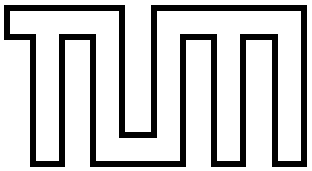


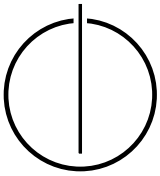
FAKULTÄT FÜR INFORMATIK

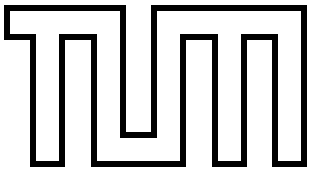
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Cleaning and Repairing Time Series Data

Philipp Sedlmeier





FAKULTÄT FÜR INFORMATIK

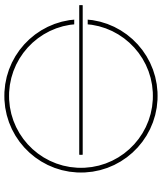
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Cleaning and Repairing Time Series Data

Bereinigung und Reparatur von Zeitreihendaten

Author:	Philipp Sedlmeier
Supervisor:	Prof. Dr. Christian Mendl
Advisors:	Dr. Felix Dietrich, Dr. Xiufeng Liu
Date:	15.05.2022



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Hamburg, 14.05.2022 Philipp Sedlmeier

Abstract

In Denmark, district heating is very common. Denmark's Technical University (DTU) collects sensor data of three of these district heating systems, including measurements of pressure, flow and temperature within those systems. However, due to a variety of reasons such as transmission or sensor failures, data points from these time series measurements are missing. While sensor failures affect a single sensor for a longer period of time, transmission failures may affect multiple sensors at the same time. Such a lack of data points within one or multiple time series can negatively impact downstream applications. Hence, it is advisable to reconstruct these missing data points beforehand.

This thesis addresses the questions resulting from that task. We will investigate and implement approaches to recover such missing data points. To this end, it is also necessary to examine whether there are also anomalies in the existing data, and how we can clean or even correct those. We will see, that there are many more or less obvious anomalies in our data, and that we will need to develop our own algorithm to remove the most obvious ones.

Moreover, the suitable recovery mechanism also depends on whether a real time recovery is necessary, or whether we can perform the recovery on a periodic time scale. In this thesis, we disregard the former and concentrate on the latter part.

We will examine multiple mechanisms to impute missing data. We will see that regular interpolation and regression will not be good choices for most of our data. However, by making some adjustments to standard polynomial regression, we will be able to get comparable results to other state of the art methods. Speaking of which, we will also see that we cannot use matrix imputation or the currently beloved Variational Autoencoder so far, but we will discuss what would be necessary to do so.

Zusammenfassung

In Dänemark wird vor allem mittels Fernwärme geheizt Die Technische Universität Dänemarks (DTU) sammelt Sensordaten von drei dieser Fernwärmesysteme, darunter Druck, Durchfluss und Wassertemperatur in diesen Systemen. Durch eine Vielzahl an Gründen, wie z.B. Übertragungsfehlern oder defekten Sensoren, fehlen jedoch Datenpunkte aus diesen Zeitreihendaten. Während defekte Sensoren einen einzelnen Sensormesswert über einen längeren Zeitraum beeinträchtigen, können Übertragungsfehler auch mehrere Sensoren betreffen. Solch ein Mangel an Datenpunkten über einen längeren Zeitraum kann auch Anwendungen beeinträchtigen, die im Anschluss mit diesen Daten arbeiten möchten. Deshalb ist es ratsam, fehlende Daten bereits im Voraus zu rekonstruieren.

Diese Arbeit beschäftigt sich mit den daraus resultierenden Fragen. Wir werden Vorgehensweisen zur Wiederherstellung solcher fehlenden Datenpunkte untersuchen und implementieren. Dazu ist es zunächst notwendig zu untersuchen, ob die vorhandenen Daten Anomalien enthalten, und wie wir diese gegebenenfalls bereinigen oder korrigieren können. Wir werden sehen, dass unsere Daten allerhand mehr oder weniger offensichtliche Anomalien enthalten, und dass wir unseren eigenen Algorithmus entwickeln müssen, um ausgerechnet die offensichtlichsten davon zu entfernen.

Zudem hängt ein geeigneter Rekonstruktionsmechanismus auch davon ab, ob eine Wiederherstellung der Daten in Echtzeit notwendig oder wenigstens gewünscht ist, oder ob wir die Wiederherstellung auch periodisch und im Nachhinein durchführen können. In dieser Arbeit werden wir uns ausschließlich auf die nachgenannte Variante konzentrieren.

Dabei werden wir mehrere Mechanismen zur Datenwiederherstellung untersuchen. Wir werden sehen, dass die standardmäßige Herangehensweisen bei Interpolation und Regression keine gute Auswahl für unsere Daten sind. Allerdings werden wir ebenfalls sehen, dass nur wenige Änderungen an der polynomiellen Regression nötig sind, um zu anderen modernen Techniken vergleichbare Ergebnisse zu erhalten. Zudem werden wir uns damit beschäftigen, warum andere moderne Techniken wie die Vervollständigung von Matrizen oder Autoencoder in unserem Fall momentan nicht geeignet sind - und was wir dafür tun müssten, damit sie es sind.

Contents

1 Introduction

District heating is very common in Denmark. There exist six large and a few hundred smaller district heating areas, to which almost two thirds of all private households are connected to obtain their space heating and warm water [Dan17]. These heating systems are supervised with a variety of sensors that measure, for example, the pressure, flow and temperature within the pipes. For three of these systems, Denmark's Technical University (DTU) collects time-series sensor data in a data lake.

Due to a variety of reasons, most notably transmission or sensor failures, some data points are missing from these measurements. However, the absence of data points may affect statistical properties of the time series and therefore downstream analysis applications [KATCM20]. Hence, it is advisable to reconstruct these missing data points beforehand.

We will investigate methods to do that in this thesis. As is customary, we will first present some related work in Chapter 2.

Then, we have to examine the data from DTU's data lake that we are working with in detail in Section 3.1. In particular, we have to find out in which cases data is not present "by design", and in which cases data points are actually missing. However, time series may not only suffer from missing data, but also from anomalies in the data that is present. These anomalies also exist in our data, as we will see in Subsection 3.1.4.

Therefore, we will devote ourselves to detecting such anomalies methodically in Section 3.2. We will present multiple state-of-the-art mechanisms for that and evaluate them on our own data first. Then, we suggest our own enhancement to detect outliers in our datasets in Subsection 3.2.6, and evaluate the approach of our choice in Subsection 3.2.7.

Subsequently, we will move on to Section 3.3, where we will discuss what it means to reconstruct missing data points in the first place. We start by interpolating our data with different methods in Subsection 3.3.1 and discuss the resulting problems. We then proceed by examining how we can reconstruct our data with polynomial regression in Subsection 3.3.2, propose some modifications to it and evaluate the results.

In Subsection 3.3.3, we discuss whether and how we could reconstruct multiple time series at once, or how we could make use of one time series to reconstruct another. Before we conclude, we provide some words on the omnipresent Variational Autoencoders in Subsection 3.3.4.

2 State of the Art

When telling a friend about the topic of this thesis, he reacted somewhat surprised that there is still research to do in this area. And indeed, there already exists a vast amount of approaches to and research on dealing with missing or anomalous time series data. In fact, some methods to handle gaps in the available data, e.g. linear interpolation, can be traced back to Ancient Babylon and Greece [Mei02]. Arguably, for Greek statisticians, it went pretty much downhill from there [Sch18].

Nowadays, there exist a large number of approaches to deal with missing time series data - and, with *Momo* by Michael Ende, at least one book that deals with the problem when time itself is missing - or rather stolen. Nevertheless, this field of research is still very vibrant today - with the emergence of the Internet of Things and Big Data maybe more than ever.

Handling Missing Data

Ironically, a very common way to handle missing data is to delete data [Kan13]. But while this will seem at least counterintuitive at first glance, it is actually not inherently unreasonable. Note here, that something being common is not a sufficient reason for it to make sense on its own - just remember that the geocentric astronomic model was once very predominant. I can also remember Stephen Hawking describing how he tried to convince his fellow researchers and colleagues of the Big Bang theory. When he succeeded and it was widely accepted, he started doubting it [HK11].

If we project the approach of deleting data onto our application though, it does not mean deleting data from a time series, but rather removing entire time series - namely those that are missing some data points. Removing those cases is only possible, if it does not introduce biases; this is the case, *as long as the data is missing completely at random.*

In contrast, if it is missing not at random, just discarding the missing data could very well influence the results. Just consider a simple artificial example, where we send out a short survey about their monthly earnings to 2,000 persons that we selected at random from a population. Now, we want to calculate the average monthly income of this population. If 100 persons did not respond, we can probably just discard those survey forms and still get a decent estimate from the remaining 1,900 responses. However, if the surveyed persons had to send their responses back to us by letter, and sending this letter requires the payment of a very high postage, those 100 responses might be missing from the people with the lowest incomes, i.e. not at random. Just discarding the missing data will give us an average income estimate that is (much) too high in this case.

And indeed, we can verify that deleting time series is not an uncommon practice. In a project similar to ours, Johra et al. study smart energy meter data from Danish dwellings that are connected to a district heating network. They try to identify, among others, profiles of daily household energy use. Their data is missing some 3.7% of all measurements, whereof some gaps in the data are longer than 9 hours, which they deem too large to interpolate.

Thus, all time series containing gaps at least this long are discarded.

But this means that with about 3.7% of all measurements missing, they remove over 38% of all their data in the first place. The remaining (shorter) gaps in the data are then imputed using an exponential weighted moving average or linear interpolation, depending on the respective data type [JLH+20].

In another study on household energy use, households with missing data were excluded as well. They also underline the importance of well-reported data on a high level of detail [WLV+09].

Filling Gaps in the Data

If we cannot or do not want to simply remove those time series or data sets that are missing data points from consideration, for example because some data points are missing in all of them - what can we do to reconstruct this missing data? Well, we can already suspect a large number of approaches to the topic by looking at the large number of recent surveys on the topic.

We can find more general surveys, such as the one by Wang and Wang [WW20], who classify data cleaning techniques and tools. We can find even more surveys that focus only on specific aspects. For instance, they focus on a specific class of approaches as Fang and Wang do on Deep Learning methods [FW20]; or they focus on a specific programming language, e.g. R [MBB17]; or on a specific application domain, e.g. climate data [AYMTF20]. An overview on statistical methods to impute missing data is provided by [LR02], a survey of interpolation in time series can be found in [LAC17].

These surveys also differ greatly in their level of detail. We can find many papers and articles that shortly summarize useful techniques for handling missing data [Kan13, Zuc20]. Most of these cover techniques from mean imputation over regression to KNN imputation, and then sprinkle a few more flamboyant ones on top for a personal touch.

Instead of interpolating or imputing the resulting data, another approach is to model the process that creates this data. For example, there exist some models to simulate district heating systems, which are mostly concerned with operational optimization, since the instantaneous heat demand in connected buildings is usually not known [Ben91, PLB+99]. Instead of using a microscopic model, we can also try to find a macroscopic one that is cheaper to evaluate, i.e. a *Surrogate Model*. This, in turn, can potentially be generated from observations generated by the microscopic model [Die17].

