FH University of Applied Sciences TECHNIKUM WIEN

# 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, April 24, 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, April 24, 2022 Signature

# Kurzfassung

Change this

TODO Add Kurzfassung

# Abstract

Change this

This paper analyzes methods for anomaly detection to cleanse data. An overview of some key features of data quality is provided in the beginning. The difference between data cleaning and data cleansing is elaborated. Additionally a few methods for anomaly detection (mainly outlier detection) are outlined. The hypothesis of this paper is: Which characteristics define data quality, with regard to IoT (Internet of Things) Sensors? Which methods are there to detect and clean or cleanse faulty data?

# Contents


**1 Overview** **1**
- 1.1 Introduction . . . 1
- 1.2 Research Question . . . 1
- 1.3 Research Method . . . 1

**2 Data Quality** **2**
- 2.1 Features of Data Quality . . . 2
- 2.2 Improving Data Quality . . . 3
- 2.3 Data Cleaning & Cleansing Approaches . . . 3
- 2.4 Data Cleaning . . . 4
- 2.5 Data Cleansing . . . 5

**3 Outlier Detection** **6**
- 3.1 Outlier Types . . . 6
- 3.2 Outlier Detection Approaches . . . 9
- 3.3 Threshold based Outlier Detection . . . 10
- 3.4 Outlier Detection using z-score . . . 11
- 3.5 Outlier Detection using modified z-score . . . 11

**4 Outlier Detection based on Water Level Data** **12**
- 4.1 What is the Goal? . . . 12
- 4.2 How to retrieve the Data (Description of the API) . . . 12
- 4.3 Overview of the Data . . . 16
- 4.4 Explorative Data Analysis . . . 16
- 4.5 Manually detect outliers for a subset of data . . . 16
- 4.6 Outlier Detection performance Metrics . . . 19
  - 4.6.1 Confusion Matrix . . . 19
  - 4.6.2 Accuracy . . . 20
  - 4.6.3 Precision and Recall . . . 20
  - 4.6.4 F-score . . . 21
- 4.7 Implement different Outlier Detection Approaches . . . 22
- 4.8 Compare different Outlier Detection Approaches . . . 22

**5 Conclusion** **23**
- 5.1 Advantages and Disadvantages of used Outlier Detection Methods . . . 23

**Bibliography** **24**

**List of Figures** **26**

**List of Tables** **27**

**List of source codes** **28**

**List of Abbreviations** **29**

# 1 Overview

## 1.1 Introduction

With the growing popularity of Internet of Things (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.

Update Introduction

## 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

## 1.3 Research Method

This paper is a literature research. To get an overview of the topic, papers related to: data cleaning, anomaly detection for IoT data / time series data and outlier detection methods were researched. After some base knowledge was established the major topics of the paper were defined. Subsequently more research was done on the major topics (Features of data quality, data cleaning & cleansing, outlier detection). To organize the references found while researching Zotero was used, with the Add-on Better BibTeX.

Adapt research method

# 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. [1, 2]

#### 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. [1]

#### 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. [1]

#### 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 [3], 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) [1]

## 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 [4]. For ANNs (artificial neural networks) Long Short-Term Memory (LSTM) (Long short-term memory) or Gated Recurrent Unit (GRU) (Gated recurrent unit) can be used to predict missing data. [2]

#### 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. [2]

#### Consistency

To facilitate consistent data, statistical smoothing or forecasting methods can be used. Examples methods are: AutoRegressive Integrated Moving Average (ARIMA) (Autoregressive integrated moving average) or Gaussian Process (GP) ( Gaussian Process). ANNs can also be used to unify the time series interval between datapoints. [2]

#### 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.
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 [5]

| `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. [6] Examples of three point outliers are shown in Figure 1.

