3 there are also different approaches. Either the machine was indeed not running then it would make sense, to set the `production_speed` to 0.0, if short downtimes for this machine are very unlikely then the `machine_running` value could be set to True. A possible result after the data cleansing is shown in Table 3 [6]

Die Methode verändert damit die Messung, d.h. bestimmte statistische Operationen lassen sich dann nicht mehr "sauber" durchführen.

`ID`	`timestamp`	`production_speed` (meter/minute)	`machine_running`
0	2021-12-01T12:00:00.000	1.56	True
1	2021-12-01T12:01:00.000	1.58	True
2	2021-12-01T12:02:00.000	2.00	True
3	2021-12-01T12:03:00.000	1.50	True
4	2021-12-01T12:04:00.000	1.50	True
5	2021-12-01T12:05:00.000	1.49	True

Table 3: Example of IoT sensor data after cleansing

3 Outlier Detection

3.1 Outlier Types

Outliers can be categorized as point outliers or subsequence outliers.

Point Outliers

A point outlier is a single datapoint that strongly varies from the usual trend of the datapoints. [7] Examples of three point outliers are shown in Figure 1. Point outliers are usually easier to detect than subsequence outliers.

Lassen sich diese "outliers" algorithmisch erfassen/detektieren?

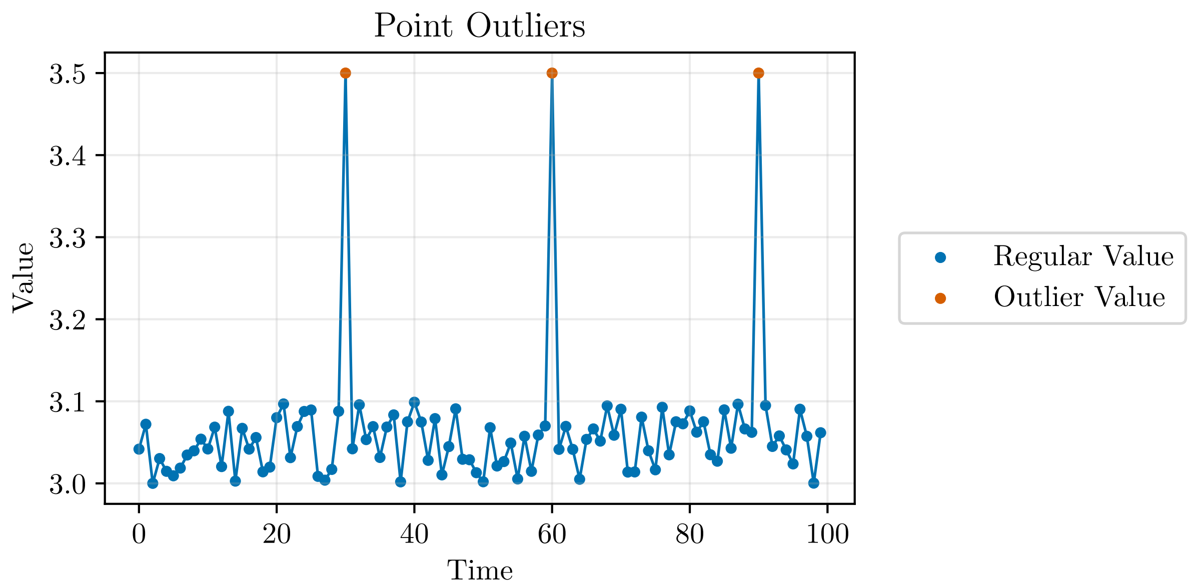

Figure 1: Examples of three point outliers

Subsequence outliers

Subsequence outliers are multiple consecutive datapoints that strongly vary from the usual trend of the datapoints. [7] In Figure 2 examples of subsequence outliers are shown.

Furthermore outliers can be divided into local and global outliers.

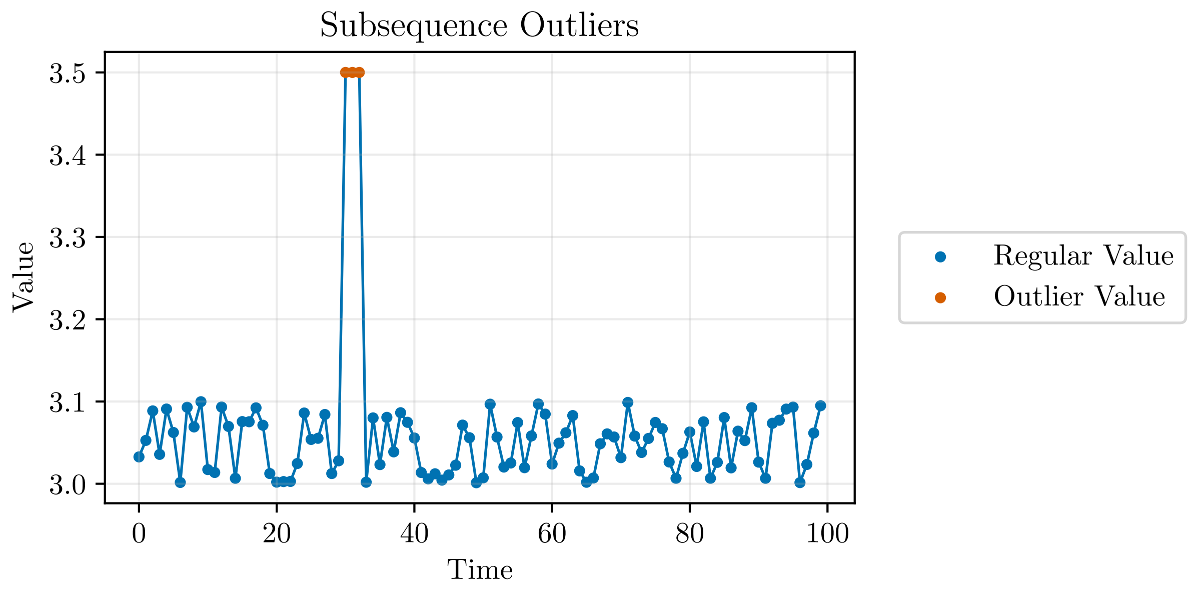

Figure 2: Examples of three subsequence outliers

Local Outliers

A local outlier has a greater variance to its direct neighbouring datapoints (previous and next one) [7] In Figure 3 and Figure 4 examples of local outliers are shown. The first two outliers in Figure 3 have a great variance towards it direct neighbours, however not to the values from Time 60 onwards.

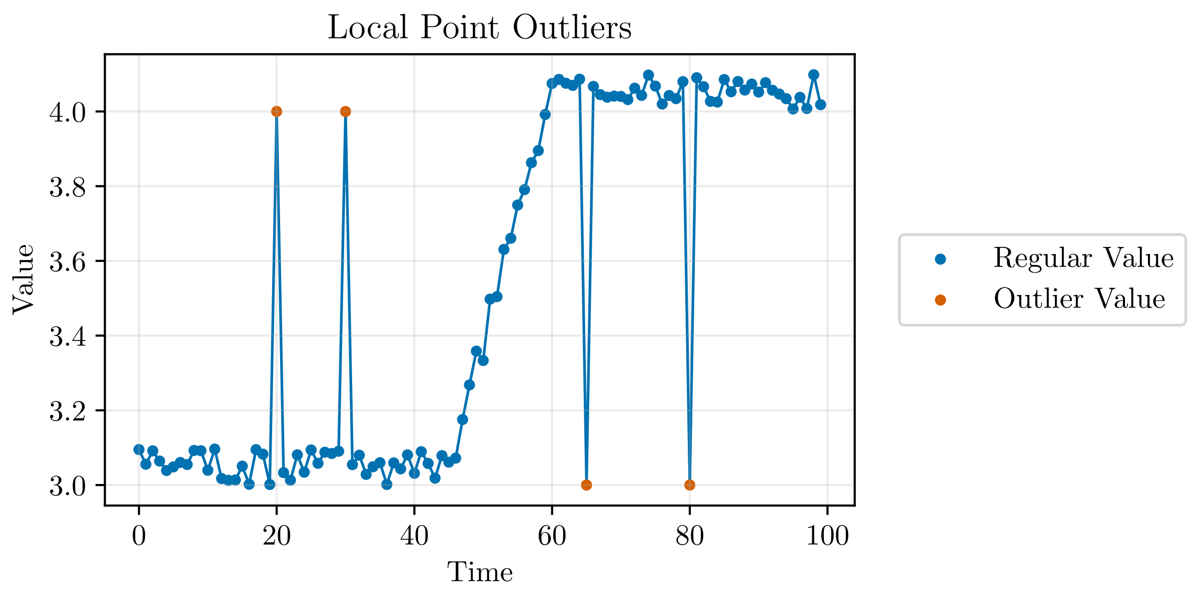

Figure 3: Examples of four local point outliers

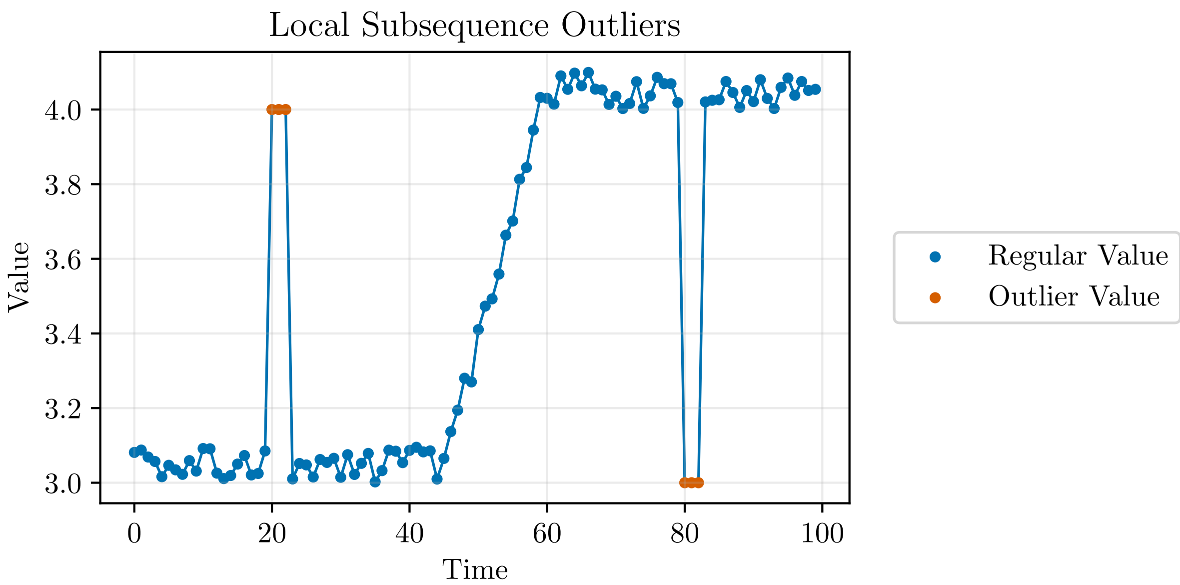

Figure 4: Examples of six local subsequence outliers

Global Outliers

Whereas a global outlier varies more in regard to all datapoints. Figure 1, 2 and 5 show picture examples of global outliers. [7]

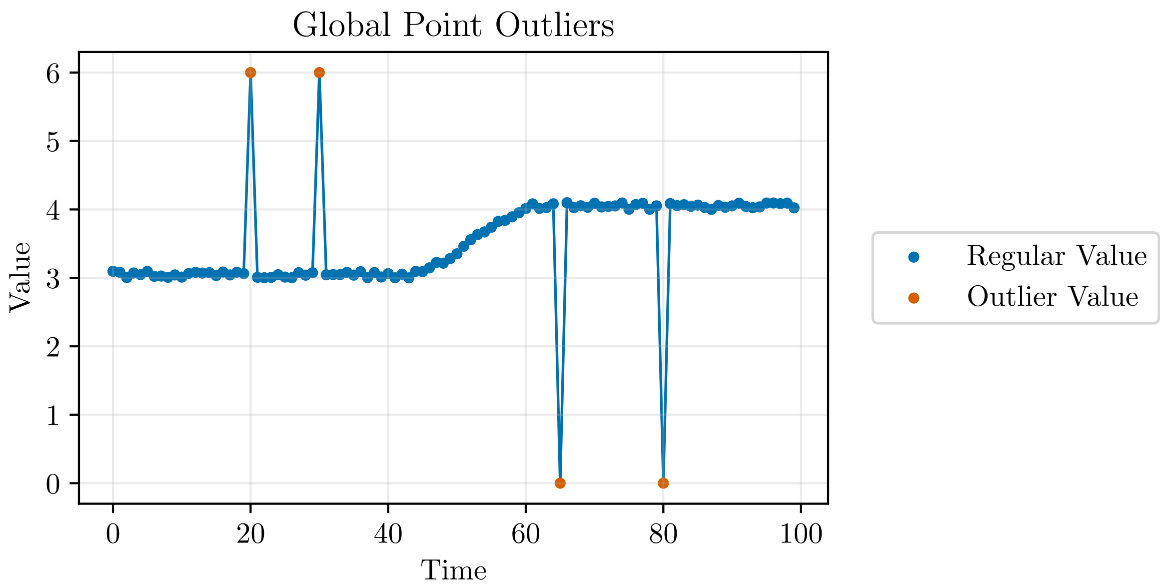

Figure 5: Examples of four global point outliers

3.2 Outlier Detection Approaches

Outlier detection methods can be divided into the following groups

Statistical

For statistical outlier detection, historical data is taken to develop a model that pictures the expected behavior of the data. An example of a statistical outlier detection is the threshold based method described in section 3.3 [8, 9]