As for anything remotely data-related today, we can already suspect Deep Learning approaches to be near - and indeed, they are [CGL+20]. Champion et al. try to discover the underlying (or "governing") equations from the scientific data directly, in this case by using an autoencoder network [CLKB19].

Autoencoders have apparently gained a lot of interest in this field in general. Liguori et al. study three different autoencoder-based neural networks in their attempt to reconstruct missing data from buildings, such as CO_2 concentration, temperature or humidity in a room [LMD+21]. They claim that their method outperforms polynomial interpolation for smaller gaps in the data.

Khayati et al. evaluate multiple concrete matrix- and pattern-based algorithms for the imputation of missing values, using a benchmark [KLTCM20]. As a matter of fact, they find that none of the 12 algorithms under consideration delivers high accuracy in all test cases.

Hence, choosing a "good" algorithm will highly depend on the underlying problem - an algorithm or technique that provides great results on one dataset might completely fail on another one. This is also a well-known and long-established policy guideline in other fields.

In Geoinformatics, data is not missing by sensor failures, but by design. For example, it is simply not possible to obtain meteorological measurements *everywhere*. We can only install weather stations with sensors at a few locations and perform measurements there. For the areas in between these stations, we need to perform some kind of spatial interpolation. However, GIS tools should always include multiple techniques for geospatial interpolation, since the assumptions made by different methods will eventually affect the outcome [MM99]. Similarly, there is also no standard approach to impute missing district heating data, as we can see in in [JLH+20].

Along these lines, Khayati et al. provide some guidance on how to pick an algorithm. Given some distinctive features of the data set of interest as criteria, they suggest which algorithm might outperform the others [KLTCM20].

In another paper, they argue that common recovery techniques are not suitable for time series data that arrives in a streaming fashion. For this, they introduce a new technique based on Centroid Decomposition [KATCM20].

As they also mention, missing values can impair downstream applications, because they can alter statistical properties of the data. Thus, it would not seem far-fetched to make those downstream tasks more robust instead. However, making them more robust is often done by reconstructing the gaps in the data as well, for example see [LKK17, LGG08].

Anomalies

Data is not always completely missing. A data set can also contain anomalies, i.e. "patterns in data that do not conform to expected behavior" [CBK09].

Merely from a philosophical standpoint, anomalies are a more difficult problem than missing data points. A data point that is not there is definitely not there - but a data point that is there may or may not be anomalous. Yet whether it is anomalous depends on what we expect or accept as normal behavior, and what we consequently have to define as an anomaly.

Let's assume that we get reports from all the teaching assistants at our faculty about their tutorials. In our first report, the tutorial apparently was visited by 4 students. For the sake of the argument, let's assume it is the tutorial on Monday morning, 8 a.m. That's why it is our first report and that's also why there are only 4 students.

Of these 4 students, 3 have packed an apple for lunch and 1 student has not. Usually, inspecting the students' lunch is not in a teaching assistant's job description - but this week, it is important for our example; all students know and comply. We will probably accept the behavior of the single, apple-less student as completely normal.

However, over the course of one week, reports from all subsequent tutorials trickle in. Interestingly, in all tutorials all students have packed an apple. So we now harbor the suspicion, that our first (and so far only) apple-less student is either very peculiar, or that the measurement is wrong (meaning, the teaching assistant either forgot to mention the fourth apple in the report or just wasn't able to see it in the student's bag).

But it is also possible that neither of our suspicions is true - the student might have simply forgotten her apple this particular Monday, or has an appointment with a doctor

later and does not want to keep her away. Hence, we can consider the same data point to be anomalous or not, depending on our interpretation of the situation; and from looking at the data alone, we cannot see whether our conclusion is actually correct.

This suggests that outlier detection with purely statistical or unsupervised learning methods do not necessarily yield useful outcomes, since they cannot really perform any meaningful interpretation themselves[1]. But they could also detect some anomalies instead that we didn't even think about ourselves. For example, it might turn out that all students took Elstar apples with them, except two connoisseurs who prefer Geheimrat Dr. Oldenburg. In our example, this is only possible, because our teaching assistants also know the cultivar of each apple and included that in the reports.

Nevertheless, there exist many anomaly or outlier detection techniques [con22, CBK09]. Recent approaches to this field often make use of Deep Learning methods, in particular Autoencoders and Generative Adversarial Networks (GAN) [Ass17][2] and variants thereof [Ass17, AC15, PS18]. As it is the case with data imputation, anomaly detection on streaming data requires different approaches [DF13].

Caveats

We should note, that in the vast field of research on time series data, we better take everything with a grain of salt. Many practices that are often examined and frequently used are later shown to be flawed, to say the least. For example, Keogh et al. showed that the frequently used mechanism of clustering streaming time series or time series subsequences is completely "meaningless" [KLT04], i.e. its output is independent from the input.

We have also already discussed some difficulties of the general idea of outlier detection, which is done in much more detail by [CZS+16]. Moreover, a recent paper by Kim et al. suggests that most methods for time series anomaly detection largely overestimate their performance. While their choice of words is less brutal than Keogh's, the paper's authors still claim that current anomaly detection techniques do not perform better than the technique of guessing [KCC+22]. Or, verbatim: "even a random anomaly score can easily turn into a state-of-the-art method" [KCC+22].

In addition, considering how accurate the reconstruction of missing data points should or have to be is also important. Of course, we can build Deep Learning networks with millions of nodes to train them on thousands of time series. However, if our time series contains daily temperature measurements and is missing only two datapoints, $temp(today) = temp(yesterday)$ might also cut it perfectly. The results may not be very good, but often good enough. It always depends on the particular application; but in general, we should apply Occam's razor wherever possible.

[1] For some general ideas and insightful takes on this, [Sha19] is an interesting read.
[2] `https://keras.io/examples/timeseries/timeseries_anomaly_detection/`

3 Cleaning and Repairing Time Series Data

3.1 Available Data

At the core of our studies is a data lake that contains time series data from district heating networks[1]. These time series are provided in three separate data sets - Brønderslev, Hillerød and Trefor. While these three data sets all contain sensor measurements from district heating networks, they are structured very differently. In this chapter, we will look at them in more detail.

3.1.1 Measurement Types

While the data sets are overall very different, the main measurement types are very similar for all three data sets. Each set contains multiple measurements, i.e. also multiple sensors, for the supply and return water temperatures, as well as the flow through the respective water pipe. Also contained in each dataset are some power measurements, although the labeling and scaling of the respective measurements differs between the three sources[2], and some data of the outside weather conditions. The data sets from Brønderslev and Hillerød also contain multiple pressure measurements.

Unique to the Brønderslev data are further measurements from the production units, e.g. motors and boilers[3] and data of the setpoints. Unfortunately, unique to it is also some level of chaos. For example, one measurement is described as a return pressure setpoint; however, label and ID identify it is an actual measurement. In general, the naming of basically anything does not follow a common convention, so it is challenging to find out which measurement is supposed to measure what in the first place.

An overview about the available measurement types can be found in Table 3.1.

We can also state already that even time series with the same measurement type from the same data set can vary greatly in their values. For example, we can have a look at all supply temperature measurements from the Hillerød dataset during the month of December 2021 in Figure 3.1. We would expect them to differ in temperature levels to some extent, due to some temperature loss the further away from the heating plant we are measuring. However, they do not only that, but they also greatly differ in their underlying patterns.

We have highlighted three time series in Figure 3.1b in particular. The `KRA14_TF` time series oscillates around a constant value. The center of these oscillations remains the same during the whole month. The `BKV_TF2` and `MELOESE_BY_TF` time series also oscillate around a constant center. But this center changes at least once at some point during the month, while the amplitude of the oscillations are much smaller. In addition, over the course of

[1] `https://orbit.dtu.dk/en/projects/heat-40`

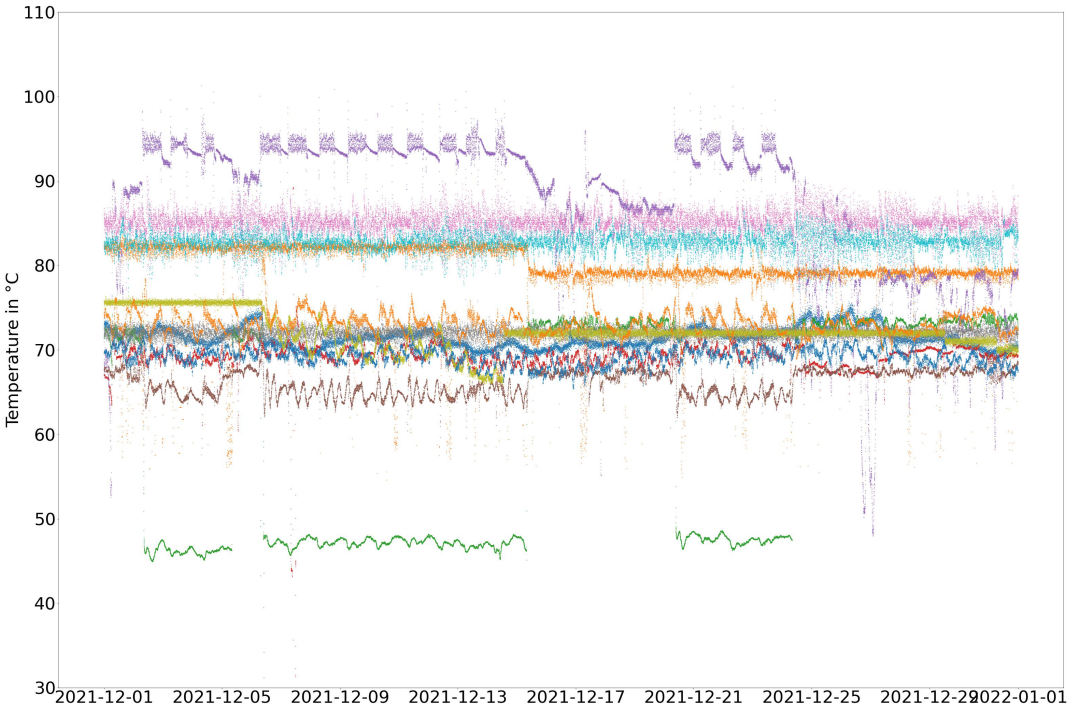
[2] Brønderslev: *Effekt* or *Energi* in MW; Hillerød: *Production* in W; Trefor: *Varmeeffekt* in kW. In addition, the Brønderslev data also includes measurements labelled *Varmeproduktion*, but measured in MW h.