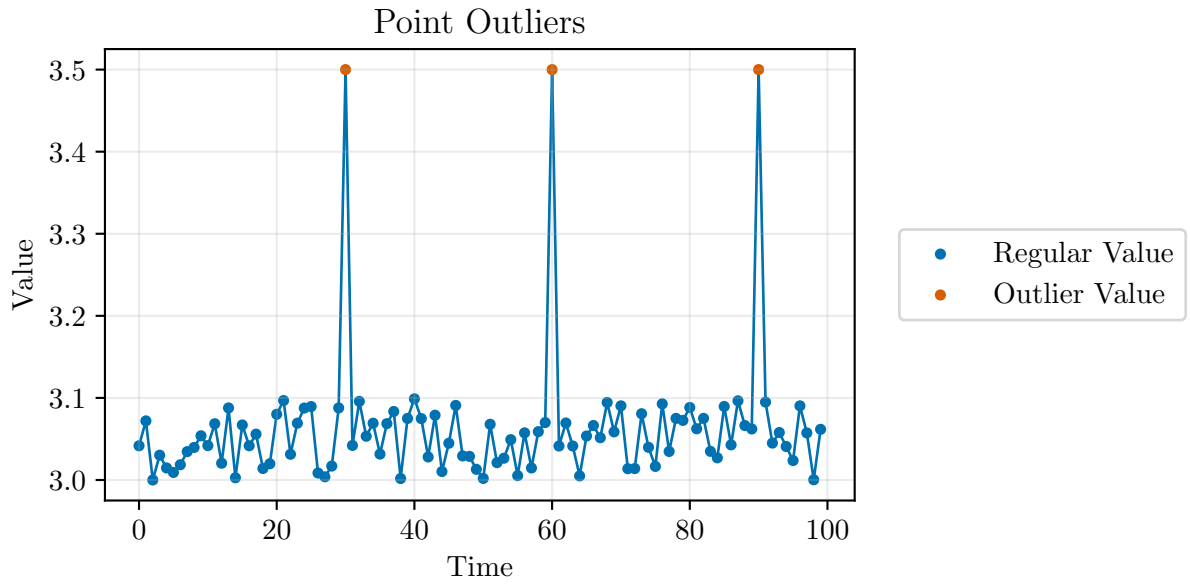


Figure 1: Examples of three point outliers

### Subsequence outliers

Subsequence outliers are multiple consecutive datapoints that strongly vary from the usual trend of the datapoints. [6] In Figure 2 examples of subsequence outliers are shown.

Furthermore outliers can be divided into local and global outliers.