Distance based

For this approach a distance metric needs to be defined, (e.g. Euclidean distance). Then each datapoint is compared to the data preceding it. The greater the distance between the current and previous datapoints the greater the probability of an anomaly. [8–10]

Clustering

Clustering also requires a set of historical data in order to train the clustering model. Usually the data is clustered into two clusters: normal data and anomalous data. Depending on the distance of a new datapoint to the "normal" and the "anomalous" cluster it is classified. [8–10]

Predictive

In this approach a prediction model needs to be developed, based on previous data. The prediction of this model is then compared with the actual datapoint (new data, which was not used in training the model). If the actual datapoint differs too much from the prediction it is labelled as an anomaly. [8, 9]

Ensemble

as the word ensemble suggests, this is a collection of outlier detection methods that use a specific vote mechanism to determine whether a datapoint is faulty or normal. For example using the majority vote system and a statistical, distance based and predictive method to detect outliers. If at least two methods flag a datapoint as an outlier the ensemble reports it as an outlier as well. If only one method reports it as an outlier the ensemble does not flag it as an anomaly. [8]

3.3 Threshold based Outlier Detection

Threshold based detection methods are able to identify outliers based on a given threshold τ. These Methods can be described with the following formula

$$|x_t - \hat{x}_t| > \tau \text{ [7]} \tag{1}$$

Where x_t is the actual value and $\hat{x}_t$ is the expected value and τ is a given threshold. Methods to calculate $\hat{x}_t$ will be described in the following sections. Furthermore $\hat{x}_t$ can be calculated using the entire data series or with subsets (of equal length) of the entire data series. This means $\hat{x}_t$ can be either calculated for the whole data series or for just a segment. Depending on the sensitivity wanted for outlier detection an appropriate τ needs to be chosen. The greater τ is the fewer outliers will be detected. The smaller τ is the more outliers will be identified. [7]

Mean

$$\text{mean} = \bar{x} = \frac{1}{n}\sum_{t=0}^{n} x_t \tag{2}$$

Where n is the total number of samples. Using the mean as an expected value is not robust to outliers, because the median is not as robust as the mean in hindsight to outliers. To calculate the mean all datapoints of a series must be summed up and then divided by the number of datapoints.

Median

If n is odd:

$$median(x) = x_{(n+1)/2} \tag{3}$$

If n is even:

$$median(x) = \frac{x_{n/2} + x_{(n+1)/2}}{2} \tag{4}$$

Where x is a dataset of n elements ordered from smallest to largest ($x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_{n-2} \leq x_{n-1} \leq x_n$) [7] To calculate the median all values must be sorted from smallest to largest. If the number of datapoints is odd then the most center datapoint is the Median (e.g. if the series consist of 7 values the third value is the median). If the number of datapoints is even then the median is the mean of the two datapoints in the center.

Median Absolute Deviation (MAD)

The Median Absolute Deviation

$$MAD = median(|x_t - median(x)|) \tag{5}$$

MAD is a more robust (regarding outliers) way to calculate the deviation of a dataset. To calculate the MAD firstly the median of the dataset must be calculated. Then the absolute difference between x_t and the median of the dataset is calculated. The Median of all differences results in the MAD [11, 12]

3.4 Outlier Detection using z-score

The z-score, also known as the standard score, is the factor of how many standard deviations a datapoint differs from the mean. Using standard score to detect outliers works best, when the data is distributed normally. Because then it can be assumed, that e.g. the top and bottom 0.5% are outliers and therefore every value with $|z| >\approx 2.576$ can be classified as an outlier. A visual representation of the z-score in a normal distribution is shown in Figure 6

$$z_t = \frac{x_t - \mu}{\sigma} \tag{6}$$

Where μ is the mean of the dataset and σ is the standard deviation. When the mean and the standard deviation of the entire dataset is now known the mean and the standard deviation of a known sample can be used. [13–15]

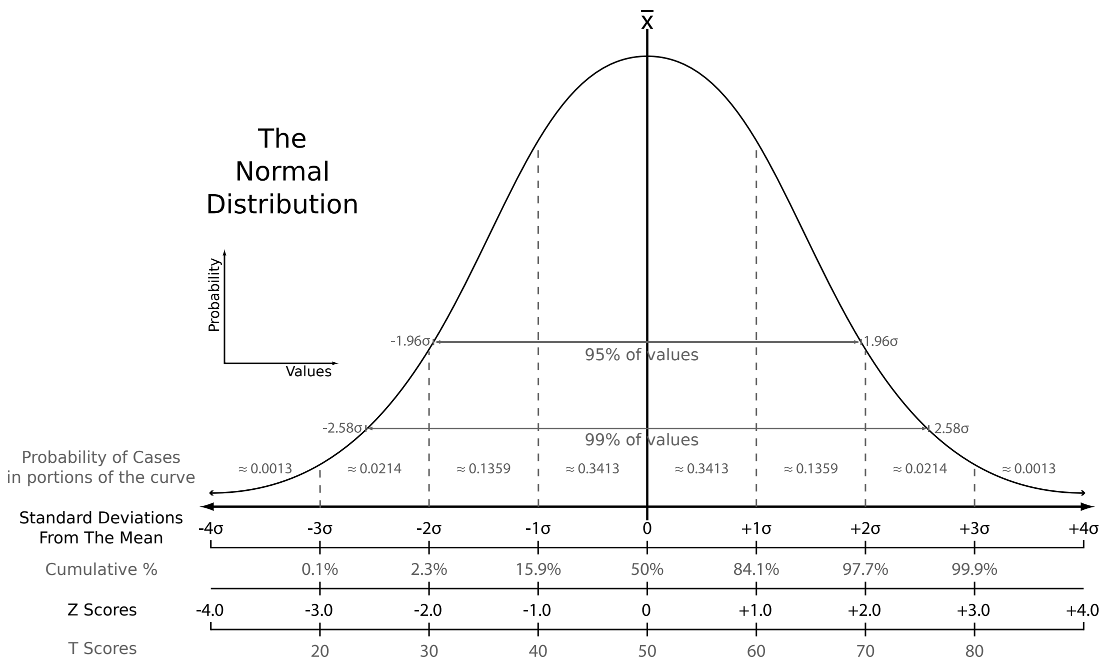

Figure 6: Standard Score in a normal distribution [16]

$$\sigma = s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{7}$$

[14, 15]

To check whether the value x_t is an outlier the absolute of its z-score is compared against a threshold (τ) and if the absolute value of the z-score exceeds the threshold x_t is classified as an outlier.

$$|z_t| > \tau \tag{8}$$

3.5 Outlier Detection using modified z-score

Because the mean and standard deviation are not robust towards outliers the z-score can be modified to use more robust metrics for the expected value and the variation of the values. To make the outlier detection with the z-score more robust the mean can be replaced with the median and the standard deviation with the MAD or the Normalized Median Absolute Deviation (MADN). A possible formula for a modified z-score, as described in [17], could be:

$$m_t = \frac{|x_t - median(X)|}{MADN(X)} \tag{9}$$

Where the formula for the MADN is:

$$MADN(X) = \frac{MAD(X)}{0.6745} \tag{10}$$

The constant value 0.6745 is the 75$^{\text{th}}$ percentile of a standard normal distribution, which is equal to the MAD of a standard normal distribution with $\sigma = 1$. [17]

The classification of outliers using the modified z-score (m_t) is the same for the regular standard score described in section 3.4:

$$|m_t| > \tau \tag{11}$$

4 Outlier Detection based on Water Level Data

FloodAlert (or “Pegelalarm” in German) is a service that provides water levels from about 30,000 measurement stations. It provides notifications to registered individuals, when a certain water level threshold is reached. Thus warning people about possible floods in their area. [1]

Unfortunately sometimes an incorrect water level is reported by the sensors, which are not maintained by the team of FloodAlert. FloodAlert just fetches the data from different maintainers of sensors. To reduce the frequency of false alarms, outliers should be identified and flagged. The theoretical parts of the previous sections are applied and tested on real-word data.

4.1 What is the Goal?

The ideal goal of FloodAlert would be to have a general model, that works on a variety of rivers. The model should be able to predict the probability of being an outlier for each water level value, without knowing future values. Thereby making it possible to classify incoming datapoints in real time, without needing future datapoints to make a prediction. Furthermore it would be ideal if the outlier detection method is able to adjust its parameters automatically for each river / water level measurement location. Because different rivers have different fluctuations in water level. Therefore each water level measurement station needs individual parameters. For example one river regularly has an increase and decrease of 10 centimeters whereas for another river 10 cm of water level variance from one datapoint to the next is definitely an outlier.

Another approach would be to also take future values into consideration when predicting the probability of a value being an outlier. This method is probably easier and likely leads to better results. However a method which does not take future values into consideration would be more useful for FloodAlert, since the values could be classified immediately.

4.2 How to retrieve the Data (Description of the API)

To make the access to the API easier SOBOS GmbH developed a Python wrapper which returns the requested data as a Pandas dataframe. [18] The code for the Python wrapper is available on GitHub [19]: https://github.com/SOBOS-GmbH/pegelalarm_public_pas_doc [20]

In order to request data, an API key needs to be requested using credentials. To request the API key a POST request needs to be sent to this endpoint https://api.pegelalarm.at/api/login, where the request body contains the users’ credentials as shown in Listing 1. To generate

Listing 1 Request body to get API key

```json
{
    "username": "myUsername",
    "password": "myPassword"
}
```

the actual API key the key from the response needs to be hashed using the Keyed-Hashing for Message Authentication (HMAC) Algorithm. This is done using Python's built in "hmac" module. [21]. Afterwards the HMAC is byte64 encoded, so it can be sent in the "X-AUTH-TOKEN" header field.

To get the unique identifier for a specific measurement station the list endpoint can be used. Which accepts three optional query parameters (qStationName - station name, qWater - water name and commonid - the unique identifier of a station) and returns a list of matching stations with metadata like coordinates, country or last water level. E.g. to retrieve the identifier of the Danube station located in Linz the URL is the following: https://api.pegelalarm.at/api/station/1.1/list?qStationName=Linz&qWater=Donau. Keep in mind in order to retrieve any data the header value "X-AUTH-TOKEN" must be set. An example response of this request is shown in Listing 2. Where the unique identifier is the "commonid" in line 9.

To retrieve historical data, the history endpoint can be used. The request URL has the following structure: https://api.pegelalarm.at/api/station/1.1/<unit>/<commonid>/history?<parameters>. The unit can either be "height" or "flow". For this thesis only height data was used. The following parameters can be set:

- **loadStartDate**: The start timestamp of the queried data.
- **loadEndDate**: The end timestamp of the queried data.
- **granularity**: The granularity of the response.

Possible values for the granularity are: "raw" (for the last 3 months of data), "hour", "day", "month", "year" or "era" (era returns one value for a given time) When requesting aggregated data (anything other than "raw"), the maximum value of the timespan is used. The timestamps are in the following format: "<DD>.<MM>.<YYYY>T<HH>:<MM>:<SS><+-UTCOFFSET>", where the '+' is URL encoded to "%2B". E.g.: "31.03.2022T13:35:40%2B0200". If no parameters are provided the API returns the last few datapoints.

An example request would be: https://api.pegelalarm.at/api/station/1.1/height/207068-at/history?loadStartDate=01.03.2022T13:35:40%2B0200&loadEndDate=01.03.2022T18:00:00%2B0200&granularity=hour The result of this request is shown in Listing 3.

Listing 2 Example response of the list endpoint


```json
{
    "status": {
        "code": 200
    },
    "payload": {
        "stations": [
            {
                "name": "Donau / Linz / at",
                "commonid": "207068-at",
                "country": "Österreich",
                "stationName": "Linz",
                "water": "Donau",
                "region": "Oberösterreich",
                "latitude": 48.306915712282,
                "longitude": 14.284689597541,
                "positionKm": 2135.17,
                "altitudeM": 247.74,
                "defaultWarnValueCm": 550.0,
                "defaultAlarmValueCm": 630.0,
                "data": [
                    {
                        "type": "height in cm",
                        "value": 358.0,
                        "requestDate": "19.04.2022T14:59:51+0200",
                        "sourceDate": "19.04.2022T14:45:00+0200"
                    }
                ],
                "trend": 10,
                "situation": 10,
                "visibility": "PUBLIC",
                "stationType": "surfacewater"
            }
        ]
    }
}
```

Listing 3 Example response of historical water level data for one station


```json
{
    "status": {
        "code": 200
    },
    "payload": {
        "history": [
            {
                "value": 360.0,
                "sourceDate": "01.03.2022T13:00:00+0100"
            },
            {
                "value": 360.0,
                "sourceDate": "01.03.2022T14:00:00+0100"
            },
            {
                "value": 359.0,
                "sourceDate": "01.03.2022T15:00:00+0100"
            },
            {
                "value": 361.0,
                "sourceDate": "01.03.2022T16:00:00+0100"
            },
            {
                "value": 362.0,
                "sourceDate": "01.03.2022T17:00:00+0100"
            }
        ]
    }
}
```

4.3 Overview of the Data

To test and tune the outlier detection methods the entire history of data was used. The granularity of the data usually is hourly up to the last three months of data. There are a few exceptions, where the granularity is not hourly, because either one or a few datapoints are missing. For the last three months the granularity is “raw”, which is different for each station. Additionally the base (where the sensor reports water level equals zero) also differs per station. Some station have an arbitrary zero level whereas some stations have the sea level as a base. Thus the water level does not actually reflect the actual height, or depth, of the water. Instead it represents the relative water level to a specific height.