[3] Curiously, now labeled *Varmeeffekt* here

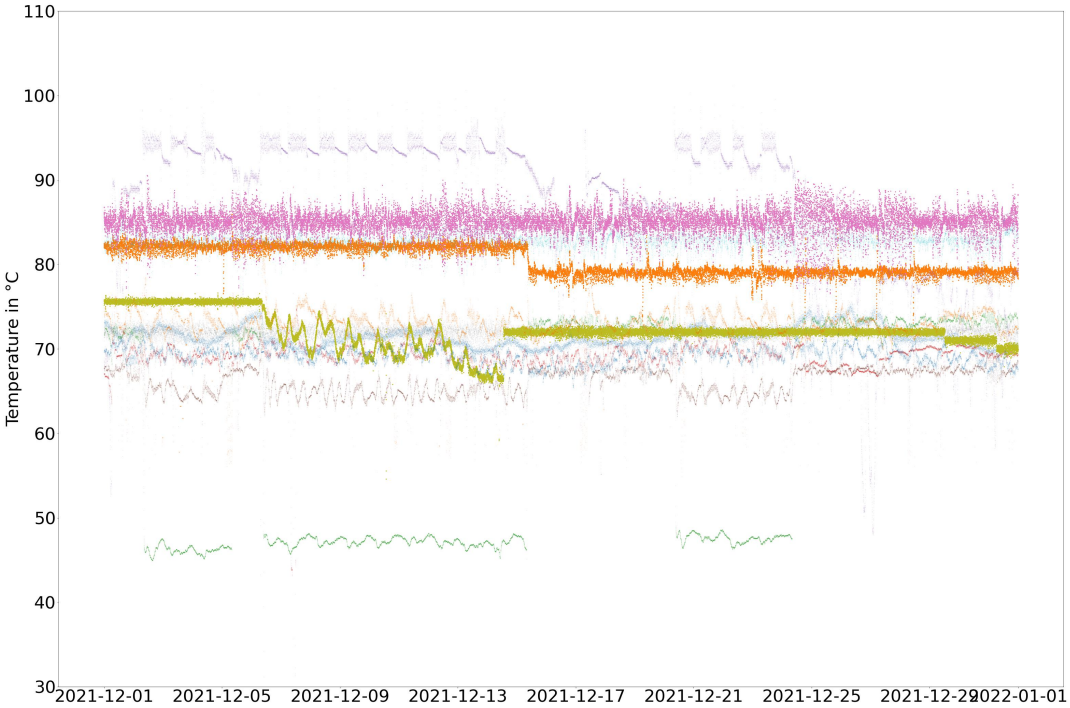
Table 3.1: Type and number of measurements provided by each data set.

Measurement Type	Brøndersley	Hillerød	Trefor
Supply temperature in °C	5	9	11
Return temperature in °C	3	12	5
Flow in $\frac{m^3}{h}$ or $\frac{kg}{s}$	6	8	7
Power in W or MW	4[1]	11[2]	5
Pressure in bar or Pa	7[3]	2	-
Outdoor temperature in °C	2	1	1
Wind speed in $\frac{m}{s}$	-	1	-
Wind direction	-	1	-
Pump speed in %	6	-	-

[1] Plus multiple measurements from the production units.
[2] The Hillerød data contains 7 measurements with three time series each: for *production from transmission*, *production local (gas)* and *total production* (the sum of the previous ones). Then, there are an additional 4 measurements labeled *Heat Production*.
[3] Plus some dubious ones.



(a) We can see different temperature levels and patterns in the different time series.



(b) The `KRA14_TF` (pink), `BKV_TF2` (orange) and `MELOESE_BY_TF` (green) measurements are highlighted in particular.

Figure 3.1: All supply temperature measurements from the Hillerød data during the month of December 2021.

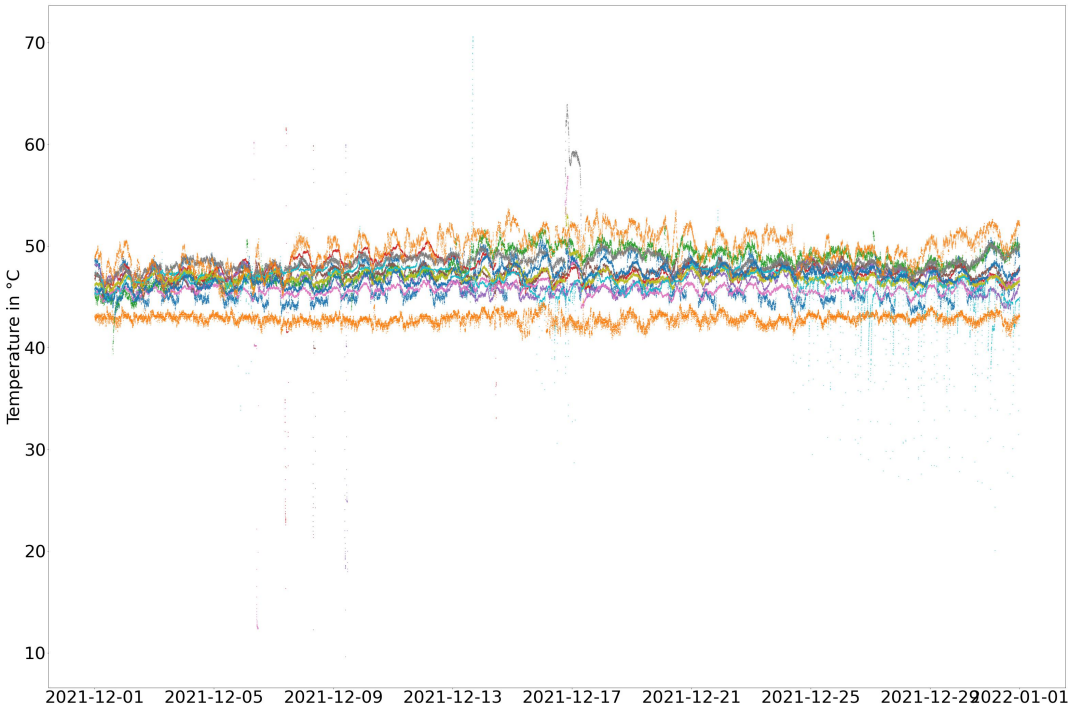


Figure 3.2: All return temperature measurements from the Hillerød data during the month of December 2021. We can see there is much less variance than in the supply temperatures.

approximately one week, the `MELOESE_BY_TF` series does not oscillate around a constant value at all, but exhibits a - for lack of a better word - mountainous regionin between. At the end of the mountain range, there is a very sudden return to a new temperature level.

In contrast, the differences between the return temperatures from the Hillerød data set are much less significant, as we can see in Figure 3.2.

We should already mention here, that all the measurements we have mentioned in Table 3.1 are those that are available *generally*. That does not mean, that all these measurements are available *regularly*. For example, all time series from the Brønderslev data set exhibit an almost two month long gap in the data between August and September 2021 because of transmission errors. Other measurements, i.e. time series, are very sparse with only a few data points over the whole time period, or sometimes do not contain any data at all.

3.1.2 Measurement Period

What is the whole time period anyway? We can find the first available dates from each data set with simple SQL queries[4] that give us different starting dates for each set. The earliest time stamp of the Hillerød data is March 14$^{\text{th}}$, 2020 at 07:32:32. For the Brønderslev data, we can find data as early as September 1$^{\text{st}}$, 2019[5]. Except for three outliers, the time series

[4]For example, `SELECT MIN(t) FROM hillerod.readings`

[5]Well, we can also find some data points from January 1601, but we feel confident in ignoring them.

from the Trefor data start at June 2nd, 2020.

Regarding the last available dates, all data sets provide current data with a short delay.

3.1.3 Time Distances Between Measurements

Our goal is to repair missing time series data. But to know whether data is missing in a time series, we need to know the regular distances between two measurements in the first place. To this end, we examine the data sets more closely and can find, that these distances differ both between and within the three datasets.

For all available time series, we retrieve the data for the first quarter of 2022. For each of these time series, we can then simply calculate the time differences between two consecutive measurements - rounded to a full second, disregarding petty differences - and look for a pattern. We can use Panda's Date and Time functionality[6] for this as in Listing 3.1. Nevertheless, this still can be tricky sometimes. It took us an outrageously long amount of time to figure out that `pd.Series(X, dtype='datetime64[s]')` is **not** the same as `pd.Series(X.astype('datetime64[s]'))`. Don't try this at home, kids!

Listing 3.1: Calculating the distances between consecutive timestamps

```
pandas.Series(
        data.timestamp[1:] - data.timestamp[:-1]
).dt.round(freq='S')
```

The results of this approach are depicted in Figure 3.3, with the data from Bronderslev in blue, from Hillerød in orange and from Trefor in green. In all plots, the x-axis represents the number of seconds that lie between two consecutive measurements; the y-axis represents the share of measurements in a dataset. Thus, all plots show the share of consecutive measurements in each dataset that lie a specific number of seconds apart from each other.

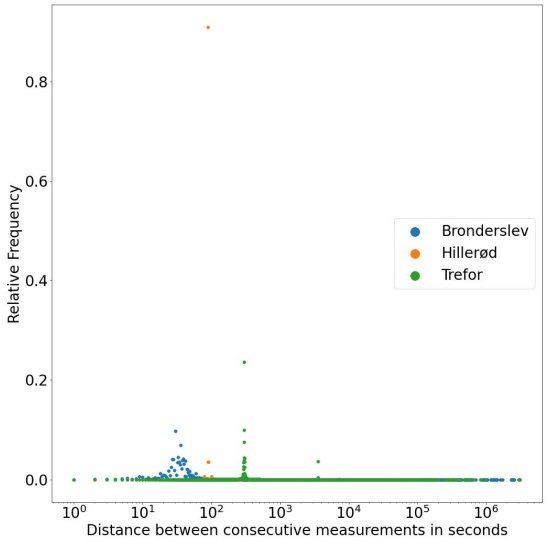
In Figure 3.3a we can see that those distances can actually be quite large. Both in the Brønderslev and Trefor data, we can find measurements that lie more than 10^6 seconds, i.e. more than 11 days apart from each other[7]. However, we can also see that these long distances are relatively scarce. Figure 3.3b shows only those distances that occur between more than 0.5% of the measurements. We can see immediately that the scale of the x-axis is much shorter in this case. And indeed, we can verify that the largest distances with a relative frequency of more than 0.5% are 60, 101 and 3,600 seconds for the three datasets, respectively.

We can also see much better here that the distances in the Bronderslev data are distributed quite broadly, with a bell-shaped curve centered around a distance of 36 seconds. In contrast, the measurements from the Hillerød data lie 90 seconds apart from each other in almost 97.5% of the data. Most measurements from the Trefor dataset are 300 seconds, i.e. 5 minutes, apart, although there is more variance in the exact lengths of the gap.