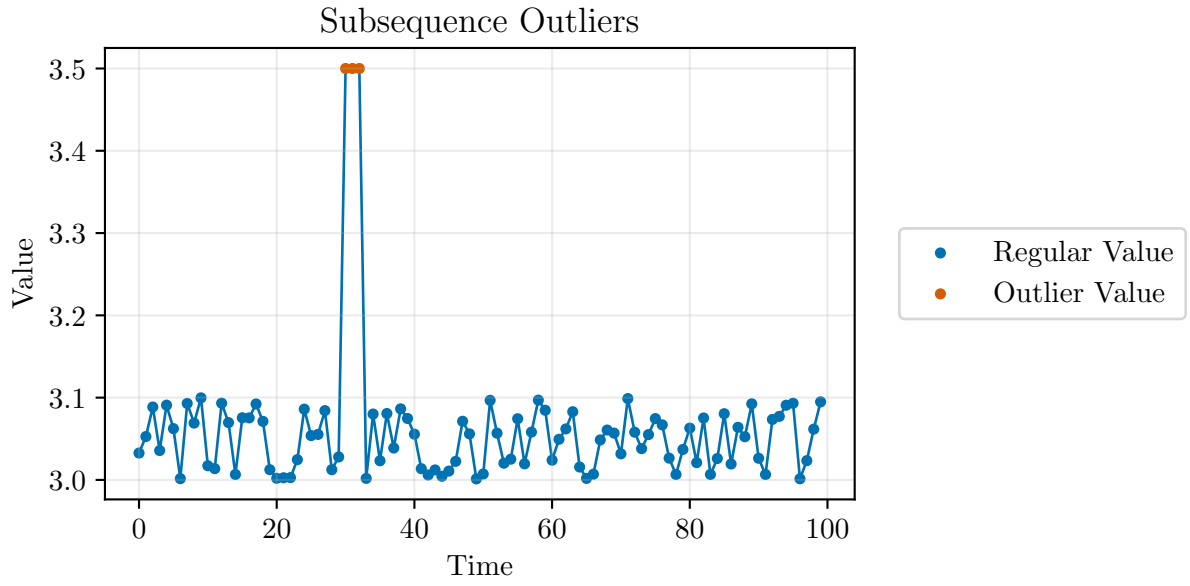


Figure 2: Examples of three subsequence outliers

### Local Outliers

A local outlier has a greater variance to its direct neighbouring datapoints (previous and next one) [6] 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.