The following water level measurement stations were used to test the outlier detection methods.

Check and maybe remove automatic hyphenation

Station Aghacashlaun, Aghacashlaun (36022-ie)

This station, on the river Aghacashlaun and in the area of Aghacashlaun, is located in northern Ireland. The coordinates of this station are: 54.03647, -7.94672 (Latitude (Lat), Longitude (Long)). [22]

Station Murg, Frauenfeld (2386-ch)

The station, on the river Murg and in the area of Frauenfeld, is located in northern Switzerland. The coordinates of this station are: 47.56852, 8.89432 (Lat, Long). [23]

Station Sieg, Betzdorf (2720050000-de)

The station, on the river Sieg and in the area of Betzdorf, is located in western Germany. The coordinates of this station are: 50.79332, 7.86390 (Lat, Long). [24]

Station Losse, Helsa (42960105-de)

The station, on the river Losse and in the area of Helsa, is located in central Germany . The coordinates of this station are: 51.25574, 9.68537 (Lat, Long). [25]

Station Crana, Tullyarvan (39003-ie)

The station, on the river Crana and in the area of Tullyarvan, is located in northern Ireland. The coordinates of this station are: 55.14356, -7.45239 (Lat, Long). [26]

4.4 Manually detect outliers for a subset of data

In order to speed up the manual labeling of outliers a program was written. The program is a Plotly Dash [27] web application which displays the water level data as a scatter chart. By clicking the datapoints in the chart the user is able to toggle the datapoint as an outlier or back to a regular value. In Listing 4 the source code of the Dash application is shown. In Figure 7 the Website of the Python app is shown, the red dots between 25 and October 28 are already classified outliers. Below the chart a rangeslider is located, to move the zoomed in view horizontally. The refresh button on the left side refreshes the graph, thus updating the color and label of previously selected outliers. Furthermore it saves the data to a parquet file. The upper and lower limit of the y-axis also gets updated, when pressing the refresh button. The limits are automatically set to the lowest and highest regular value. This was implemented, because some datasets had outliers with a huge difference towards the regular data. Without the automatic scaling of the y-axis, detecting other outliers with a smaller difference was not possible.

Which reference to add? Website, Documentation or GitHub?

Figure 7: Dash Webapp to classify outliers

Manual Outlier Detection Webapp using Dash

```python
import json

import numpy as np
import pandas as pd
import plotly.express as px
from dash import Dash, dcc, html
```

```python
from dash.dependencies import Input, Output

app = Dash(__name__)

stations_dict = pd.read_csv('./data/stations.csv').groupby(
    ['common_id']).first().to_dict('index')

common_id = '2736731000100-de'
df = pd.read_parquet(f'./data/{common_id}_outliers_classified.parquet')
df.info()
fig = px.scatter(df, x='timestamp', y='water_level',
                 title=f'{common_id}', color='is_outlier')
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     step="day",
                     stepmode="backward"),
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)
app.layout = html.Div([
    html.H1('Manual Outlier Selection'),
    dcc.Graph(
        id='water-level-graph',
        figure=fig
    ),
    html.Button('Refresh', id='refresh-btn', n_clicks=0),
    html.Div([
        dcc.Markdown('Clicked value:'),
        html.Pre(id='click-data'),
    ]),
])

```

```
def toggle_outlier(timestamp: str):
    df.loc[(df['timestamp'] == timestamp), 'is_outlier'] = np.invert(
        df.loc[(df['timestamp'] == timestamp), 'is_outlier'])


@app.callback(
    Output('click-data', 'children'),
    Input('water-level-graph', 'clickData'))
def display_click_data(clickData):
    if clickData is not None:
        toggle_outlier(clickData['points'][0]['x'])
    return json.dumps(clickData, indent=2)


@app.callback(
    Output('water-level-graph', 'figure'),
    Input('refresh-btn', 'n_clicks'))
def update_output(n_clicks):
    fig = px.scatter(df, x='timestamp', y='water_level',
                     title=f'{common_id}', color='is_outlier')
    df.to_parquet(f'./data/{common_id}_outliers_classified.parquet')
    fig.update_layout(
        xaxis=dict(
            rangeselector=dict(
            ),
            rangeslider=dict(
                visible=True
            ),
            type="date"
        )
    )
    fig.update_layout(
        yaxis_range=[df.loc[df['is_outlier'] == False,
                            'water_level'].min() - 5,
                     df.loc[df['is_outlier'] == False,
                            'water_level'].max() + 5])
    return fig


```

	water_level	water_level_diff	timedelta
count	27189.000000	27188.000000	27188.000000
mean	36.009754	-0.000077	0.809990
std	14.716286	7.220175	1.256409
min	0.000000	-107.500000	0.000000
25%	26.400000	-0.300000	1.000000
50%	31.700000	-0.100000	1.000000
75%	40.300000	0.000000	1.000000
max	190.000000	94.800000	178.000000

Table 4: Seven number summary of Aghacashlaun - Aghacashlaun (all values)

```
if __name__ == '__main__':
    app.run_server(debug=True)
```

Listing 4: Manual Outlier Detection Webapp using Dash

add comments to code

4.5 Explorative Data Analysis

add another station

add line chart over whole time with 1-2 zooms

4.5.1 Aghacashlaun Station

change formatting of tables

Table 4 shows seven number summary for all datapoints (outliers and regular values). The column “water_level_diff” is the difference in water level between two consecutive datapoints. The column “timedelta” is the time difference between two consecutive datapoints in hours. Table 5 and Table 6 picture seven number summaries for regular and only outlier values. The standard deviation is clearly higher for outlier values when compared to the standard deviation of regular values.

In figure Figure 8 the class distribution is shown. The classes are not balanced, there are far more regular values than outliers. This needs to be kept in mind when choosing a performance metric.

Figure 9 shows boxplots for all values, regular values and outlier values. The boxplots show, that the upper and lower whisker boundaries cannot be used to classify outliers reliably. Since there are many regular values that exceed the upper limit of the whisker, which is 1.5 times the InterQuartile Range (IQR).

In Figure 10 histograms of the different water levels are shown. The y-axis is scaled logarithmically, so less common values are also visible. The histogram shows, that values between 20 and 40 are most common and the higher the water level is the less common it becomes.

	water_level	water_level_diff	timedelta
count	26544.000000	26543.000000	26543.000000
mean	35.381205	-0.411276	0.808311
std	13.763332	5.165770	1.268532
min	20.000000	-107.500000	0.000000
25%	26.400000	-0.300000	1.000000
50%	31.400000	-0.100000	1.000000
75%	39.800000	0.000000	1.000000
max	151.600000	51.900000	178.000000

Table 5: Seven number summary of Aghacashlaun - Aghacashlaun (regular values)

	water_level	water_level_diff	timedelta
count	645.000000	645.000000	645.000000
mean	61.876744	16.921550	0.879070
std	25.476859	28.411003	0.560844
min	0.000000	-90.000000	0.000000
25%	40.000000	-10.000000	1.000000
50%	60.000000	12.500000	1.000000
75%	80.000000	39.400000	1.000000
max	190.000000	94.800000	9.000000

Table 6: Seven number summary of Aghacashlaun - Aghacashlaun (outlier values)

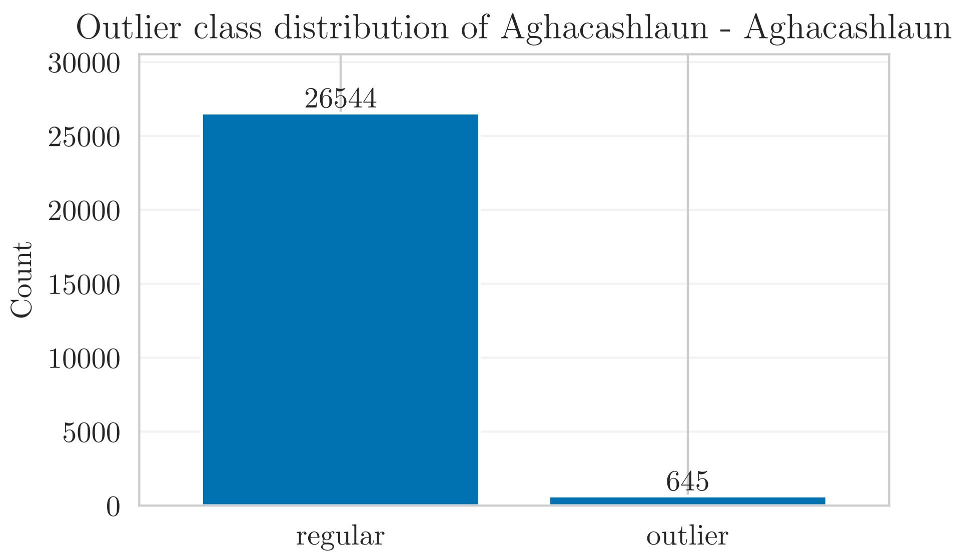

Figure 8: Class distribution of Aghacashlaun - Aghacashlaun

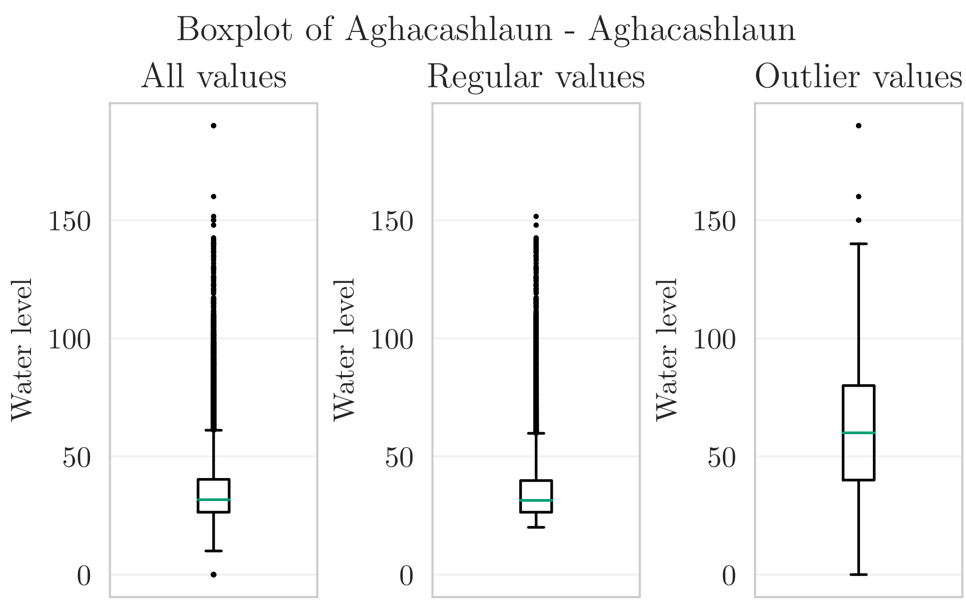

Figure 9: Boxplot of Aghacashlaun - Aghacashlaun

Figure 11 is quite similar, but instead of the water level the difference in water level between two consecutive datapoints is used.
Figure 12 Shows the histogram of the time difference, in hours, between each datapoint. The largest gap of data is 178 hours, which is about one week.
In Figure 13 the water level over the time is shown. This chart shows, that the water level usually oscillates between 20 and 100. Figure 14 displays an example of subsequence outliers. Within one hour the water level rises by double the previous value. This is very unlikely, thus these values were classified as outliers. The reason, for multiple consecutive outliers, with decreasing water level over time could be, that the station does not report the actual water level but a moving average over the last few recorded values. This would at least explain the pattern of outliers shown in Figure 14.
Code: [28]