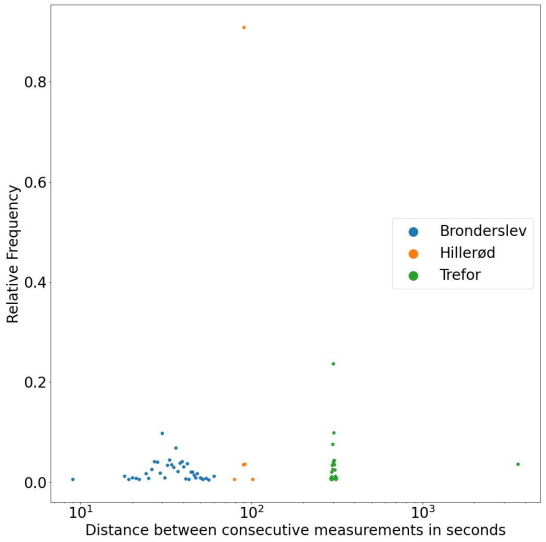
The contrast between the three datasets becomes clearer in Figure 3.3c. It depicts the cumulative share of measurements that is *at most* a given number of seconds apart from each other. The graph for the Hillerød data increases from 0 to 100 virtually in an instant,

[6] `https://pandas.pydata.org/docs/user_guide/timeseries.html`

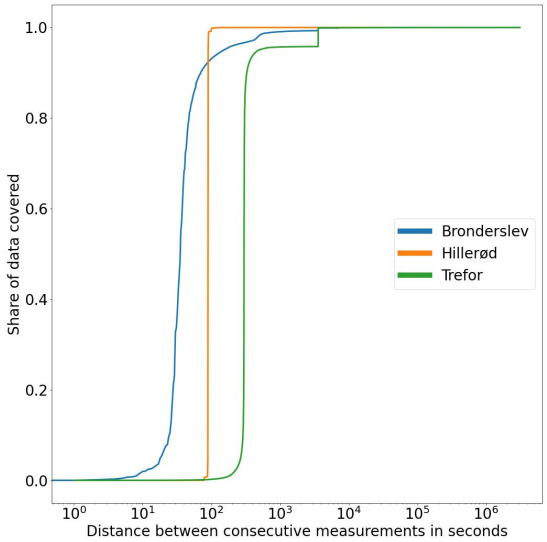
[7] In fact, the largest distances we can find are 36 days 00:28:07s for the Bronderslev and 35 days 08:01:11s for the Trefor data. The Hillerød data lacks these extreme outliers, with the largest distance between two consecutive measurements amounting to 0 days 10:35:25s.



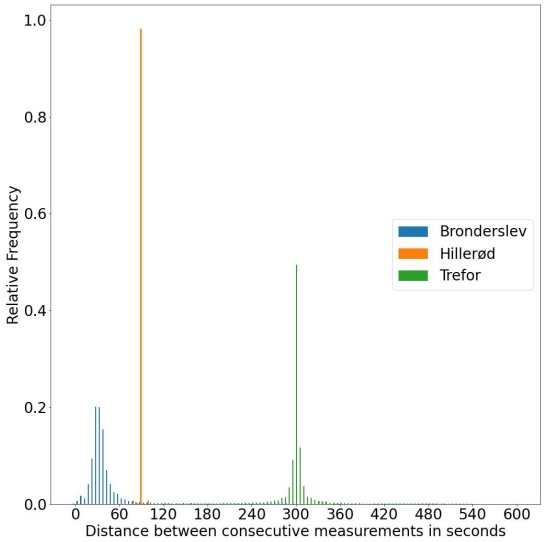
(a) Relative frequencies of time differences between consecutive measurements in seconds, full range of available data.



(b) Relative frequencies of time differences between consecutive measurements in seconds, where the relative frequency is larger than 0.5%.



(c) Cumulated share of consecutive measurements that are at most a given number of seconds apart from each other.



(d) Relative frequencies of time differences between consecutive measurements in seconds that are at most 10 minutes apart from each other, aggregated by 5-second increments.

Figure 3.3: Time differences between two consecutive measurements in each of the three data sets during the first quarter of 2022.

Table 3.2: Key statistics on the distances between consecutive measurements for each dataset.

	Brønderslev	Hillerød	Trefor
Mean Distance	0 days 00:01:45.97s	0 days 00:01:30.24s	0 days 00:09:02.60s
Standard Deviation	0 days 01:25:02.88s	0 days 00:01:15.51s	0 days 03:27:00.86s
Median Distance	0 days 00:00:36s	0 days 00:01:30s	0 days 00:05:00s

while the slope of the Bronderslev curve is much less steep; the slope of the Trefor curve lies somewhere between those extremes. We can also see here, that after more than 5 minutes, still not all of the Trefor data is covered. Instead, we can find that about 4% of all measurements are *exactly* 3,600 seconds (i.e. an hour) apart from each other.

Only about a quarter of the Trefor data is *exactly* 300 seconds apart - many measurements are off by a couple of seconds. Nevertheless, we can see in Figure 3.3d, which is aggregated to multiples of 5 seconds, that the data is distributed almost symmetrically around the mode. By this we mean that there are about as many consecutive measurements 270 seconds apart from each other as there are measurements that are 330 seconds apart from each other.

To put it more precisely, let $freq_x$ denote the relative frequency of distances between two consecutive measurements with a length of x seconds. If we calculate $\forall x \in [1, 50] \subset \mathbb{N}$: $freq_{300-x} - freq_{300+x}$, i.e. the difference between the bars that are equidistant left and right from the mode of 300, we find that the largest difference is about 2.34%, with an average over all values of only -0.11%.

To be thorough, we can calculate some key values by hand[8]. The mean distances in each dataset can be found in Table 3.2. But since we have seen in Figure 3.3 that there are some outliers very far away from the main bulk of the data, the mean is not really meaningful here anyway. Table 3.2 also shows us that the standard deviations for the Brønderslev and Trefor data are quite large. In fact, the Trefor data actually boasts the largest standard deviation here, although this amount of dispersion was not immediate from Figure 3.3b. A closer look then shows us that only 1.26% of the Brønderslev measurements are more than 10 minutes apart from each other, while this is the case for 4.6% of the Trefor measurements (and none of the Hillerød data, by the way) - including those 3.63% that are exactly an hour apart.

Thus, we should consider the median distances instead. This way, we get results that support our first intuitions from Figure 3.3. Combining that with Table 3.2, we can safely say that the Hillerød measurements are performed every 90 seconds; if two measurements lie more than 90 seconds apart from each other, there is most certainly something missing.

We can also say that the Trefor measurements are performed every 5 minutes in general, although we have to allow for more leeway here. Nevertheless, over three quarters of consecutive measurements are x seconds apart from each other, if $x \in [290, 310]$. If we allow for $x \in [270, 330]$, the share of covered measurements increases to slightly over 85%.

For the Brønderslev data, it is virtually impossible to say whether single data points are missing or not. For example, plenty of time series in it (but not all of them) contain measurements every second over some length of time. If the secondly data points switch to a 5-second-rhythm - does that mean there are 3 measurements missing over every 5-second period now? Thus, we punt on this question for now and take up this thread later in Section 3.3.

[8] Well, with Pandas or NumPy code, but we type that with our hands.

Table 3.3: Common Anomaly Types in the Data sets

Anomaly Type	Example
Measurements cut off below or above some value	Figure 3.4
Outliers	Figure 3.5
Fluctuations	Figure 3.6

3.1.4 Data Anomalies

Not only are data points missing, there are anomalies in the time series as well. Some anomalies are easy to detect and explain, for others this is not necessarily the case. In this section, we summarize the most common anomalies in Table 3.3 and provide some examples.

Cutoffs

In multiple time series, we can detect measurements that are cut off when they reach a specific threshold. This can manifest itself in different flavors, which we can see in Figure 3.4. Figure 3.4a depicts the flow measurement `GOERLOESE_BY_FLOW` from the Hillerød data over the course of one year, from June 2020 until May 2021. The graph suggests that the measurements are cut whenever they reach a value higher than 50 $\frac{\text{kg}}{\text{s}}$ - during the winter months, we would expect the wave-shaped distribution to assume values above that threshold as well. And indeed, when we look at the same measurement during December 2021 in Figure 3.4b, we can see that the data is still capped at the value of 50 until December 6^{th} around noon; afterwards, this threshold is suddenly surpassed.

Unfortunately, we cannot find a clear pattern which time series are affected by this anomaly and which aren't. Other flow measurements from the same data set do not show this behavior, as we can see in the `MELOESE_BY_FLOW` measurement over the same period of time in Figure 3.4c.

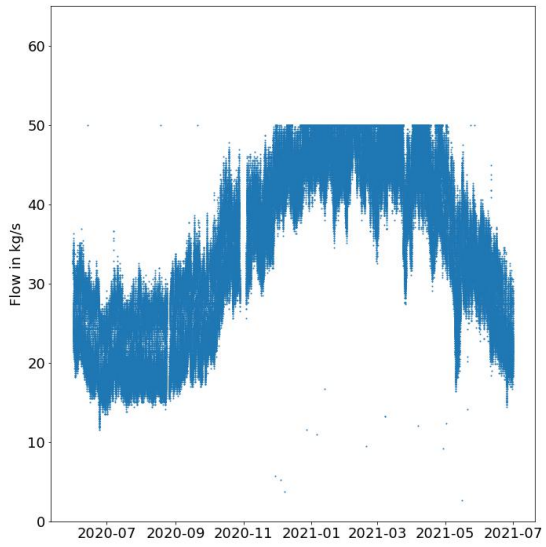
Note, that in the aforementioned case, the values are not missing - they are still contained in the data set and all of them are set to the value of 50. This will make it harder to detect this anomaly as such mainly for two reasons:

1. The value itself is a perfectly reasonable one, i.e. it is possible to measure this value. Thus, we do not necessarily know, which measurement of 50 is correct and which one is not.
2. There is variance in the data. That means, we do not have a long string of the same measured value that we could discard, there are actual measurements in between.

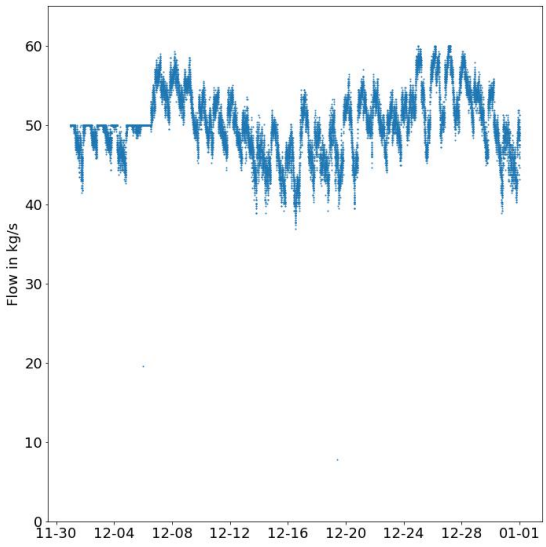
Moreover, while these measurements are definitely anomalous, they are not outliers. In our example, 4.4% of the data are measurements of $50\frac{\text{kg}}{\text{s}}$, which make it the most frequent measurement. But from this fact alone, we could not be sure that these measurements are anomalous[9].