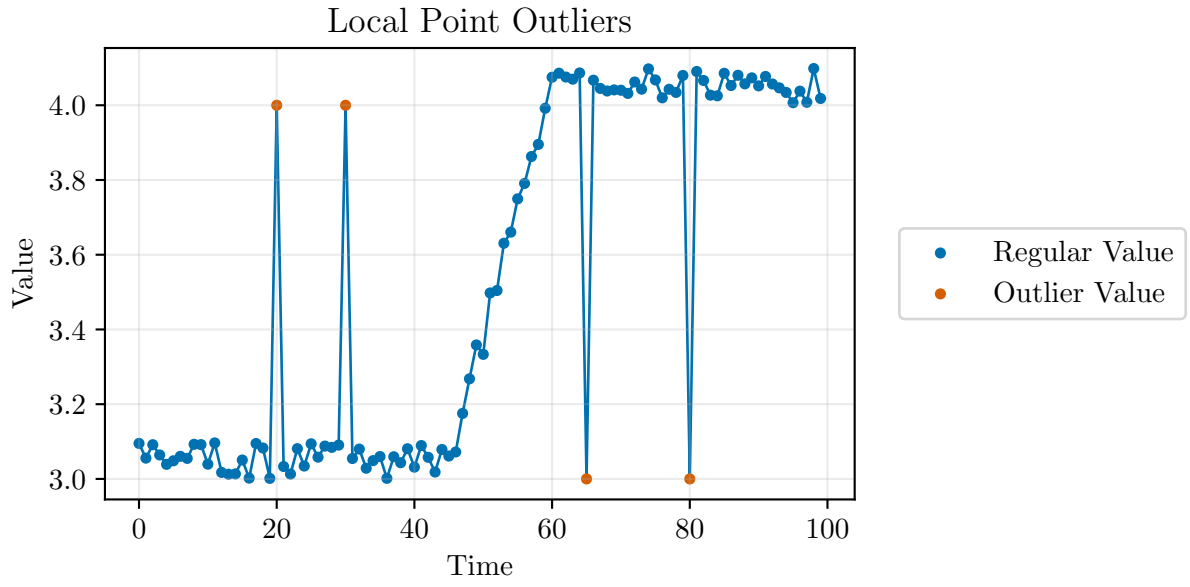


Figure 3: Examples of four local point outliers

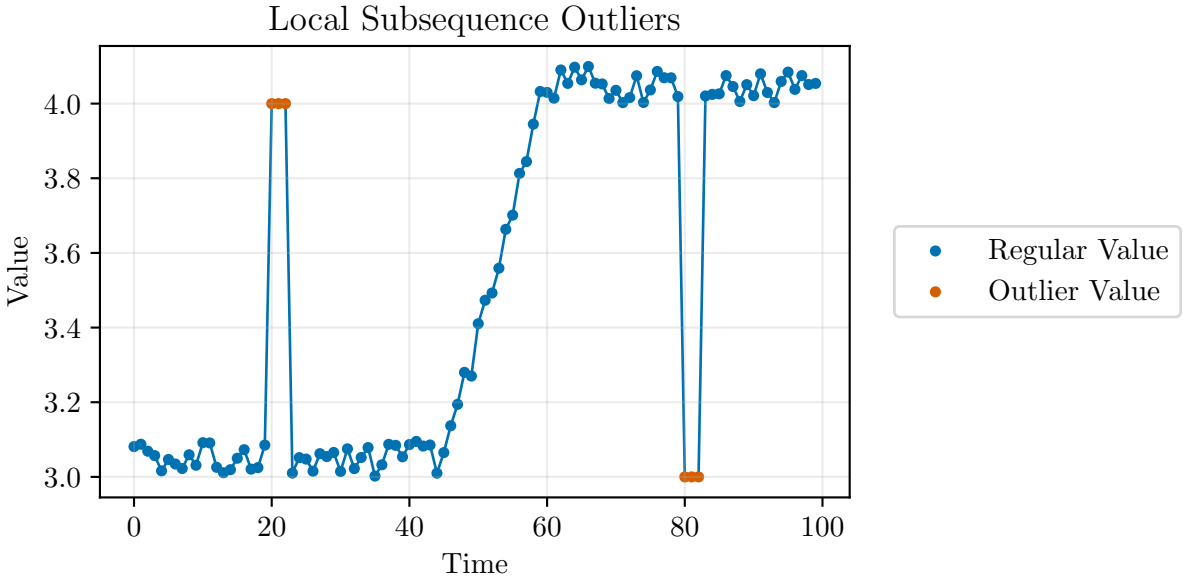


Figure 4: Examples of six local subsequence outliers

### Global Outliers

Whereas a global outlier varies more in regard to all datapoints. Figure 1, 2 and 5 show picture examples of global outliers. [6]

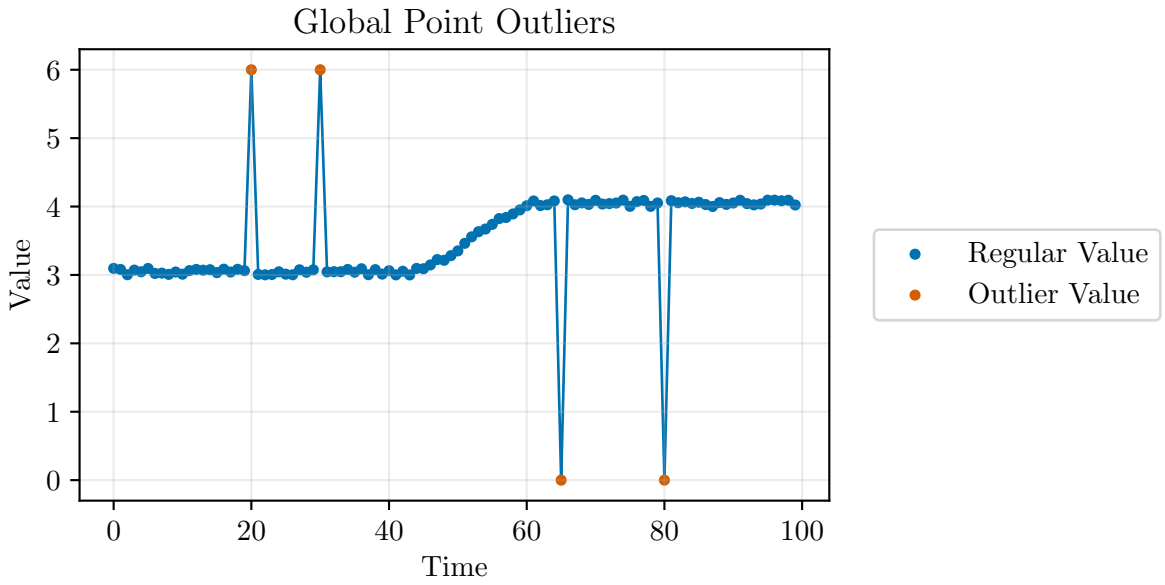


Figure 5: Examples of four global point outliers

## 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 [7, 8]

**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. [7–9]

**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. [7–9]

**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. [7, 8]

**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. [7]

## 3.3 Threshold based Outlier Detection

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

$$|x_t - \hat{x}_t| > \tau \text{ [6]}$$

Where $x_t$ is the actual value and $\hat{x}_t$ is the expected value and $\tau$ 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 $\tau$ needs to be chosen. The greater $\tau$ is the fewer outliers will be detected. The smaller $\tau$ is the more outliers will be identified. [6]