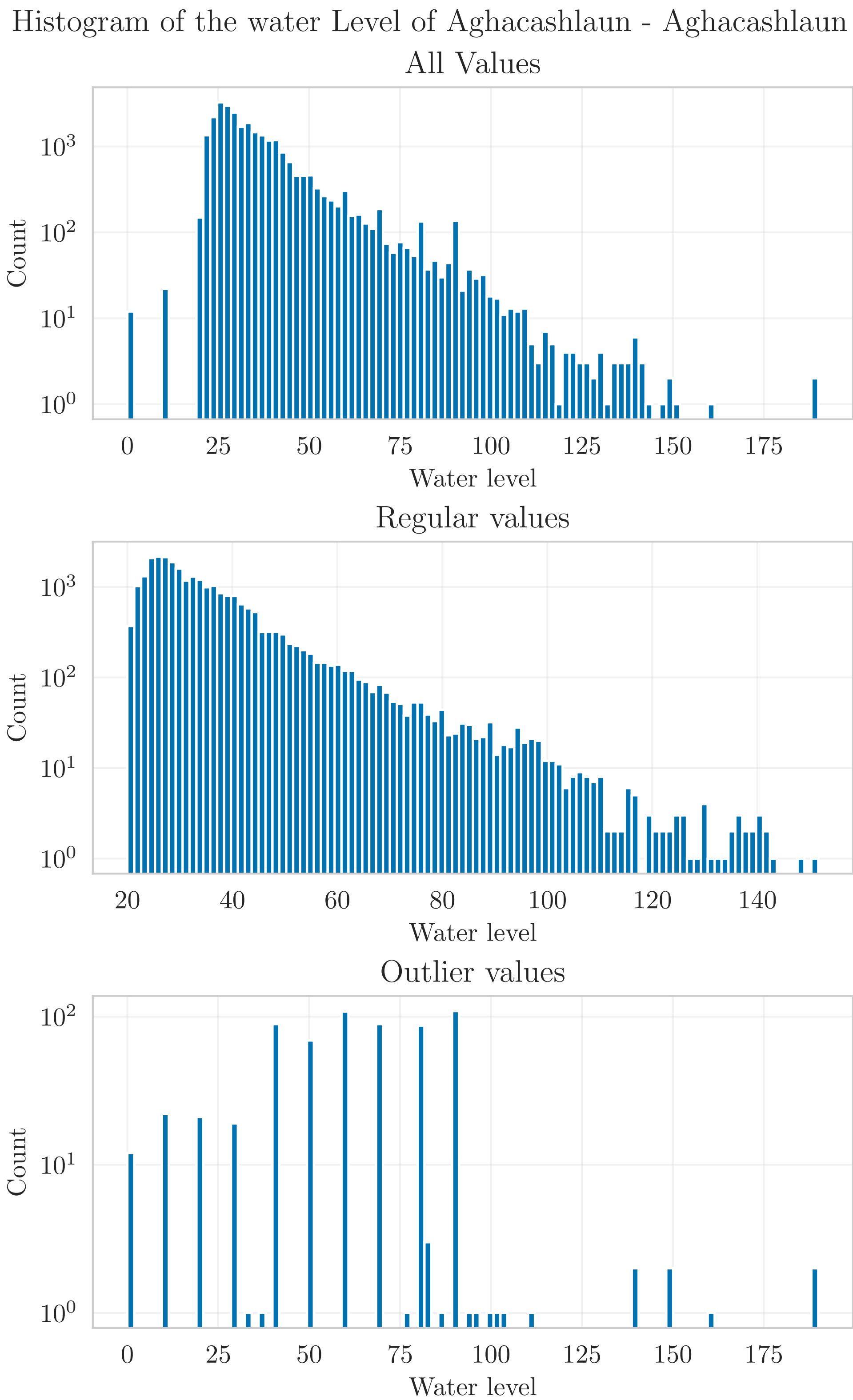

Figure 10: Histogram of the water level of Aghacashlaun - Aghacashlaun

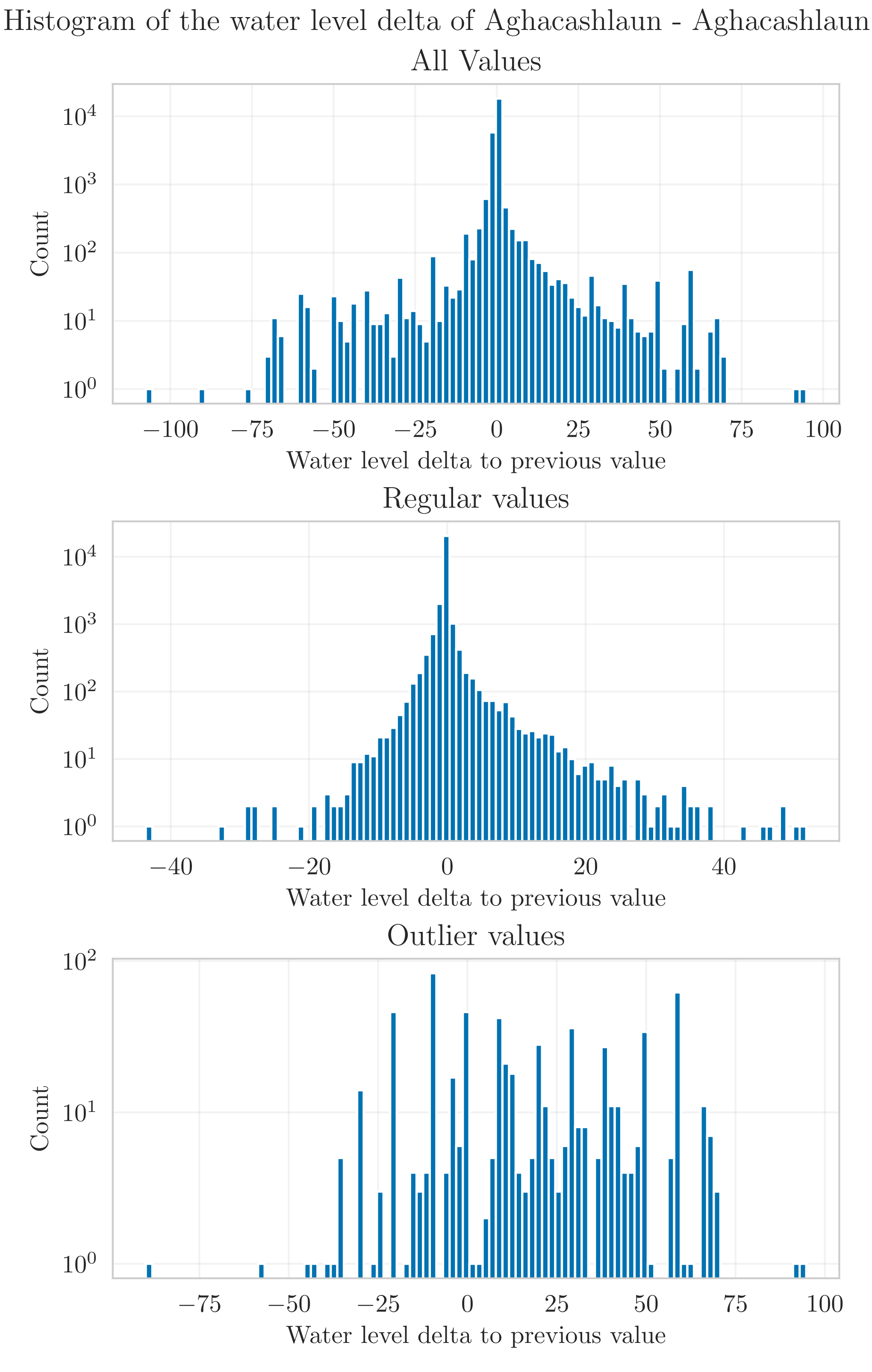

Figure 11: Histogram of the water level delta of Aghacashlaun - Aghacashlaun

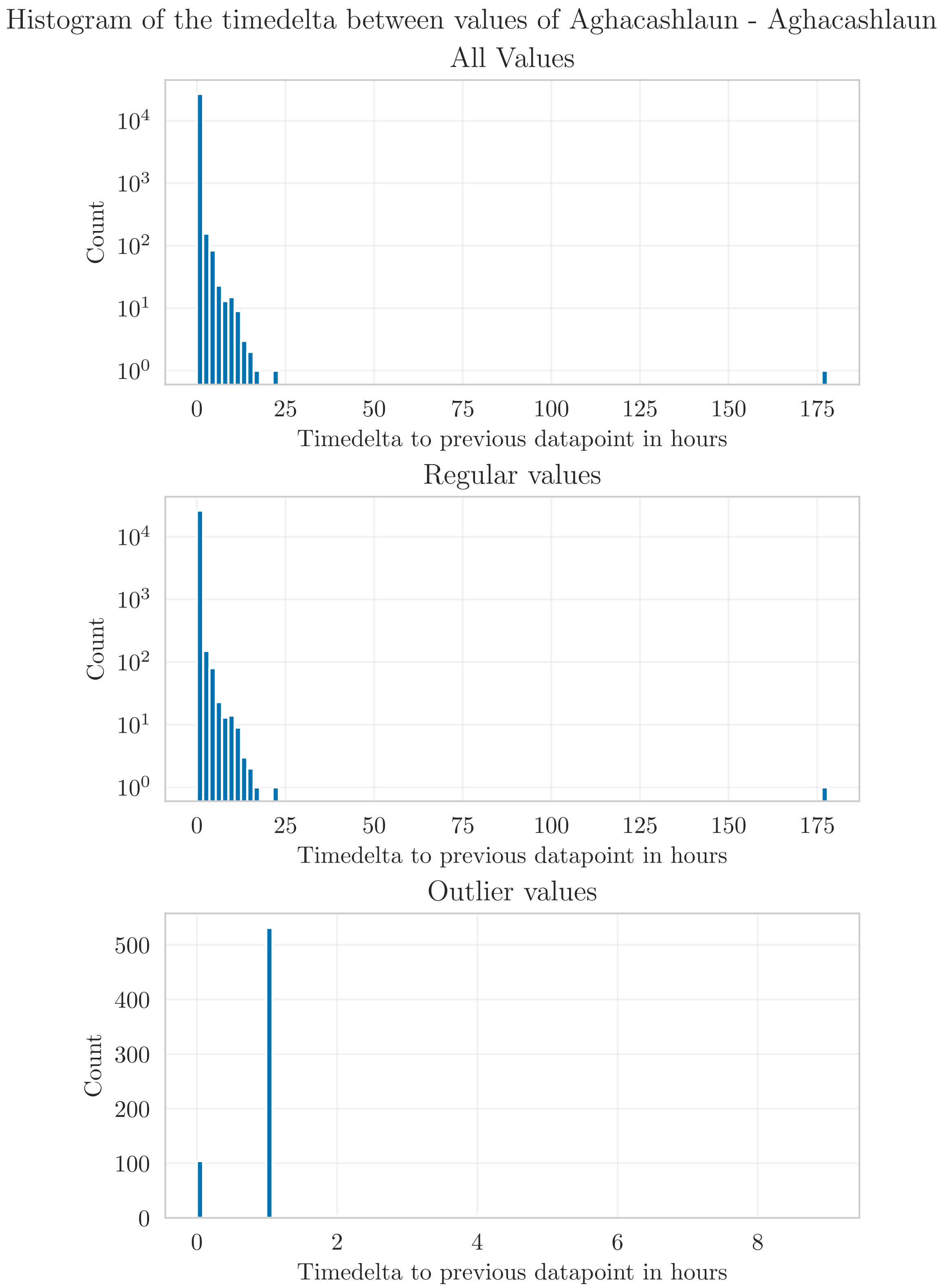

Figure 12: Histogram of the time delta (in hours) of Aghacashlaun - Aghacashlaun

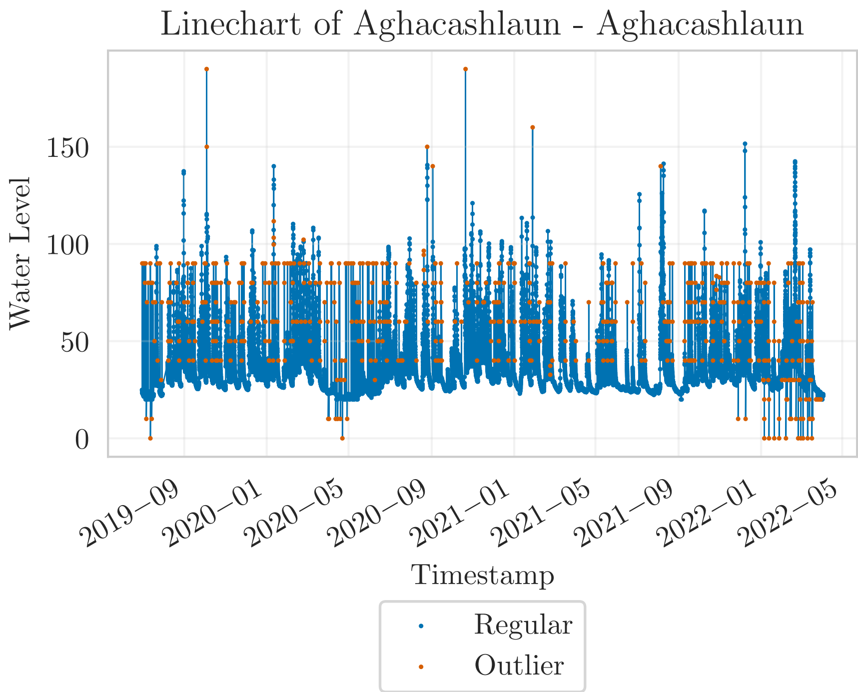

Figure 13: Line chart of Aghacashlaun - Aghacashlaun (whole data)

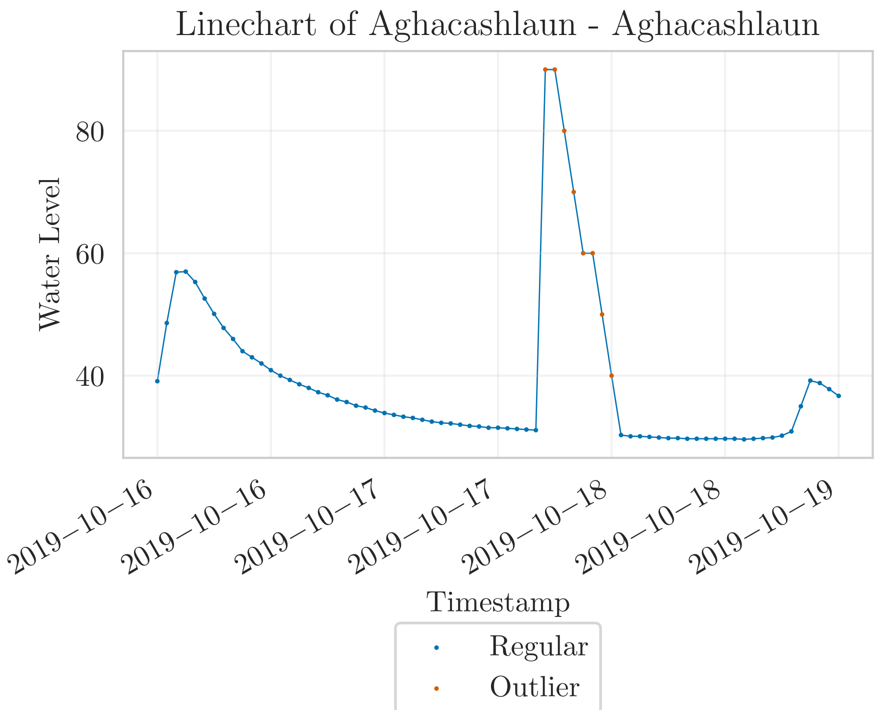

Figure 14: Line chart of Aghacashlaun - Aghacashlaun (2019-10-16 - 2019-10-19)

4.6 Outlier Detection performance Metrics

4.6.1 Confusion Matrix

To compare the performance of two classification methods a confusion matrix, shown in Table 7, can be used to provide an overview. A confusion matrix consists of the following elements (explained on the basis of outlier detection):

- True Positive (TP): actual class: outlier, predicted class: outlier
- False Negative (FN): actual class: outlier, predicted class: regular
- True Negative (TN): actual class: regular, predicted class: regular
- False Positive (FP): actual class: regular, predicted class: outlier

		Predicted label	
		Positive (P)	Negative (N)
True/ actual label	Positive	True positive	False negative
	Negative	False positive	True negative

Table 7: Example of binary confusion matrix

The green cells represent the correctly classified values (TP & TN) and the red cells represent the incorrectly classified values (FN & FP). The confusion matrix is the basis of many more performance metrics.