There are also time series, where the anomaly is more obvious. Figure 3.4d shows the measurements of the outdoor temperature `VS_FRE__T_AMB` over the same time period,

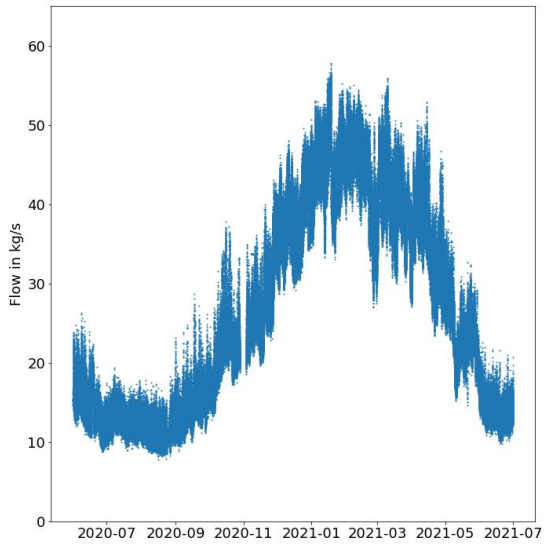
[9]The next most frequent measurement, rounded to a single decimal, is the value 46.1, which comes up about 1.42% of the time out of 142 different values overall.



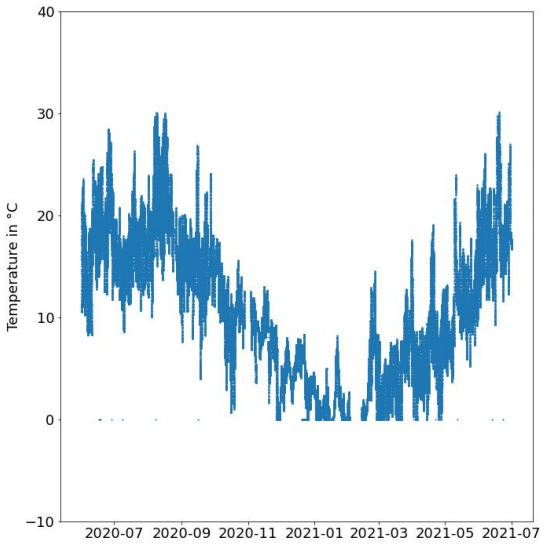
(a) Measurement of a flow in $\frac{\text{kg}}{\text{s}}$ from the Hillerød data set (`GOERLOESE_BY_FLOW`). The value of 50 is reached often, but never surpassed.



(b) The `GOERLOESE_BY_FLOW` measurement during December 2021. The cutoff at the value 50 ends halfway during the day of December 6^{th}.



(c) `MELOESE_BY_FLOW` measurement in $\frac{\text{kg}}{\text{s}}$ from the Hillerød data set. Here, the data is not cut off



(d) Outdoor temperature measurements in °C. Temperature measurements do not go below 0°C.

Figure 3.4: Measurements do not assume values above or below a certain threshold.

also from the Hillerød data. This time series does not contain any negative temperature measurements, which is unusual for Denmark over the course of a whole year.

These measurements are not completely missing from the time series, but appear in the data set as outliers with a value of either 1802.59998 or 6553.5. Since we are measuring the temperature in Denmark, not on the surface of the sun, we can easily detect and remove these values. Apart from that, we can assume the remaining (feasible) values to be correct for now.

Outliers

Our data contains outliers in a broader fashion as well, as we can see in Figure 3.5. For example, Figure 3.5a shows the measurement `FRE_FLOW` during December 2021. Most of the data is displayed as a straight line at the bottom of the graph, with values close to 0. If we look more closely, however, we can find that the graph contains some data points at the top as well, with values close to 10^{10}; for better visibility, we have encircled them in red.

If we remove those values from the data, we end up with the graph in Figure 3.5b. Unlike in the previous examples, where these outliers were included in the data instead of some measurements that fell below a certain threshold, we cannot find a specific pattern in this case. Nevertheless, the outliers are still very easily identifiable as such and we can safely assume those data points to be "wrong" or faulty data.

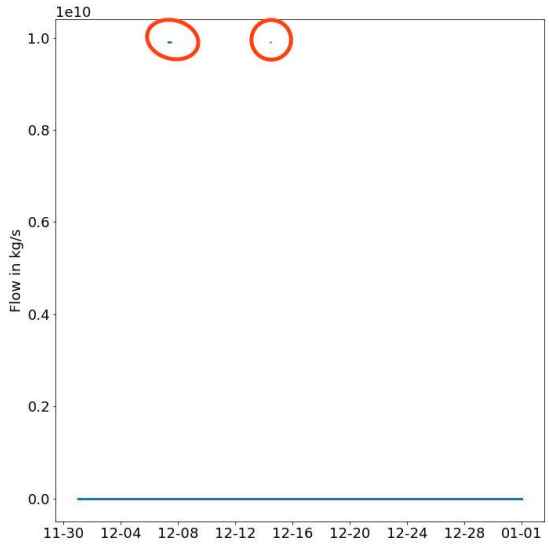
Unfortunately, this is also not always the case. For example, we can look at the `GOERLOESE_BY_FLOW` measurement again in Figure 3.5c. In the red circles, we find multiple data points that are clearly distinguishable from the bulk of the data. However, while these data points do not fit into the wave-shaped band distribution of the data, the values still fall into a reasonable and feasible domain. In other words, these outliers could be faulty measurements, but they could also be correct measurements of some underlying anomaly.

Which of those two options they are depends also on the kind of measurement, since these outliers can be found across all kind of time series in the data sets. Figure 3.5d shows another supply temperature measurement, again with some possible outliers in the bottom right corner. In both Figure 3.5c and Figure 3.5d, it is extremely doubtful or even physically impossible that the flow or temperature in a water pipe drops that drastically and rises back again just in a few minutes. But this is exactly what the graphs suggest, since these measurements are virtually singled out. However, the outlying flow measurements are still feasible in general, since there are similarly low measurements from July to September. Temperature measurements of roughly 10°C in a supply pipe of a district heating network seem highly improbable in any case.

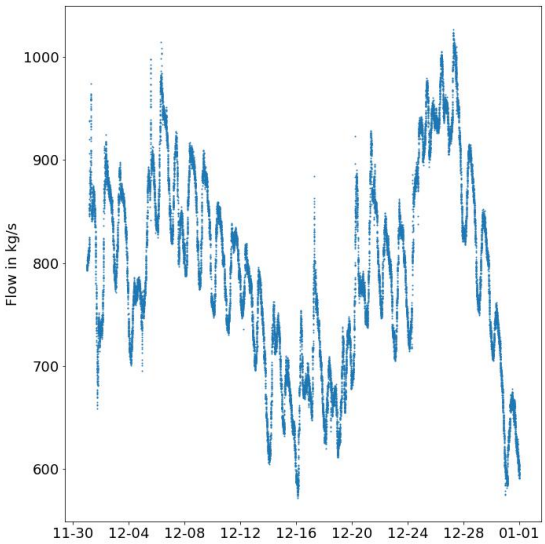
Fluctuations

Most time series exhibit some pattern of fluctuations. Again, they manifest themselves in different flavors and may or may not be correct measurements. Some amount of fluctuations, or rather oscillations, is normal. If the measurements in each time series would follow an easy continuous function or would even be constant, there would not be any struggle to recover missing data in the first place. Thus, we are only concerned with fluctuations that are, for example, much larger in scale or (dis-)appear very suddenly.

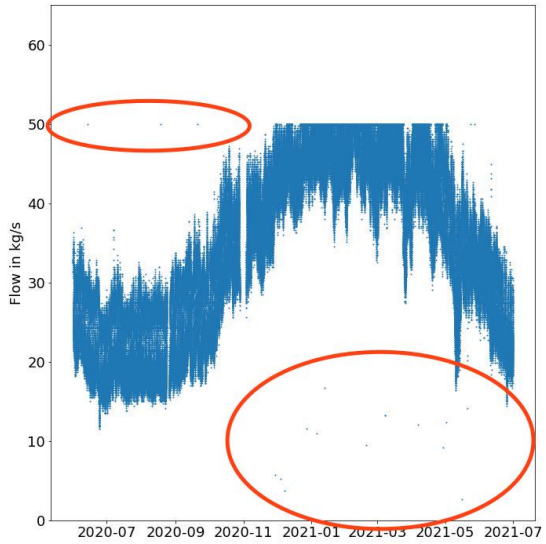
We can differentiate between two main types of anomalous fluctuations. The one type are



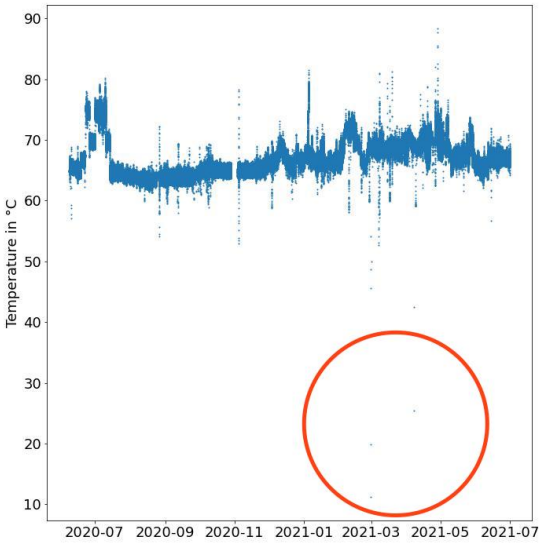
(a) Flow measurement `FRE_FLOW` during December 2021. The outliers are encircled in red. Note the scale on the y-axis, that reaches 1e10 here.



(b) The same flow measurement `FRE_FLOW` during December 2021. The outliers are removed from the data. Note the scale on the y-axis, it only reaches 1,000 here.

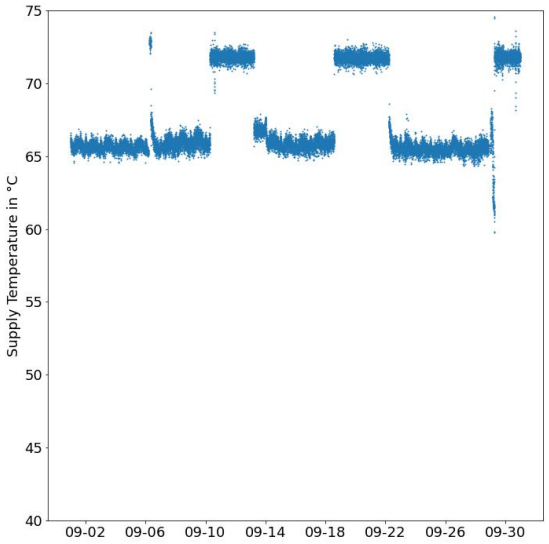


(c) The `GOERLOESE_BY_FLOW` measurement over the course of 1 year. We can find outliers in the bottom right of this graph, here encircled in red. Since these values are at least possible, it is not immediate whether these outliers are wrong measurements or not.