**Mean**

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

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:

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

If $n$ is even:

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

Where $x$ is a dataset of $n$ elements ordered from smallest to largest
($x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_{n-2} \leq x_{n-1} \leq x_n$) [6] 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 (Mean Absolute Deviation (MAD))**

The Median Absolute Deviation

$$MAD = median(|x_t - median(x)|)$$

$MAD$ 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 $MAD$ [10, 11]

## 3.4 Outlier Detection using z-score

write

## 3.5 Outlier Detection using modified z-score

[12]

write

# 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. [13]

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. The code for the Python wrapper is available on GitHub [14]: https://github.com/SOBOS-GmbH/pegelalarm_public_pas_doc [15]

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

```
{
    "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. [16]. 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


```
{
    "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


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

- List of station addresses

add final stations

## 4.4 Explorative Data Analysis

Similar to overview of the data Code: [17]

write

## 4.5 Manually detect outliers for a subset of data

add examples of outliers in the data

Describe how the outliers are stored

Remove this

Show cases of outliers in the data and manually classify them. (Also define a way/data structure to classify outliers for time series data)

In order to speed up the manual labeling of outliers a program was written. The program is a Plotly Dash 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 6 you can see the Website of the Python app. 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.

add reference

## Manual Outlier Selection

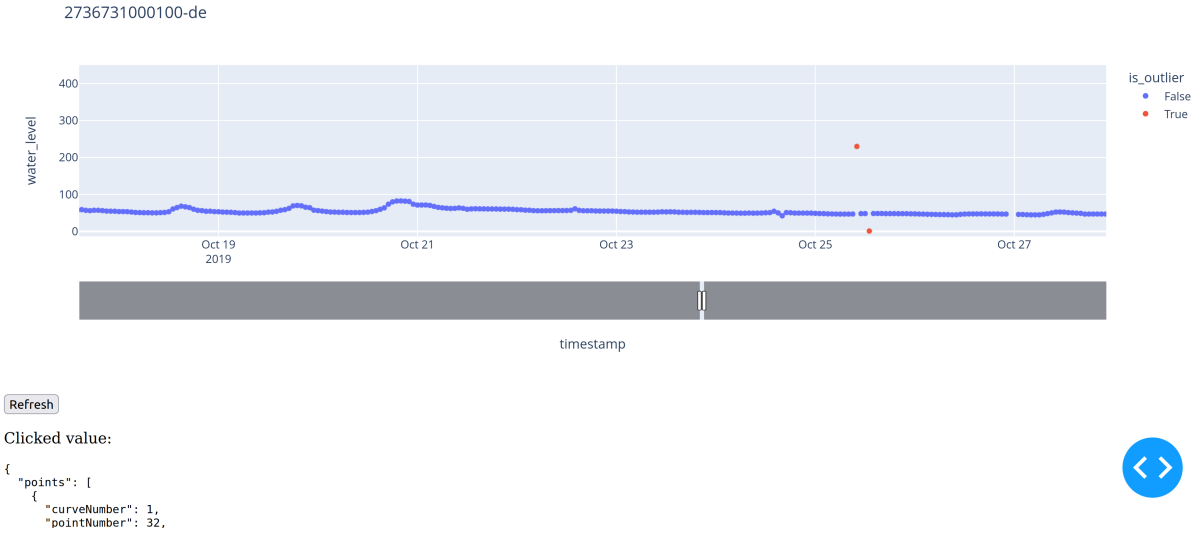


Figure 6: Dash Webapp to classify outliers

### Manual Outlier Detection Webapp using Dash

```
import json

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


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

Listing 4: Manual Outlier Detection Webapp using Dash

add comments to code

## 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 4, 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 4: Example of binary confusion matrix

add reference? https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=1058352752

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:

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

add caption to equations?

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:

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

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

### 4.6.3 Precision and Recall

Figure 7 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.

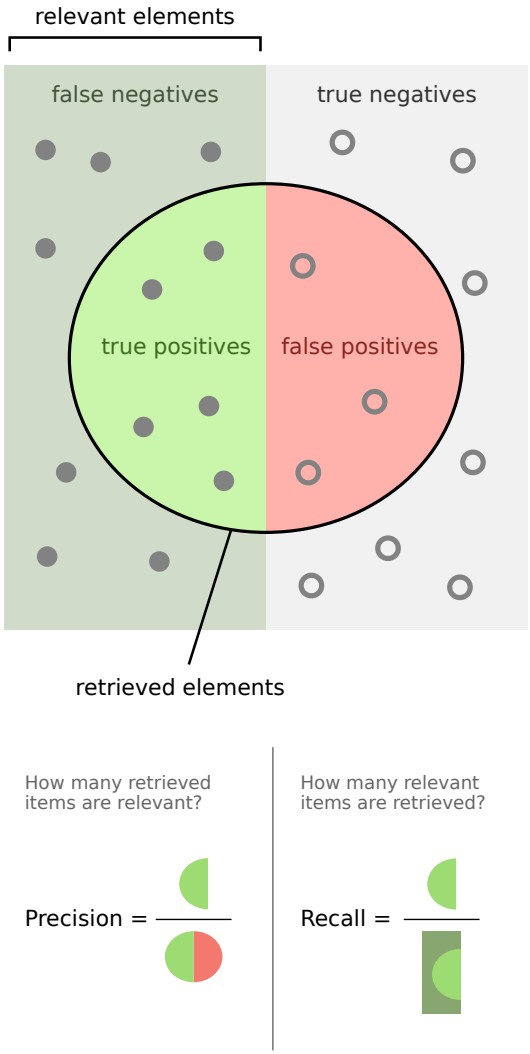


Figure 7: Precision vs Recall [18]

**Precision**

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