4.6.2 Accuracy

To make the comparison easier, the confusion matrix can be aggregated into one single value as a metric. There are numerous ways how this single metric can be calculated. One of the Most common and basic methods is the Accuracy:

$$Accuracy = \frac{Number\,of\,correct\,predictions}{Total\,number\,of\,predictions} = \frac{TP+TN}{TP+FN+TP+FP} = \frac{TP+TN}{P+N} \tag{12}$$

The accuracy provides information what percentage of values are correctly classified. However if the classes are not equally distributed, the accuracy is a bad metric. This is especially true for heavily imbalanced classes. For example if the dataset has $1\,000\,000$ positive values and 10 negative, when optimizing for a high accuracy it can happen that the classifier predicts every value as positive and still achieves a very high accuracy:

$$Accuracy = \frac{1\,000\,000+0}{1\,000\,000+10} = 0.99999 \tag{13}$$

Thus for measuring the performance of outliers, the accuracy is not a suitable metric.

4.6.3 Precision and Recall

Figure 15 shows a visual comparison between precision and recall. The precision provides information about what percentage of positive predictions were actually correct. Whereas the recall provides information about what percentage of all positive values were actually predicted as such. Precision and Recall are used to calculate the F-Score described in subsection 4.6.4.

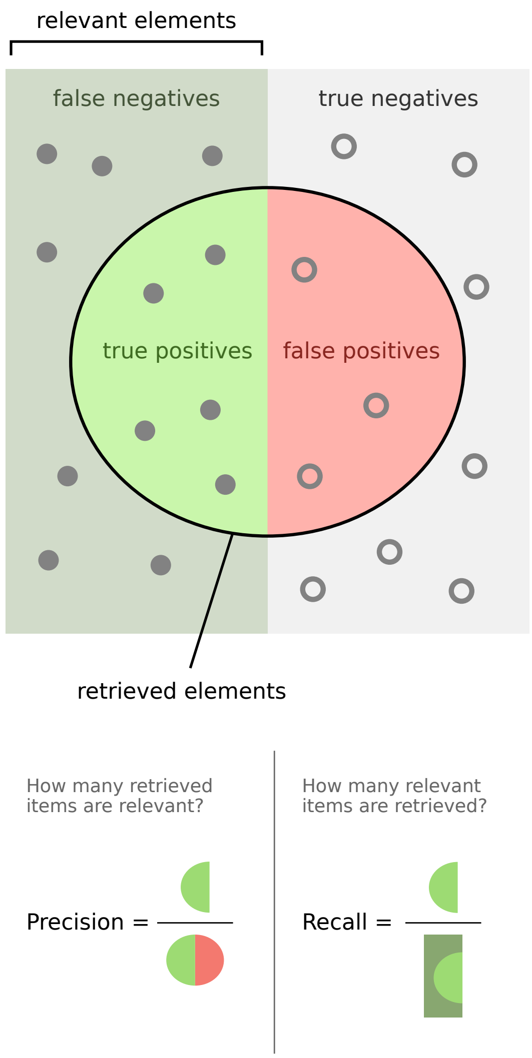

Figure 15: Precision vs Recall [29]

Precision

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

Recall

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

4.6.4 F-score

The $F_1 - score$ is the harmonic mean of the precision and recall.

$$F_1 - score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2}{\frac{Precision+Recall}{Precision \cdot Recall}} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{16}$$

[30, 31]

The $F_\beta - score$ is more sophisticated compared to the $F_1 - score$ where an additional parameter (β) needs to be chosen, which is used to weight the precision.

$$F_\beta - score = \frac{(\beta^2 + 1) \cdot Precision \cdot Recall}{\beta^2 \cdot Precsion + Recall} \tag{17}$$

[31, 32]

The $F_1 - score$ uses 1 for β so the precision and the recall are equally weighted.

4.7 Implementation of different Outlier Detection Approaches

The input data structure for the different approaches is a pandas DataFrame. [18] The structure of the DataFrame is equal for all measurement stations. It contains the following columns:

- `water_level`: the water level of the river in cm (datatype: float64)
- `timestamp`: the date and time of the datapoint in UTC (datatype: datetime64[ns, UTC])
- `is_outlier`: true or false depending if the value is an outlier (datatype: bool)

To test the performance of the models the column `is_outlier` contains the ground truth, which was manually labeled. An example of the input data can be seen in Table 8.

To speed up the process of calculating the result of the outlier detection method ($|x_t - \hat{x}_t|$, the z-score or the modified z-score) is all done in one step and in the second step the result of this calculation is compared against a range of thresholds.

4.7.1 Mean Threshold

Calculating $\hat{x}_t$ using the mean for a pandas DataFrame [18] is quite straightforward and is shown in Listing 5. To reduce the amount of duplicate code, the imports are only included in Listing 5.

4.7.2 Median Threshold

The calculation of the median is quite similar to the mean. It is shown in Listing 6.

Listing 5 First step of classifying outliers using the mean

```python
from typing import Union

import numpy as np
import pandas as pd


def mean_outlier_detection(input_df: pd.DataFrame,
                           window: Union[int, None],
                           center_window: bool):
    """
    Detects outliers in a dataframe using a (moving) average.
    :param input_df: the input dataframe where the values are stored
                     in the column water_level
    :param window: the size of the window, None if no window should
                   be used
    :param center_window: whether the window should be centered or not
    :return: a copy of the input dataframe where the column result
             should be compared to a threshold to detect outliers
    """
    od_df = input_df.copy()
    if window is None:
        od_df['x_hat'] = od_df['water_level'].mean()
    else:
        od_df['x_hat'] = \
            od_df['water_level'].rolling(window=window,
                                         center=center_window,
                                         min_periods=1).mean()
    od_df['result'] = np.abs(od_df['water_level'] - od_df['x_hat'])
    return od_df
```

water_level	timestamp	is_outlier
24.9	2019-06-30 15:00:00+00:00	False
24.9	2019-06-30 16:00:00+00:00	False
24.8	2019-06-30 17:00:00+00:00	False
24.4	2019-06-30 18:00:00+00:00	False
24.4	2019-06-30 19:00:00+00:00	False
24.3	2019-06-30 20:00:00+00:00	False
24.1	2019-06-30 21:00:00+00:00	False
24.0	2019-06-30 22:00:00+00:00	False
90.0	2019-06-30 23:00:00+00:00	True

Table 8: First 9 values of Aghacashlaun - Aghacashlaun

Listing 6 First step of classifying outliers using the median

```
def median_outlier_detection(input_df: pd.DataFrame,
                             window: Union[int, None],
                             center_window: bool):
    """
    Detects outliers in a dataframe using a (moving) average.
    :param input_df: the input dataframe where the values are stored
                     in the column water_level
    :param window: the size of the window, None if no window should
                   be used
    :param center_window: whether the window should be centered or not
    :return: a copy of the input dataframe where the column result
             should be compared to a threshold to detect outliers
    """
    od_df = input_df.copy()
    if window is None:
        od_df['x_hat'] = od_df['water_level'].median()
    else:
        od_df['x_hat'] = \
            od_df['water_level'].rolling(window=window,
                                         center=center_window,
                                         min_periods=1).median()
    od_df['result'] = np.abs(od_df['water_level'] - od_df['x_hat'])
    return od_df
```

Listing 7 First step of classifying outliers using the MAD

```
def mad_outlier_detection(input_df: pd.DataFrame,
                          window: Union[int, None],
                          center_window: bool):
    """
    Detects outliers in a dataframe using a (moving) average.
    :param input_df: the input dataframe where the values are stored
                     in the column water_level
    :param window: the size of the window, None if no window should
                   be used
    :param center_window: whether the window should be centered or not
    :return: a copy of the input dataframe where the column result
             should be compared to a threshold to detect outliers
    """
    od_df = input_df.copy()
    if window is None:
        od_df['x_hat'] = np.median(
            np.abs(od_df['water_level'] - np.median(
                od_df['water_level'])))
    else:
        od_df['x_hat'] = \
            od_df['water_level'].rolling(window=window,
                                         center=center_window,
                                         min_periods=1).apply(
                lambda x: np.median(np.abs(x - np.median(x))))
    od_df['result'] = np.abs(od_df['water_level'] - od_df['x_hat'])
    return od_df
```

4.7.3 MAD Threshold

In Listing 7 an example for an implementation for calculating the MAD is provided.

4.7.4 Z-Score

Listing 8 shows how to calculate th z-score in Python.

4.7.5 Modified z-score

The code for calculating the result of the modified z-score (using the MADN) is shown in Listing 9.

Listing 8 First step of classifying outliers using the z-score

```python
def z_score_outlier_detection(input_df: pd.DataFrame,
                              window: Union[int, None],
                              center_window: bool):
    """
    Detects outliers in a dataframe using a (moving) average.
    :param input_df: the input dataframe where the values are stored
                     in the column water_level
    :param window: the size of the window, None if no window should
                   be used
    :param center_window: whether the window should be centered or not
    :return: a copy of the input dataframe where the column result
             should be compared to a threshold to detect outliers
    """
    od_df = input_df.copy()
    if window is None:
        od_df['mean'] = od_df['water_level'].mean()
        od_df['std'] = od_df['water_level'].std()
    else:
        od_df['mean'] = \
            od_df['water_level'].rolling(window=window,
                                         center=center_window,
                                         min_periods=1).mean()
        od_df['std'] = \
            od_df['water_level'].rolling(window=window,
                                         center=center_window,
                                         min_periods=1).std()
    od_df['result'] = \
        (od_df['water_level'] - od_df['mean']).divide(od_df['std'])
    return od_df
```

Listing 9 First step of classifying outliers using the modified z-score (MADN-z-score)

```
def madn_z_score_outlier_detection(input_df: pd.DataFrame,
                                   window: Union[int, None],
                                   center_window: bool):
    """
    Detects outliers in a dataframe using a (moving) average.
    :param input_df: the input dataframe where the values are stored
                     in the column water_level
    :param window: the size of the window, None if no window should
                   be used
    :param center_window: whether the window should be centered or not
    :return: a copy of the input dataframe where the column result
             should be compared to a threshold to detect outliers
    """
    od_df = input_df.copy()
    if window is None:
        od_df['median'] = od_df['water_level'].median()
        od_df['mad'] = np.median(
            np.abs(od_df['water_level'] - od_df['median']))
        od_df['madn'] = od_df['mad'] / 0.6745
    else:
        od_df['median'] = \
            od_df['water_level'].rolling(window=window,
                                         min_periods=1,
                                         center=center_window).median()
        od_df['mad'] = \
            od_df['water_level'].rolling(window=window,
                                         min_periods=1,
                                         center=center_window).apply(
                lambda x: np.median(np.abs(x - np.median(x))))
        od_df['madn'] = od_df['mad'] / 0.6745
    od_df['result'] = \
        (od_df['water_level'] - od_df['median']).abs() \
            .divide(od_df['madn'])
    return od_df
```

4.7.6 Preprocessing the Data

In the preprocessing step outliers which extremely vary from the usual trend of the other values were removed. This was done by defining and upper and lower limit for the data. If a value is not inside this limit it is removed from the dataset. The thought behind preprocessing the data was to increase the model performance by removing extreme outlier values from the beginning. The disadvantage of the preprocessing step is, that two limits need to be defined for each measurement station. The limits need to be chosen carefully. On the one hand, if the valid value range is too large, no extreme outliers are removed and the preprocessing is useless, on the other hand if the valid value range is too small the preprocessing might remove valid values, which would indicate a possible flood. Thus it is also a good idea to regularly check and maybe update those limits.
Core Url: https://github.com/cellularegg/bachelor-thesis-code/blob/main/preprocessing.ipynb

add code to appendix

4.7.7 Finding Parameters

To find the best parameters a grid search was used. The first step was to calculate the results for different methods, window sizes and types (centered an non centered). For each unique parameter combination the result dataframe was stored as a file, where the filename provided information about the parameters. An example for a filename would be "`12_cw_median.parquet`", where "`12`" is the window size, "`cw`" stands for center window ("`nocw`" is for no center window) and "`median`" is the method used. This was done for every unique combination of parameters. The possible values for each parameter, that was used, is listed below.