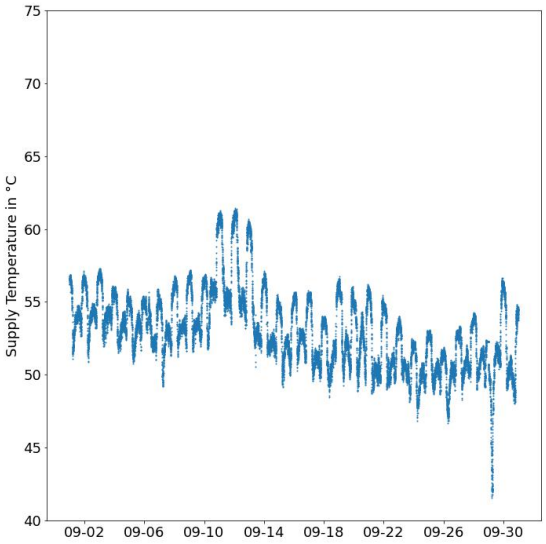


(d) The `FRE_TF` measurement over the course of 1 year. We can find outliers in the bottom right of this graph, here encircled in red. Since these values are at least possible, it is not immediate whether these outliers are wrong measurements or not.

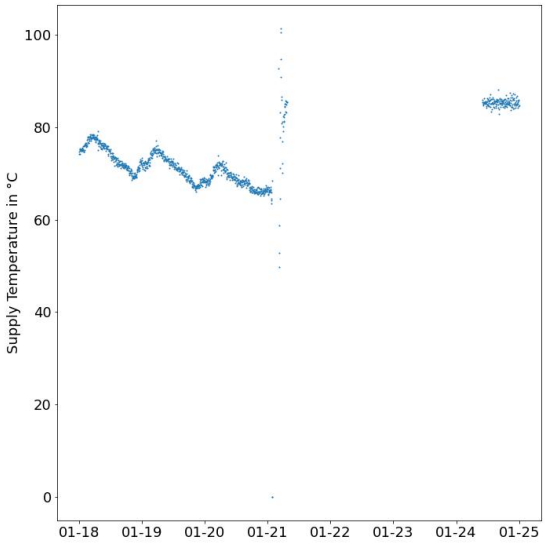
Figure 3.5: Time series can contain outliers. Some are easy to detect, but others are not clearly identifiable as wrong data.



(a) Measurement of the supply temperature `GOERLOESE_BY_TF` during September 2021. The temperature level changes by several degrees Celsius and back again quite suddenly, with few to no measurements in between.



(b) Measurement of the return temperature `GOERLOESE_BY_TR` during September 2021. The fluctuations that we can see in the corresponding supply temperature in Figure 3.6a are not reflected here.



(c) Supply temperature `60SYDOCT50` from the Trefor data during one week in January 2021. We can see sudden large fluctuations in the measurements, including 2 measurements of 0°C and 2 over 100°C, before the measurements are missing completely.

Figure 3.6: Some time series display fluctuations, either of the values themselves or of the value levels.

measurements that usually oscillate around a certain value, but the center of oscillations suddenly changes. The supply temperature measurement in Figure 3.6a is an example of this. In this case, the sudden changes strike us as odd. For one thing, these changes are not reflected in the corresponding return temperature in Figure 3.6b. For another thing, there are no or at most very few measurements between the high and low "bands".

We cannot calculate the exact amount of time the process of cooling down would need here. At least, we would need more information about the the nature and temperature of the surroundings, according to Newton's law of cooling, which is not the case. But from our own daily experience, we would expect the heating and especially the cooling of the water to take some time - similar to what we can see for the return temperature.

The other type are anomalous fluctuations in the values altogether. One example is displayed in Figure 3.6c, where the supply temperature measurements suddenly change very rapidly, before they are missing completely. In particular, they also contain outliers, as there are measurements of 0 and over 100°C in a very short time span.

3.1.5 Additional Data Sources

As we have seen in Subsection 3.1.4, our own data sources are far from perfect. Given the anomalies present in them, we tried to examine similar data from other sources as well. Finding such data proved to be more difficult than expected.

The International Energy Agency (IEA) provides annual data on energy and heat supply and consumption for multiple countries. However, a subscription to retrieve the data is necessary which is subject to a charge[10].

Similarly, the Danish Energy Agency provides annual and monthly statistics, including district heating data, but free of charge[11]; so does the Swedish Energy Agency[12]. But since it is monthly, it is unfortunately not useful for our purpose.

We can also find more detailed time series that are not necessarily related to our application, but include sensor data as well[13]. However, those time series include only up to a few thousand measurements in total. In comparison, most temperature measurements from the Hillerød data contain over 25,000 measurements per month.

[10] https://www.iea.org/data-and-statistics/data-product/electricity-information
[11] https://ens.dk/en/our-services/statistics-data-key-figures-and-energy-maps
[12] https://www.energimyndigheten.se/en/facts-and-figures/statistics/
[13] http://www.timeseriesclassification.com/dataset.php

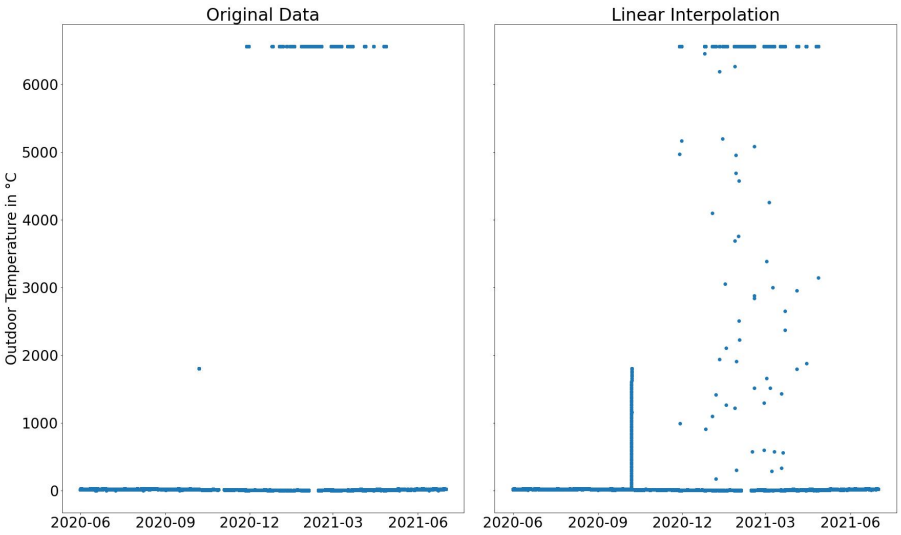


Figure 3.7: Original outdoor temperature measurements (left) with outliers, and linear interpolation to shorter intervals (right). The number of outliers has multiplied.

3.2 Anomaly Detection

In Subsection 3.1.4, we have seen that many of our time series contain some amount of anomalies. But since there are not really any data alternatives available at the moment, we will have to look into this problem ourselves. Therefore, we will dedicate this chapter to the problem of Anomaly Detection.

To motivate this, let us look at an example what can and will happen, if we do not remove outliers first. We have seen that the outdoor temperature measurement from the Hillerød data does not contain temperature measurements below freezing. Instead, the time series contains measurements of a few thousand degrees. If we do not remove these outliers and directly perform a linear interpolation to some shorter intervals, it should come as no surprise that the interpolation is also performed between the regular the erroneous data points. In fact, as we can see in Figure 3.7, the number of outliers is now a multiple of what it has been before.

In this specific example, we can easily provide remedy by removing the obvious outliers from the data before interpolating. It is easy in this example, because we can safely assume that the outside temperature in Denmark or, frankly, on the planet Earth will not surpass the threshold of 100°C[14], so we can simply remove all data points above this threshold. Unfortunately, it cannot always be that straightforward.

To recall the definition again, anomalies are "patterns in data that do not conform to expected behavior" [CBK09]. As we have already discussed in Chapter 2, this is always subject to interpretation and we have seen in Subsection 3.1.4 that it is often non-trivial and/or ambiguous. Thus, we will have to proceed with some caution here - in particular, as we have already mentioned that many anomaly detection algorithms are of little value.

It is our goal to remove at least the obvious outliers of the kind we have seen in Figure 3.5a

[14]We're being very generous here.

or Figure 3.7. However, it is rather impractical to sort them out manually, as we did for Figure 3.5b. For example, while we can can safely remove outside and water temperature values above 100°C, we can remove negative temperature measurements only for the water pipes. We cannot remove negative outside temperatures as well. Flow values are higher than the temperature values, so we would need a different threshold there. For pressure measurements, we always need to keep their unit in mind, since with the identity 1bar=100,000Pa, the threshold above which we define values as outliers will have to be a vastly different one.

Therefore, some kind of automatic detection would be very helpful. That way, we can probably also detect outliers of the kind we have seen in Figure 3.5c and Figure 3.5d. We have stated, that we cannot say with certainty whether those outliers are correct or erroneous measurements. Nevertheless, even if those measurements are correct, they will probably impair any kind of data reconstruction, since they are sparse, yet far removed from the main, "regular" part of the data. Thus, any kind of interpolation will eventually lead to similar behavior as depicted in Figure 3.7.

Because there are so many anomaly detection techniques, we prefer to start with those that are already implemented for use. We find that the Python module `Scikit-learn` provides some methods to detect outliers in data sets [PVG+11]. We can use these methods for detecting outliers in unknown data, as well as for training them on unpolluted data and then predicting novel data. Its documentation provides an overview about the different methods on some example datasets[15]. In the following, we will give short examples on what these methods do and then examine them on some of our own data.

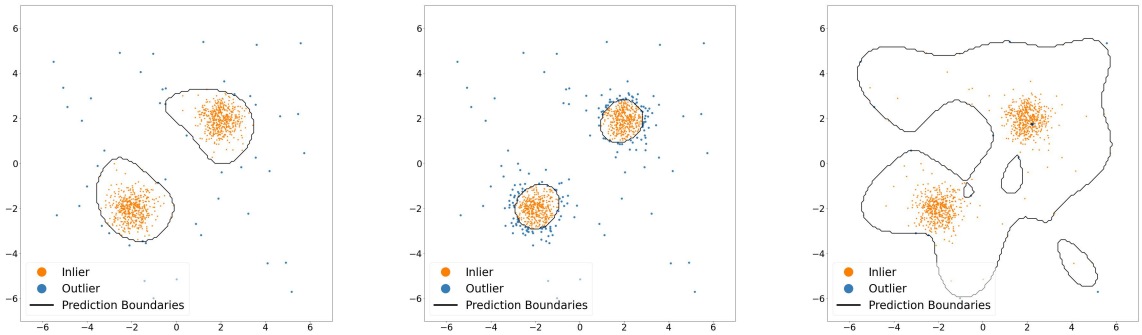
Note, that we have to set a proportion of outliers in the data for all of these methods. Default values exist for each method, but they do not lead to satisfying results in any case. That means, we have to make some kind of educated guess *beforehand*, how many outliers the data contains and then "instruct" the outlier detector how many outliers it is supposed to find. We can see very nicely in our example in Figure 3.8, in what way different choices will lead to different results. However, a general value will only be a very rough estimate - while the share of "obvious" outliers in the `FRE_FLOW` measurement from December 2021 (see Figure 3.5) is only about 0.21% of all measurements, that share is about 7.36% for the outdoor temperature measurements (see Figure 3.7) - which, in turn, only originates from winter and spring, while there are no such outliers during the summer months. For now, and unless otherwise noted, we have set this parameter to 2% for all cases.