**Recall**

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

### 4.6.4 F-score

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

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

[19, 20]
The $F_\beta - score$ is more sophisticated compared to the $F_1 - score$ where an additional parameter ($\beta$) needs to be chosen, which is used to weight the precision.

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

[21] This [21] references [20].
The $F_1 - score$ uses $1$ for $\beta$ so the precision and the recall are equally weighted.

chech whether this can also be included

## 4.7 Implement different Outlier Detection Approaches

Develop different outlier detection methods in Python and calculate performance metrics for each

write

## 4.8 Compare different Outlier Detection Approaches

Compare detection methods from the previous section.

write

# 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.

Change this! Currently copied from the paper.

## 5.1 Advantages and Disadvantages of used Outlier Detection Methods

write

# Bibliography


[1] 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.

[2] 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.

[3] "Apache Parquet," Dec. 2021. [Online]. Available: https://parquet.apache.org/

[4] 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.

[5] J. I. Maletic and A. Marcus, "Data Cleansing: Beyond Integrity Analysis," p. 10, 2000.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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

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

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

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

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

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

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

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

[20] 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

[21] 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

Figure 1 Examples of three point outliers . . . . . 6
Figure 2 Examples of three subsequence outliers . . . . . 7
Figure 3 Examples of four local point outliers . . . . . 7
Figure 4 Examples of six local subsequence outliers . . . . . 8
Figure 5 Examples of four global point outliers . . . . . 8
Figure 6 Dash Webapp to classify outliers . . . . . 17
Figure 7 Precision vs Recall [18] . . . . . 21

# List of Tables

Table 1 Example of IoT sensor data . . . . . 4
Table 2 Example of IoT sensor data after cleaning . . . . . 4
Table 3 Example of IoT sensor data after cleansing . . . . . 5
Table 4 Example of binary confusion matrix . . . . . 20

# List of source codes

1 Request body to get API key . . . . . 13
2 Example response of the list endpoint . . . . . 14
3 Example response of historical water level data for one station . . . . . 15
4 Manual Outlier Detection Webapp using Dash . . . . . 19

# List of Abbreviations

**IoT** Internet of Things

**CSV** Comma Separated Values

**JSON** Java Script Object Notation

**MAD** Mean Absolute Deviation

**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