- **normalized**: yes, no
- **preprocessed**: yes, no
- **window size**: None, 2-51 (in steps of one)
- **centered window**: yes, no
- **method**: mean, median, mad, z-score, modified z-score (madn-z-score)
- **common-id**: "36022-ie", "39003-ie", "2386-ch", "42960105-de", "2720050000-de"

Due to the large number of unique combinations of parameters the size of the resulting files was quite large (multiple Gigabytes). The grid search was executed in parallel using multiprocessing, in order to speed up the process.
After the results of each parameter was calculated, for each file a range of thresholds was tested and the performance (confusion matrix and $F_1 - score$) of this threshold and the parameters used were saved to a DataFrame. This was also conducted in parallel to decrease the runtime.
Url of Code: https://github.com/cellularegg/bachelor-thesis-code/blob/main/threshold_based_outlier_detection.py

include code in appendix

4.8 Compare different Outlier Detection Approaches

As expected, using a centered window yields the best $F_1 - score$. Furthermore the methods with the best performance are the median and the MADN z-score. While the median performed best with smaller window sizes (5-7) the modified z-score performed best with larger window sizes (22-28). Using the MAD to calculate the $\hat{x}_t$ yields the worst performance.
Table 9 shows the top 5 average $F_1 - scores$ of all stations tested. When using one set of parameters for different measurement stations the median performed best. Using the mean for $\hat{x}_t$ delivered the second best performance, when comparing shared parameters bewteen all tested stations. However the average $F_1 - score$ of the mean is only 0.47794, it was reached by using a centered window of size 5, a threshold of 18.87960 and not normalizing the data. Due to the heterogeneity of the fluctuations of the water levels for the different stations it is not recommended to use the same parameters for different stations. Similar or equal parameters should only be used when the water levels of two measurement stations behave similarly. Additionally when using the same parameters for different stations the performance of those parameters should be looked at per station and not as an average of the $F_1 - scores$. This hinders the fact that one model performs perfectly ($F_1 - score = 1$) and the other very poorly ($F_1 - score = 0.5$).

change average to harmonic mean?

window_size	center_window	normalized	threshold	model_type	average_f1_score
3.0	True	False	6.628763	median	0.725722
3.0	True	False	6.959866	median	0.725293
3.0	True	False	6.297659	median	0.723374
3.0	True	False	8.284281	median	0.719480
3.0	True	False	8.615385	median	0.719397

Table 9: Best parameters of the average F1-score of all stations tested

Using the preprocessing (described in subsection 4.7.6) usually resulted in a lower performance, when comparing the hightest $F_1 - scores$ for both datasets (preprocessed and not preprocessed). The reason for this is, that the outliers removed by the two sided filter were detected as outliers anyway. So just the number of TPs was reduced. Thus resulting in a slightly lower overall performance, since the outliers were completely removed from the dataset in the preprocessing step.

add specific examples for FP and FN

Mean

Figure 16 shows the best performing outlier detection using mean. The $F_1 - score$ is about 0.68.

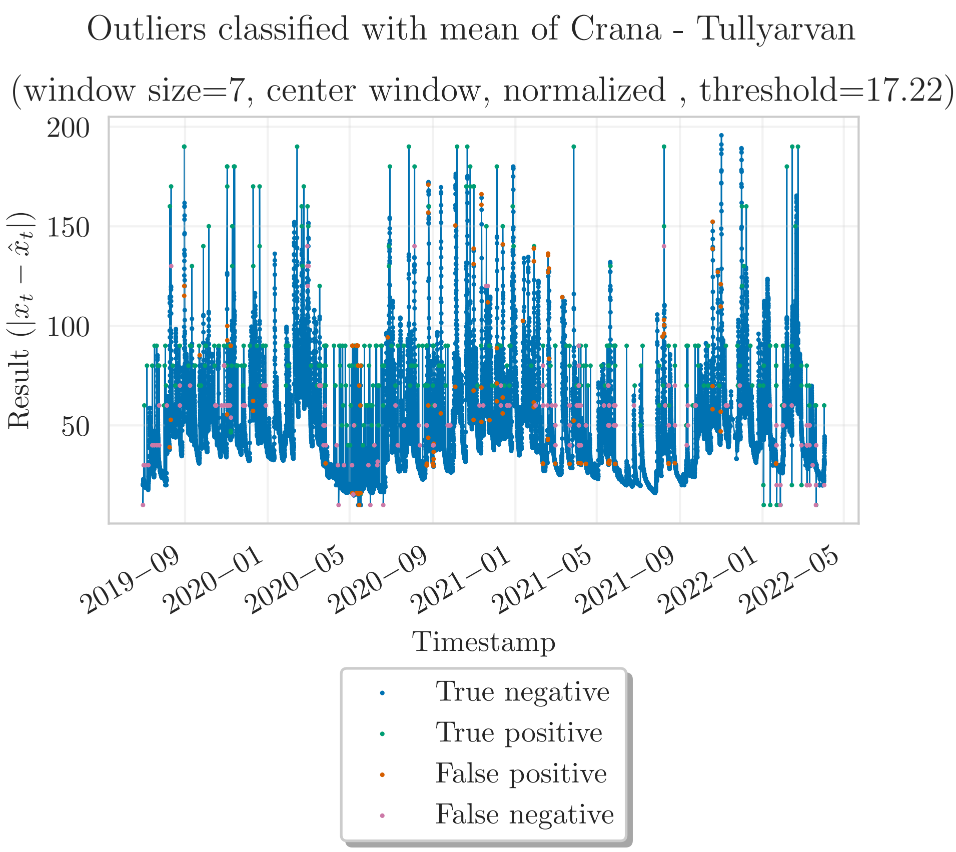

Figure 16: Best performance of outlier detection using mean (Crana - Tullyarvan)

Median

Figure 17 shows the best performing outlier detection using mean. The $F_1 - score$ is about 0.91.

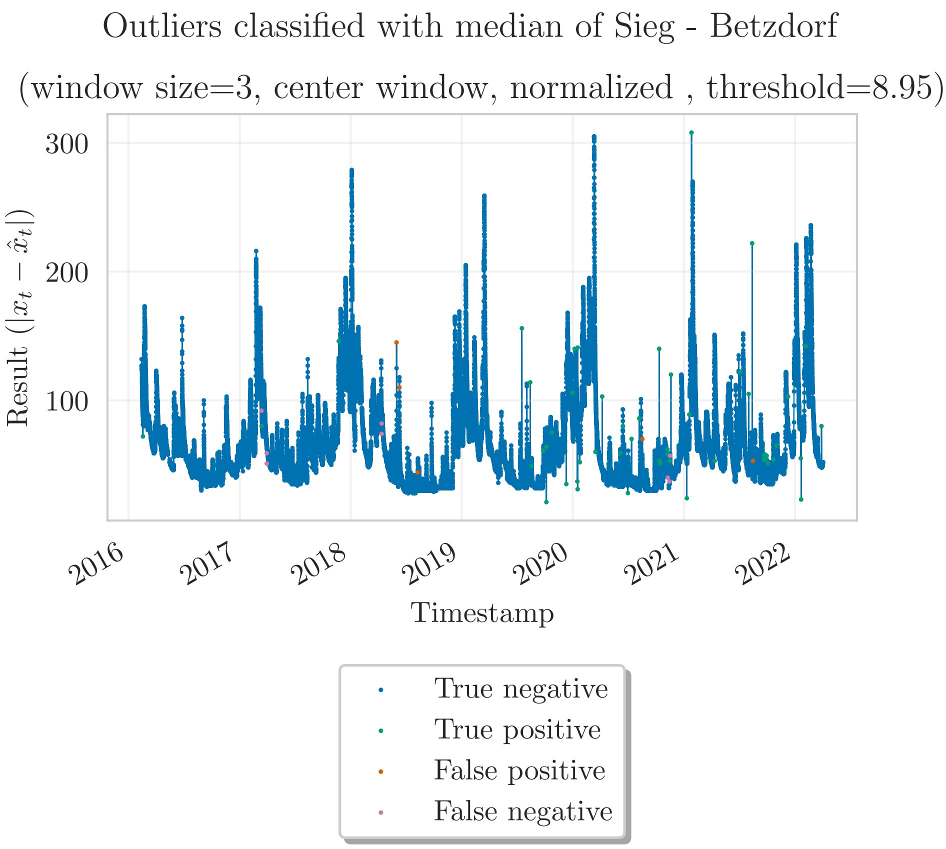

Figure 17: Best performance of outlier detection using median (Sieg - Betzdorf)

MAD

Figure 18 shows the best performing outlier detection using mean. The $F_1 - score$ is about 0.45.

Z-score

Figure 19 shows the best performing outlier detection using mean. The $F_1 - score$ is about 0.68.

MADN-z-score

Figure 20 shows the best performing outlier detection using mean. The $F_1 - score$ is about 0.79.

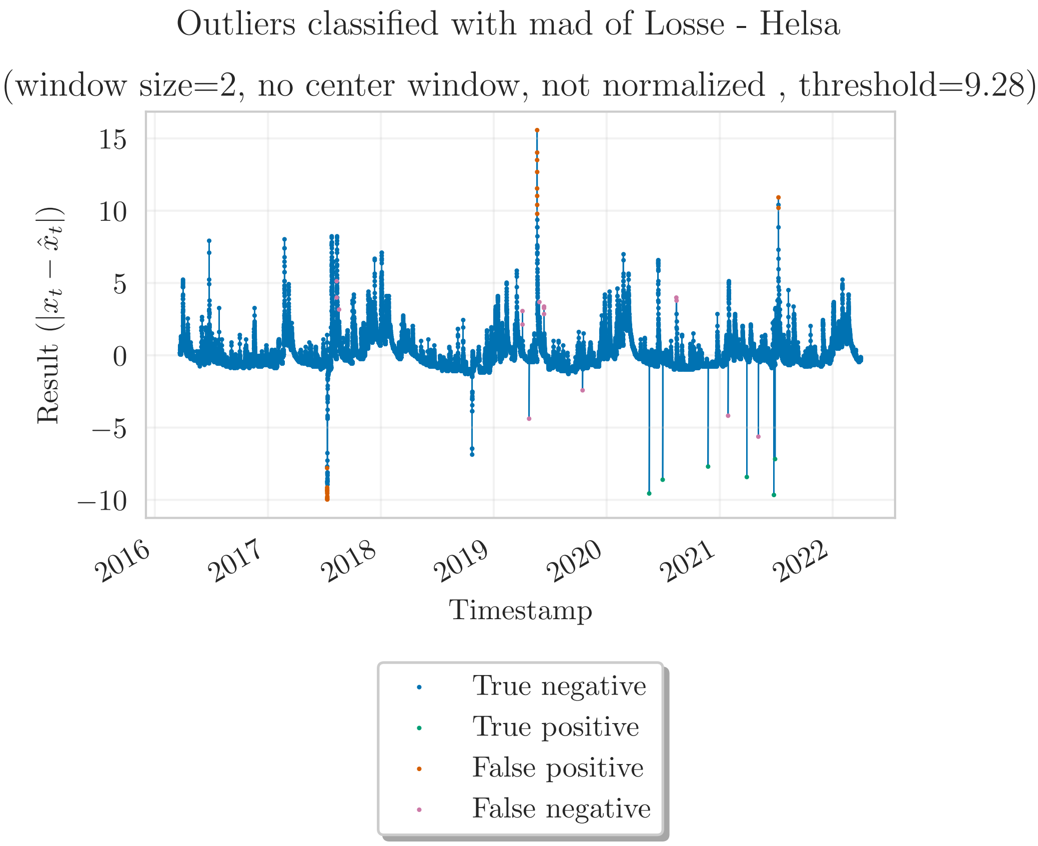

Figure 18: Best performance of outlier detection using mad (Losse - Helsa)