Before we examine the outlier detection mechanisms on some of our own data, we will first look at their behavior in general. For that purpose, we have built a little toy dataset with two blobs[16] of data, 950 values in total, and add 50 randomly generated outliers to it. That means, we end up with a dataset containing 1,000 values and an outlier fraction of 5%.

3.2.1 OneClassSVM

According to its documentation, the `sklearn.svm.OneClassSVM`[17] performs outlier detection, although it is sensitive to outliers and therefore does not perform well for outlier detection. This murky description arose our curiosity, so we tried it anyway. Spoiler alert: indeed, it does not perform well for outlier detection.

[15] `https://scikit-learn.org/stable/modules/outlier_detection.html`
[16] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`
[17] `https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html`



(a) 5% share of outliers. (b) 20% share of outliers. (c) 1% share of outliers.

Figure 3.8: Outlier Detection with `OneClassSVM` for the same toy dataset, but with different preset shares of outliers. Orange points are inliers, blue points are outliers; decision boundaries in black.

The basic idea is to identify a sphere that encloses (most) of the data [NHR12]; here, exactly $1 - x\%$ of the data, where x is the (predefined) fraction of outliers.

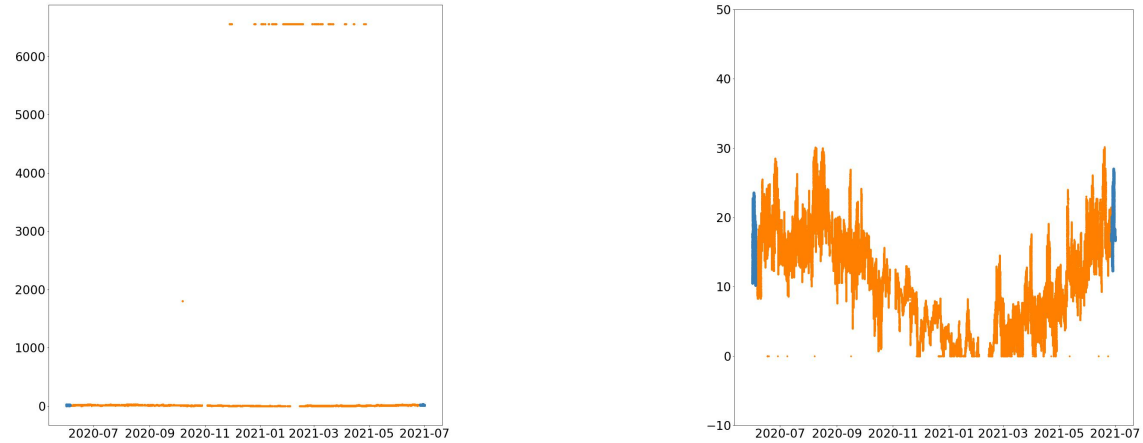
We can see the resulting behavior in Figure 3.8, where the data "inliers" are colored in orange, while the data points identified as outliers are colored blue. In addition, we show the decision boundaries in black; for that, we train the outlier detector on the toy dataset first and then predict the values on an evenly spaced grid[18]. If we set the share of outliers[19] to 5%, the two blobs are identified quite well, as Figure 3.8a shows. If we choose a higher contamination parameter, i.e. we expect a share of outliers of 20%, the blobs can still be identified in Figure 3.8b, although the decision boundaries are closer to their centers now - as we have expected, since we are forcing the SVM to find more outliers now. In contrast, if we choose a lower share of 1% instead, the result is now unsatisfying. The decision boundaries are quite unorthodox; but we can attribute this to the fact that there are only 10 outliers to be found in total, so some weird shapes can certainly be expected to appear here. However, while one point in the bottom right corner is not defined as an outlier, two points very close to the center of the right blob are - this is definitely not what we would have expected or hoped for.

Now let us examine some of our own data, for example the outside temperature measurements from the Hillerød data (cf. Figure 3.7). We can see the results of the outlier detection in Figure 3.9.

It is apparent immediately, that the obvious outliers at the top are not identified. Instead, the first and last data points are chosen to be outliers, although these points are perfectly fine. If we increase the share of outliers to the (for this time series more realistical) 7%, this simply leads to a larger chunk at the beginning and end of the series being identified as outliers, instead of some of the most obvious ones.

Interestingly, if we choose a different time series, the obvious outliers are sometimes identified correctly as such, e.g. for the `FRE_FLOW` from December 2021. But the start and end of a time series are always incorrectly identified as outliers, regardless of which time

[18] `https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html`
[19] The parameter for the outlier detector, that is. The share of outliers in the data set is 5% by design.



(a) The obvious outliers at the top of the plot are not identified.

(b) The left and right ends of the data are (wrongly) identified as outliers.

Figure 3.9: Outlier Detection with `OneClassSVM` for the outside temperature data from June 2020 to June 2021 with the share of outliers preset to 2%.

series we chose.

Moreover, the computation takes a very long amount of time. Without benchmarking it and just measuring the running time with IPython magic[20], the outlier detection on this single time series consisting of 367,394 values takes more than 30 minutes. All in all, this method does not seem to be very helpful for our task.

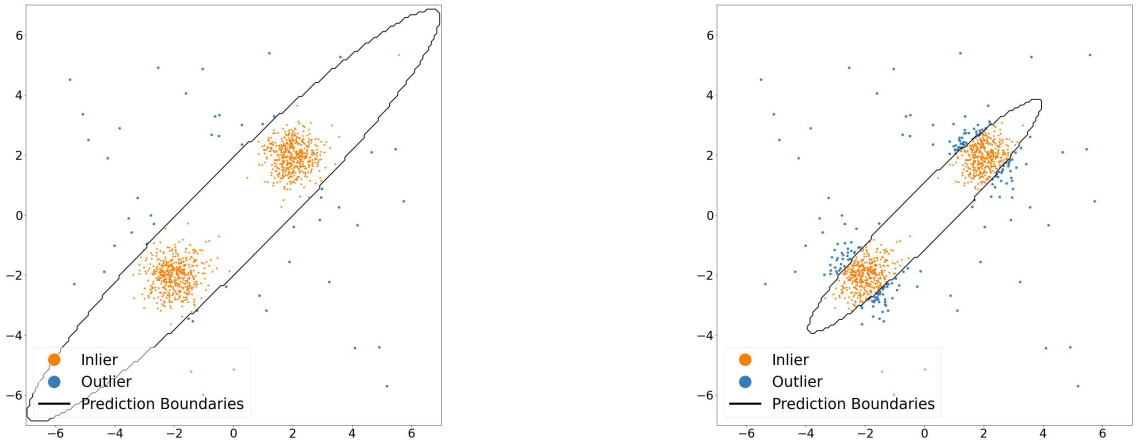
3.2.2 EllipticEnvelope

The `sklearn.covariance.EllipticEnvelope`[21] assumes that the data is Gaussian. That requires the data to be unimodal, which our data definitely is not. However, as the name suggests, it will learn to fit an ellipse around the data, as we can demonstrate on the toy dataset in Figure 3.10.

Because the toy dataset is bimodal, the ellipse does not fit very well. It includes a lot of "outlier space" between the two modes and potentially beyond them. But for our application, this might become useful for our task. If we can enclose the main part of the data inside this ellipsis, we could reliably identify the most obvious outliers.

First, it stands out that the `EllipticEnvelope` is much faster than `OneClassSVM` for large amounts of data. And indeed, if we consider the `FRE_TF` temperature measurement during the first quarter of 2022 as an example, we can see in Figure 3.11a that the elliptic envelope actually works quite well for data that lacks larger fluctuations. As we had wished for, the ellipse forms a band around the main part of the data. We get the same results for similar time series, including those that actually contain faulty measurements far away from the original data. Still, it is debatable whether the data points that have been identified

[20] `%%time` or `%%timeit`, see `https://ipython.readthedocs.io/en/stable/interactive/magics.html`
[21] `https://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html`



(a) 5% share of outliers.

(b) 20% share of outliers.

Figure 3.10: Outlier Detection with `EllipticEnvelope` for the same toy dataset, but with different preset shares of outliers.

as outliers here are actually incorrect measurements as well - they do not strike us as particularly odd. Nevertheless, removing those would make reconstructing the data easier.

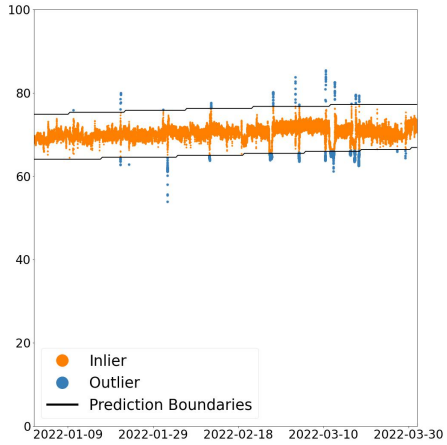
However, the `EllipticEnvelope` doesn't work that well on all the datasets. In particular, time series with more variations are not handled very well. We can see in Figure 3.11b that the ellipse cuts the graph of the `FRE_FLOW` measurement and misidentifies some part of it as outliers. We suspect in this case that the reason could be the simple fact that there are no outliers - we can detect none in this graph - yet we force the algorithm to find some. Therefore, we can look at a different time series in Figure 3.11c, that contains some points that might be considered outliers, as we have discussed in Subsection 3.1.4. But in this case, the ellipse is somewhat tilted, cutting off a relatively large part of data that does definitely not contain outliers.

For datasets with changes of the level of temperature, as in Figure 3.11d, this method does not provide meaningful results. On the outside temperature data (cf. Figure 3.9), the elliptic envelope even fails miserably. We cannot find any different results, regardless how we tune the parameters. Besides, while it is faster than the `OneClassSVM`, this method is still relatively slow for larger amounts of data. While it takes about 8.6 seconds to identify the outliers in the `FRE_FLOW` dataset with 29,725 measurements, the measurement `GOERLOESE_BY_FLOW` with 369,088 measurements already takes about 2 minutes. All in all, this method does not seem to be a very good fit for our data.