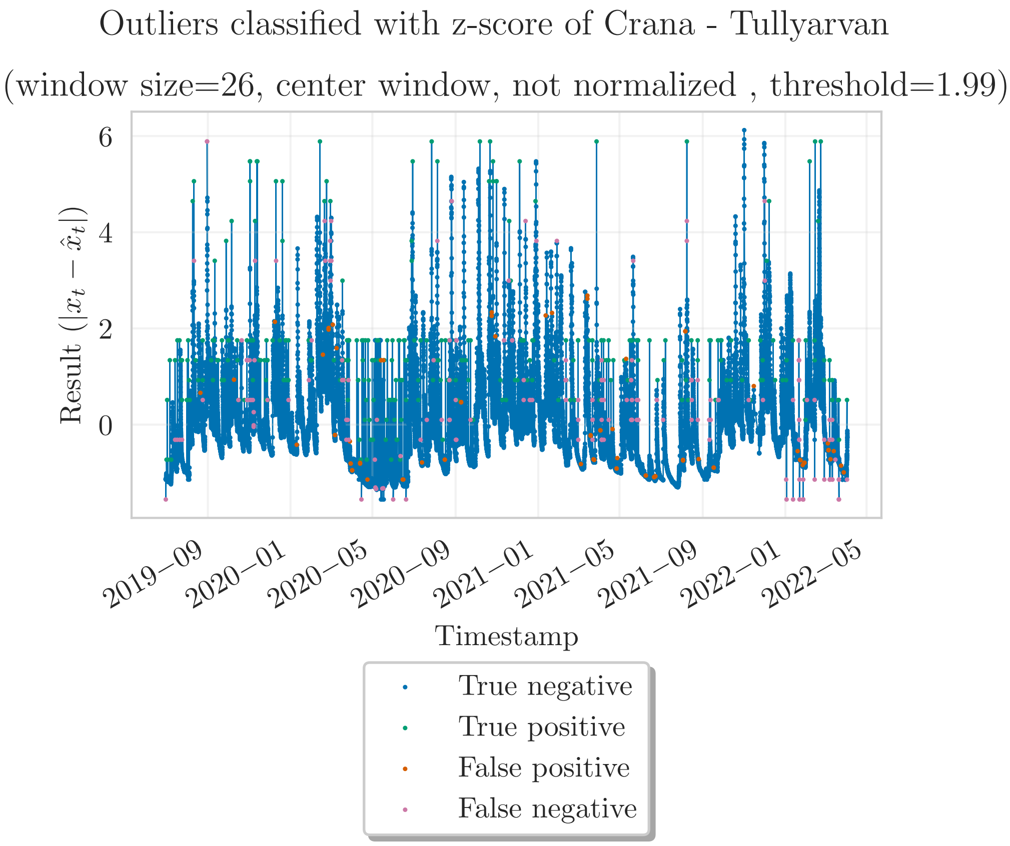

Figure 19: Best performance of outlier detection using mad (Crana - Tullyarvan)

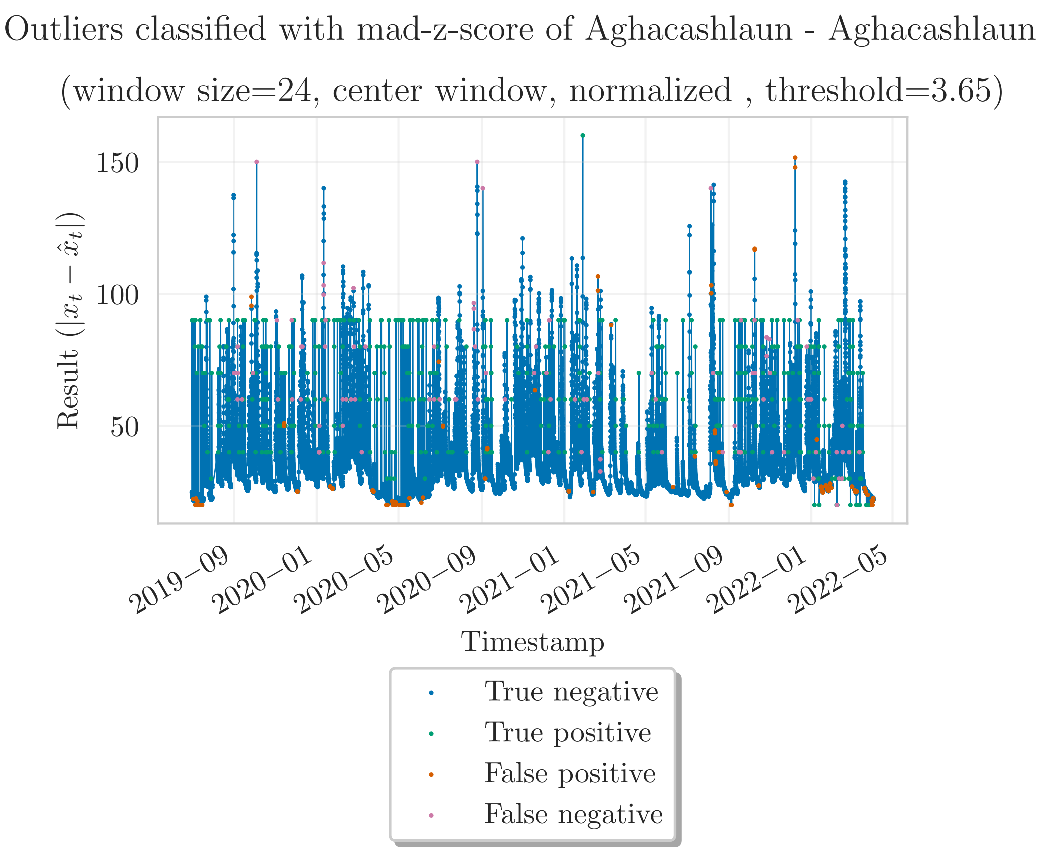

Figure 20: Best performance of outlier detection using MADN-z-score (Aghacashlaun - Aghacashlaun)

4.8.1 Station 36022-ie

Table 10 shows the top three predictions per model for normalized and not normalized data. It clearly pictures, that methods using a centered moving window perform better. It is also interesting to see that the MADN z-score and the regular z-score performed equally well regardless of the fact that the data is normalized or not.

think of explanation for that.

change mad-z-score to madn-z-score

4.8.2 Station 2386-ch

In Table 11 the top three predictions per model for normalized and not not normalized data is shown. The outlier detection for this station has the worst performance among those tested. The result of this model is shown in Figure 21. Every value above the threshold is classified as an outlier. Every value below the threshold is classified as a regular value.

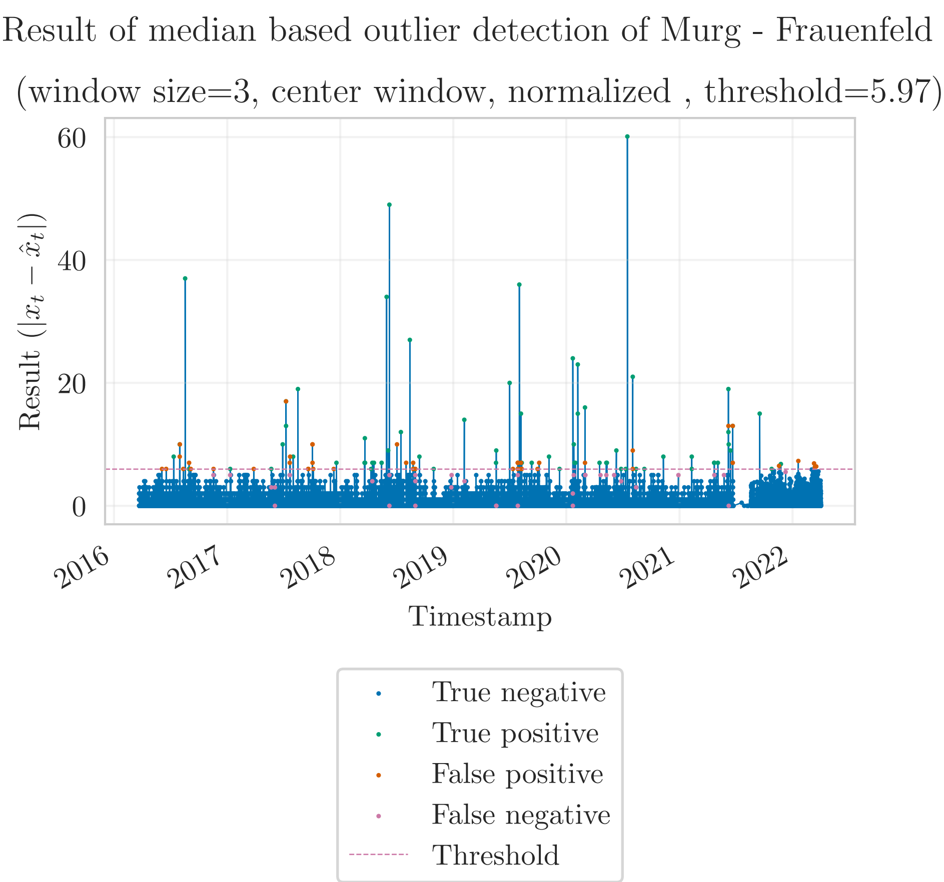

Figure 21: Result of median based outlier detection Murg - Frauenfeld

window_size	center_window	normalized	threshold	model_type	f1_score
5.0	True	False	7.622074	median	0.812550
5.0	True	False	8.284281	median	0.812348
5.0	True	False	7.953177	median	0.811688
24.0	True	True	3.648829	mad-z-score	0.789205
24.0	True	False	3.648829	mad-z-score	0.789205
22.0	True	True	3.648829	mad-z-score	0.787572
22.0	True	False	3.648829	mad-z-score	0.787572
26.0	True	True	4.311037	mad-z-score	0.786415
26.0	True	False	4.311037	mad-z-score	0.786415
9.0	True	True	1.000000	median	0.759430
7.0	True	True	1.000000	median	0.750670
8.0	True	True	1.000000	median	0.749358
24.0	True	False	1.993311	z-score	0.626321
32.0	True	True	2.324415	z-score	0.622718
32.0	True	False	2.324415	z-score	0.621457
23.0	True	True	1.993311	z-score	0.621359
27.0	True	True	2.324415	z-score	0.620833
27.0	True	False	2.324415	z-score	0.620833
9.0	True	False	16.892977	mean	0.619647
10.0	True	False	18.879599	mean	0.619173
10.0	True	False	19.210702	mean	0.618538
11.0	True	True	1.331104	mean	0.611993
9.0	True	True	1.331104	mean	0.611418
10.0	True	True	1.331104	mean	0.610659
24.0	True	True	1.993311	mad	0.404255
26.0	True	True	1.993311	mad	0.404000
28.0	True	True	1.993311	mad	0.403183
22.0	True	False	66.889632	mad	0.401679
22.0	True	False	67.220736	mad	0.401126
22.0	True	False	67.551839	mad	0.400856

Table 10: Top predictions summary of Aghacashlaun - Aghacashlaun

window_size	center_window	normalized	threshold	model_type	f1_score
3.0	True	False	5.966555	median	0.657895
3.0	True	False	5.635452	median	0.649351
3.0	True	False	6.628763	median	0.639175
3.0	True	False	4.642140	mean	0.447917
3.0	True	False	4.973244	mean	0.443804
3.0	True	False	5.304348	mean	0.429907
50.0	True	True	3.979933	z-score	0.327273
50.0	True	False	3.979933	z-score	0.327273
49.0	True	True	3.648829	z-score	0.306931
49.0	True	False	3.648829	z-score	0.306931
48.0	True	False	3.648829	z-score	0.305419
50.0	True	True	3.648829	z-score	0.303922
47.0	True	True	1.000000	median	0.120690
42.0	False	True	11.926421	median	0.120690
42.0	False	True	8.946488	median	0.120690
49.0	False	True	24.177258	mean	0.120690
34.0	True	True	9.277592	mean	0.120690
34.0	True	True	6.959866	mean	0.120690
NaN	True	True	61.591973	mad-z-score	0.120690
NaN	False	True	97.351171	mad-z-score	0.120690
NaN	False	True	98.013378	mad-z-score	0.120690
NaN	True	False	61.591973	mad-z-score	0.120690
NaN	False	False	97.351171	mad-z-score	0.120690
NaN	False	False	98.013378	mad-z-score	0.120690
7.0	True	True	1.000000	mad	0.120690
11.0	True	True	1.993311	mad	0.120690
34.0	False	True	24.839465	mad	0.120690
7.0	True	False	1.000000	mad	0.004301
27.0	True	False	4.973244	mad	0.004301
27.0	True	False	5.635452	mad	0.004301

Table 11: Top predictions summary of Murg - Frauenfeld

4.8.3 Station 2720050000-de

In Table 12 the top three predictions per model for normalized and not not normalized data is shown. With an $F_1 - score$ of 0.905109 this station had the best performance. The result of this model is shown in Figure 22. Every value above the threshold is classified as an outlier. Between 2018 and 2019 an example of a FP can be seen (the orange values above the threshold).

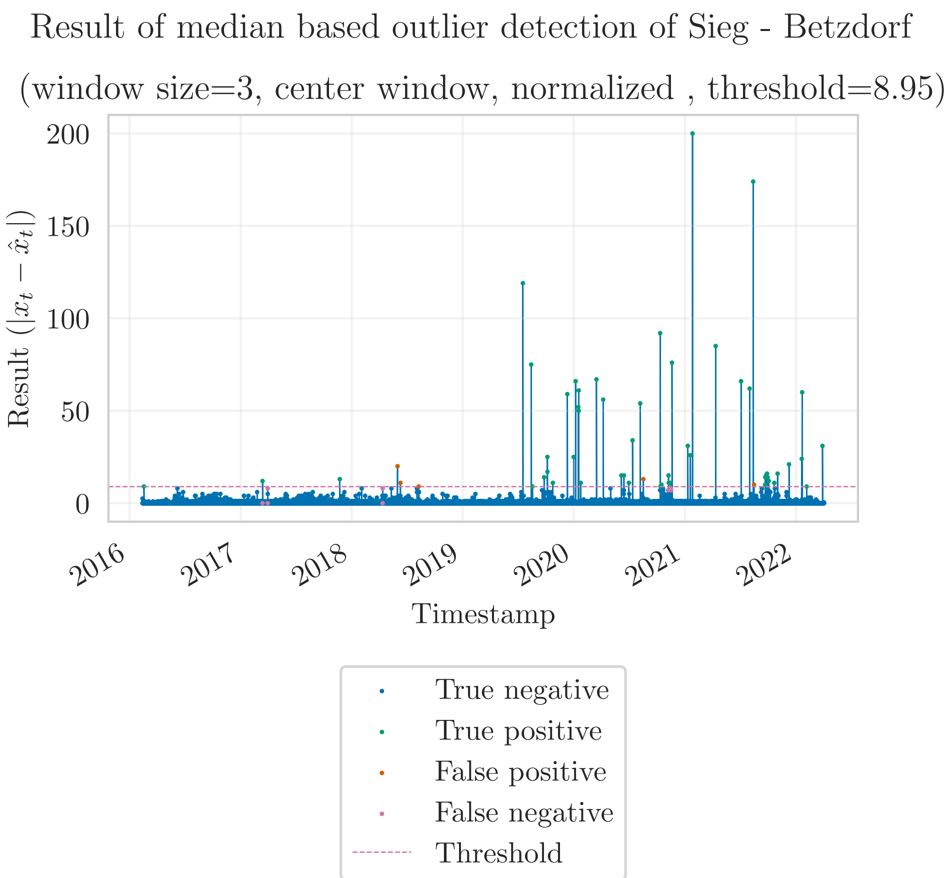

Figure 22: Result of median based outlier detection Sieg - Betzdorf

4.8.4 Station 42960105-de

In Table 14 the top three predictions per model for normalized and not not normalized data is shown.

4.8.5 Station 39003-ie

In Table 15 the top three predictions per model for normalized and not not normalized data is shown.

Table 12: Top predictions summary of Sieg - Betzdorf

window_size	center_window	normalized	threshold	model_type	f1_score
3.0	True	False	8.284281	median	0.905109
3.0	True	False	8.615385	median	0.905109
3.0	True	False	8.946488	median	0.905109
29.0	True	False	3.648829	z-score	0.545455
23.0	True	False	3.317726	z-score	0.545455
27.0	True	False	3.648829	z-score	0.542373
22.0	True	True	3.317726	z-score	0.533333
28.0	True	True	3.648829	z-score	0.533333
20.0	True	True	3.317726	z-score	0.516667
3.0	True	False	7.290970	mean	0.510288
3.0	True	False	6.959866	mean	0.510121
3.0	True	False	6.628763	mean	0.503937
NaN	True	True	10.602007	mad-z-score	0.133333
NaN	False	True	10.602007	mad-z-score	0.133333
NaN	True	False	10.602007	mad-z-score	0.133333
NaN	False	False	10.602007	mad-z-score	0.133333
NaN	True	True	10.270903	mad-z-score	0.119048
NaN	True	False	10.270903	mad-z-score	0.119048
47.0	True	True	1.000000	median	0.108108
29.0	False	True	1.000000	median	0.108108
50.0	False	True	2.324415	median	0.108108
NaN	False	True	1.662207	mean	0.108108
NaN	True	True	1.662207	mean	0.108108
NaN	True	True	1.000000	mean	0.108108
7.0	True	True	1.000000	mad	0.108108
31.0	True	True	1.000000	mad	0.108108
25.0	False	True	2.324415	mad	0.108108
49.0	True	False	100.000000	mad	0.006390
50.0	True	False	100.000000	mad	0.006379
47.0	True	False	100.000000	mad	0.006360

Table 13: Top predictions summary of Losse - Helsa

window_size	center_window	normalized	threshold	model_type	f1_score
6.0	True	True	2.655518	median	0.731707
4.0	True	False	25.170569	median	0.731707
4.0	True	False	26.494983	median	0.731707
4.0	True	False	26.163880	median	0.731707
3.0	True	True	1.000000	median	0.720000
4.0	True	True	2.324415	median	0.714286
19.0	True	True	5.966555	mean	0.571429
17.0	True	True	5.635452	mean	0.571429
15.0	True	True	5.304348	mean	0.571429
15.0	True	False	62.585284	mean	0.571429
16.0	True	False	62.585284	mean	0.571429
16.0	True	False	59.605351	mean	0.571429
2.0	False	True	9.277592	mad	0.454545
2.0	True	True	9.277592	mad	0.454545
2.0	False	True	9.939799	mad	0.439024
NaN	True	True	18.879599	z-score	0.275862
NaN	True	True	17.555184	z-score	0.275862
NaN	False	True	16.892977	z-score	0.275862
NaN	False	False	19.541806	z-score	0.275862
NaN	False	False	16.561873	z-score	0.275862
NaN	False	False	15.899666	z-score	0.275862
NaN	True	True	32.454849	mad-z-score	0.275862
NaN	True	True	31.792642	mad-z-score	0.275862
NaN	True	True	29.474916	mad-z-score	0.275862
NaN	True	False	32.454849	mad-z-score	0.275862
NaN	True	False	31.792642	mad-z-score	0.275862
NaN	True	False	29.474916	mad-z-score	0.275862
NaN	True	False	100.000000	mad	0.005068
NaN	False	False	100.000000	mad	0.005068
NaN	True	False	99.668896	mad	0.004548

Table 14: Top predictions summary of Losse - Helsa

window_size	center_window	normalized	threshold	model_type	f1_score
5.0	True	False	8.615385	median	0.859035
5.0	True	False	9.277592	median	0.858500
5.0	True	False	8.946488	median	0.857685
34.0	True	True	3.648829	mad-z-score	0.736842
34.0	True	False	3.648829	mad-z-score	0.736842
34.0	True	True	4.311037	mad-z-score	0.733962
34.0	True	False	4.311037	mad-z-score	0.733962
32.0	True	True	4.642140	mad-z-score	0.731707
32.0	True	False	4.642140	mad-z-score	0.731707
11.0	True	True	1.000000	median	0.723699
10.0	True	True	1.000000	median	0.723059
9.0	True	True	1.000000	median	0.720189
7.0	True	False	17.224080	mean	0.684211
7.0	True	False	16.230769	mean	0.683398
7.0	True	False	16.892977	mean	0.683267
26.0	True	True	1.993311	z-score	0.677668
26.0	True	False	1.993311	z-score	0.677668
27.0	True	True	1.993311	z-score	0.672489
27.0	True	False	1.993311	z-score	0.672489
25.0	True	True	1.993311	z-score	0.669633
25.0	True	False	1.993311	z-score	0.669633
8.0	True	True	1.000000	mean	0.643868
9.0	True	True	1.000000	mean	0.636678
10.0	True	True	1.000000	mean	0.633596
36.0	True	False	85.431438	mad	0.180556
34.0	True	False	85.431438	mad	0.180149
36.0	True	False	85.100334	mad	0.179840
30.0	True	True	1.662207	mad	0.173418
14.0	False	True	1.662207	mad	0.173355
29.0	True	True	1.662207	mad	0.173149

Table 15: Top predictions summary of Crana - Tullyarvan

5 Conclusion

This paper provides an overview of the topic anomaly detection. It provides a description for key features of data quality, and introduces the topic of data cleaning and data cleansing. Furthermore this paper provides general overview of outlier / anomaly detection approaches. Lastly the threshold based outlier detection is further elaborated.
There are countless methods to detect anomalies in data. There is not a go-to approach that suits all needs. It is required to assess different approaches for different applications, in order to get the best result. This paper should provide an overview of approaches to detect outliers / anomalies. It depends on the use case which method to detect outliers has the highest success rate.

Ich schlage hier vor auch einen Teil der quantitativen Analyse zusammenzufassen. Dabei sollten die Methoden und die Ergebnisse kurz skizziert werden. Vielleicht lassen sich auch Entscheidungen ableiten wann welche Methode besser greift.

6 Future Work

Chapter about which additional approaches could be tested, e.g. setting a maximum gradient for both directions (one for rising and falling values) for each measurement station.

Should I also include this or is this not common for a bachelor thesis?

Die Methode wollte ich auch vorschlagen, aber der Umfang ist so schon gross genug.

7 Appendix

Basically include the whole repo: https://github.com/cellularegg/bachelor-thesis-code

include everything needed.

Bibliography

[1] J. Strassmayr, G. Öller, E. Bragante, and J. Öhlböck, “FloodAlert Water Levels and Warning.” [Online]. Available: https://pegelalarm.at/en/

[2] L. Cai and Y. Zhu, “The Challenges of Data Quality and Data Quality Assessment in the Big Data Era,” *Data Science Journal*, vol. 14, no. 0, p. 2, May 2015.

[3] S. Song and A. Zhang, “IoT Data Quality,” in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 3517–3518.

[4] “Apache Parquet,” Dec. 2021. [Online]. Available: https://parquet.apache.org/

[5] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos, “DynaMMo: Mining and summarization of coevolving sequences with missing values,” in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’09. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 507–516.

[6] J. I. Maletic and A. Marcus, “Data Cleansing: Beyond Integrity Analysis,” p. 10, 2000.

[7] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, “A review on Outlier/Anomaly detection in time series data,” *arXiv:2002.04236 [cs, stat]*, Feb. 2020.

[8] A. A. Cook, G. Mısırlı, and Z. Fan, “Anomaly Detection for IoT Time-Series Data: A Survey,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6481–6494, Jul. 2020.

[9] F. Giannoni, M. Mancini, and F. Marinelli, “Anomaly Detection Models for IoT Time Series Data,” *arXiv:1812.00890 [cs, eess]*, Nov. 2018.

[10] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Computing Surveys*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[11] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, “Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median,” *Journal of Experimental Social Psychology*, vol. 49, no. 4, pp. 764–766, Jul. 2013.

[12] S. Mehrang, E. Helander, M. Pavel, A. Chieh, and I. Korhonen, “Outlier detection in weight time series of connected scales,” in *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Nov. 2015, pp. 1489–1496.

[13] “Detection of Spatial Outlier by Using Improved Z-Score Test.” [Online]. Available: https://ieeexplore-1ieee-1org-100033cm6022a.han.technikum-wien.at/document/8862582/

[14] G. Teschl and S. Teschl, *Spezielle Stetige Verteilungen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 325–345. [Online]. Available: http://link.springer.com/10.1007/978-3-642-54274-9_12

[15] P. J. Rousseeuw and M. Hubert, “Anomaly detection by robust statistics,” *WIREs Data Mining and Knowledge Discovery*, vol. 8, no. 2, p. e1236, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1236

[16] “Standard score,” *Wikipedia*, Apr. 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Standard_score&oldid=1082326996

[17] I. Bae and U. Ji, “Outlier Detection and Smoothing Process for Water Level Data Measured by Ultrasonic Sensor in Stream Flows,” *Water*, vol. 11, no. 5, p. 951, May 2019. [Online]. Available: https://www.mdpi.com/2073-4441/11/5/951

[18] “Pandas documentation — pandas 1.4.2 documentation.” [Online]. Available: https://pandas.pydata.org/pandas-docs/version/1.4/index.html

[19] “GitHub.” [Online]. Available: https://github.com/

[20] J. Strassmayr, “Pegelalarm API wrapper,” SOBOS-GmbH, Apr. 2022. [Online]. Available: https://github.com/SOBOS-GmbH/pegelalarm_public_pas_doc

[21] “Hmac — Keyed-Hashing for Message Authentication — Python 3.9.12 documentation.” [Online]. Available: https://docs.python.org/3.9/library/hmac.html

[22] “Station Aghacashlaun - Aghacashlaun (36022-ie).” [Online]. Available: https://waterlevel.ie/0000036022/

[23] “Station Murg - Frauenfeld (2386-ch).” [Online]. Available: https://www.hydrodaten.admin.ch/en/2386.html

[24] “Station Sieg - Betzdorf (2720050000-de).” [Online]. Available: http://www.hochwasser-rlp.de/pegeluebersichten/einzelpegel/flussgebiet/sieg/darstellung/tabellarisch/pegel/BETZDORF

[25] “Station Losse - Helsa (42960105-de).” [Online]. Available: http://www.hlnug.de/static/pegel/wiskiweb2/stations/42960105/station.html

[26] “Station Crana - Tullyarvan (39003-ie).” [Online]. Available: https://waterlevel.ie/0000039003/

[27] “Dash Documentation & User Guide | Plotly.” [Online]. Available: https://dash.plotly.com/

[28] D. Zelenay, “Outlier detection for water level data,” Apr. 2022. [Online]. Available: https://github.com/cellularegg/bachelor-thesis-code

[29] “Precision and recall,” *Wikipedia*, Mar. 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=1080323102

[30] Y. Sasaki, “The truth of the F-measure,” p. 5.

[31] N. Chinchor, “MUC-4 evaluation metrics,” in *Proceedings of the 4th Conference on Message Understanding*, ser. MUC4 ’92. USA: Association for Computational Linguistics, Jun. 1992, pp. 22–29. [Online]. Available: https://doi.org/10.3115/1072064.1072067

[32] A. A. Taha and A. Hanbury, “Metrics for evaluating 3D medical image segmentation: Analysis, selection, and tool,” *BMC Medical Imaging*, vol. 15, no. 1, p. 29, Aug. 2015. [Online]. Available: https://doi.org/10.1186/s12880-015-0068-x

List of Figures

List of Tables

List of source codes

List of Abbreviations

IoT Internet of Things

CSV Comma Separated Values

JSON Java Script Object Notation

MAD Median Absolute Deviation

MADN Normalized Median Absolute Deviation

ANN Artificial Neuronal Network

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

ARIMA AutoRegressive Integrated Moving Average

GP Gaussian Process

HMAC Keyed-Hashing for Message Authentication

P Positive

N Negative

TP True Positive

TN True Negative

FP False Positive

FN False Negative

Lat Latitude

Long Longitude

IQR InterQuartile Range