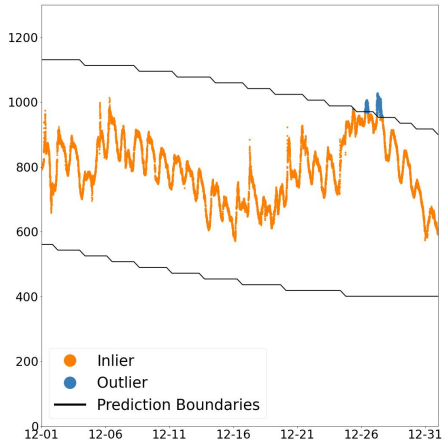
3.2.3 LocalOutlierFactor

Another option is the `sklearn.neighbors.LocalOutlierFactor`[22]. The underlying idea is to assign a degree of being an outlier to every data point [BKNS00]. This degree depends on how isolated a data point is from its surrounding neighbors. Data points with a significantly

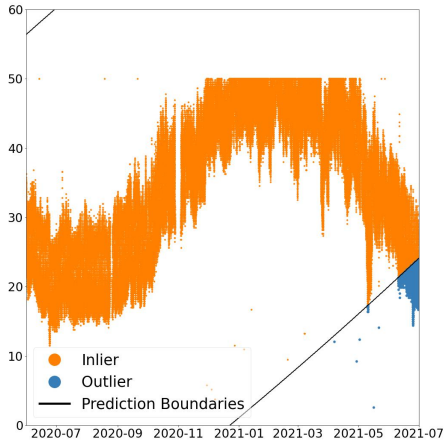
[22] `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html`



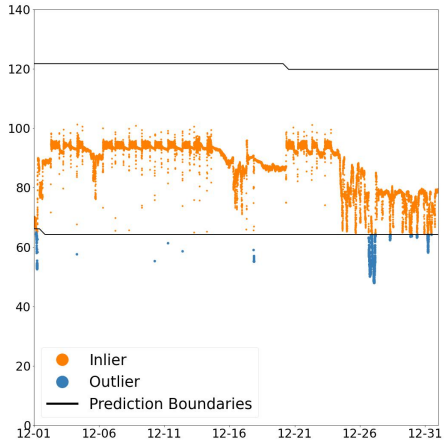
(a) `FRE_TF` temperature measurement during the first quarter of 2022.



(b) `FRE_FLOW` measurement during December 2021.

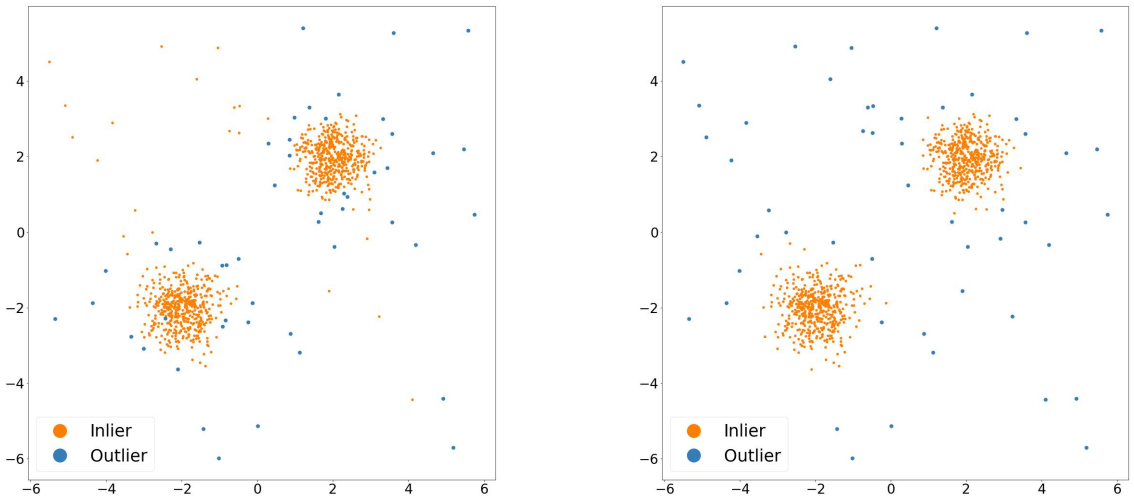


(c) `GOERLOESE_BY_FLOW` measurement over the course of one year.



(d) `HKV FREMLØBSTEMP` measurement during December 2021.

Figure 3.11: Outlier Detection with `EllipticEnvelope` on some of our datasets.



(a) 5 neighbors

(b) 20 neighbors

Figure 3.12: Outlier Detection with `LocalOutlierFactor` for the same toy dataset and a preset 5% share of outliers, but with different numbers of neighbors for a neighborhood.

lower density than their neighbors are then considered to be outliers. According to the documentation, this method also performs well for multimodal datasets[23].

There are two parameters that we may have to consider later. First, the metric for the distance computation. The specific choice of the distance metric can influence the results. But any L^p norm seems to be a reasonable choice in our scenario, so we will simply leave the default[24] in place.

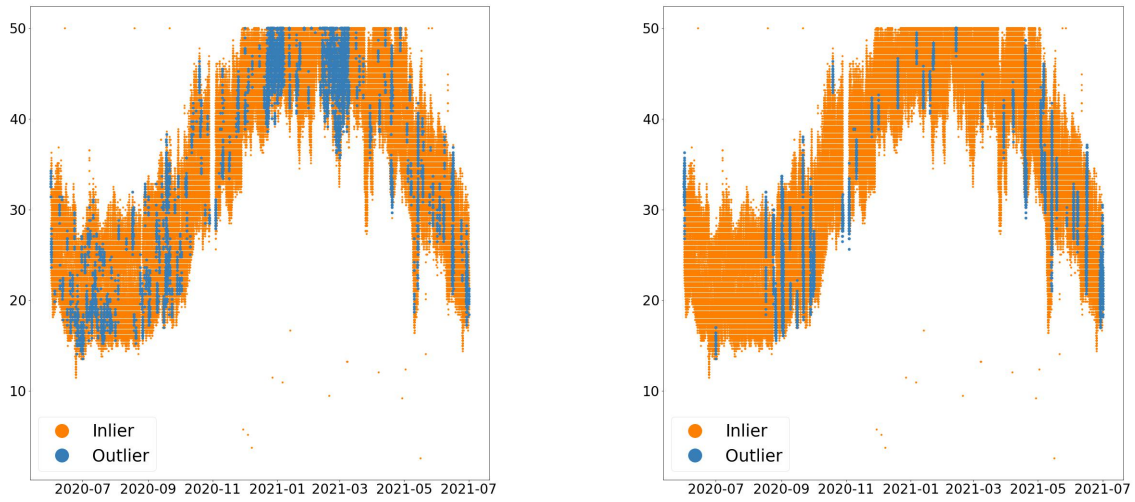
Second, the number k of neighbors under consideration for each data point. In particular, the vast outliers that we have mentioned before often appear as multiple outliers (with the same value) in a row. Therefore, we might fail to identify even the obvious outliers, as long as they are together with enough fellow outliers.

And indeed, if we take a look at our toy example again in Figure 3.12, we can see that the number of neighbors has a significant influence. If a neighborhood only consists of 5 neighbors, many outliers remain unidentified. The reason for this is simply that the neighborhood density for the isolated outliers in the top left quarter of the plot does not greatly differ from their 5 nearest neighbors - those are relatively isolated outliers as well. In contrast, those data points that actually are identified as outliers have 5 neighbors that are closer to the - much more densely populated - center. Therefore, their density is significantly lower than their neighbors', making them outliers themselves.

In contrast, if we set `n_neighbors` to the default of 20, the "outlying outliers" are identified, and the 2 blobs remain intact. Interestingly, if we raise the number of neighbors further, e.g. to 50, we do not find any changes in the results anymore.

We can attribute this behavior to the motivation of LOF, namely the detection of *rare*

[23] `https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_anomaly_comparison.html`
[24] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.minkowski.html`



(a) 20 neighbors

(b) 250 neighbors

Figure 3.13: Outlier Detection with `LocalOutlierFactor` for the dataset `GOERLOESE_BY_FLOW` and a preset 2% share of outliers, but with different values for `n_neighbors`.

outliers, e.g. criminal activities in E-commerce [BKNS00]. Since the LOF-paper is from the year 2000, we might consider to choose a more current example. Then, we find that the share of fraudulent card transactions out of all card transactions within SEPA was down to 0.036% in 2019 [Ban21].

But the approach to find rare instances might be a problem for us, since we are not necessary looking for rare outliers. Quite the contrary, since we can see in Figure 3.7 that outliers often appear in bulk.

And indeed, we can easily see that the `LocalOutlierFactor` method is largely of no use to us. We do not even have to look for examples that long, but we can just take our familiar `GOERLOESE_BY_FLOW` example again. As we can see in Figure 3.13, the obvious and the (at least) disputable outliers are not identified. Instead, datapoints out of the "regular" distribution are detected as outliers.

If we raise the number of neighbors that constitute a neighborhood, it reduces the amount of data points from the center of the data distribution to be accused of outlierism. But there is also a downside, since more data points at the beginning and end of the time series, as well as around a gap in the data during November 2020 are now identified as outliers.

Most importantly, regardless of how high or low we set the number of nearest neighbors to consider, the most obvious outliers are not identified. In fact, it seems like LOF refrains from choosing single data points as outliers altogether, but prefers to identify strings of multiple data points. This becomes more visible in Figure 3.14. Therefore, this method remains relatively useless for us.

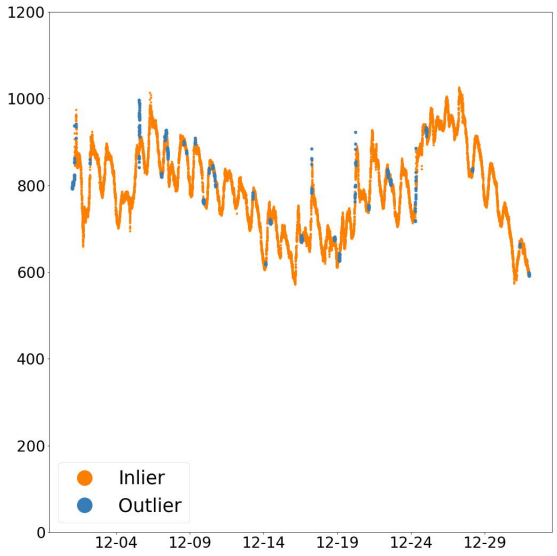


Figure 3.14: Outlier Detection with `LocalOutlierFactor` for the dataset `FRE_FLOW`.

3.2.4 IsolationForest

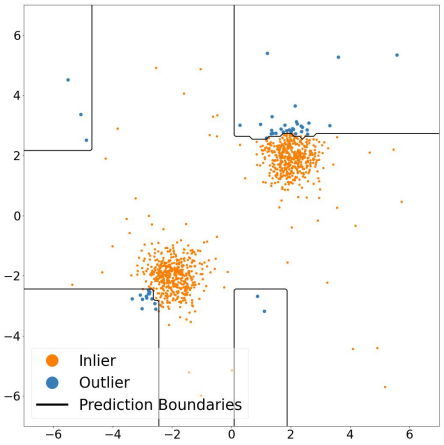
Instead of characterizing the normal data points first, and subsequently selecting non-conforming points as outliers, the `sklearn.ensemble.IsolationForest`[25] directly tries to isolate anomalies, because anomalies are "few and different" [LTZ08]. Basically, it tries to isolate data points by splitting the dataset across randomly selected split values. This partitioning is conducted recursively, so it can be represented in a tree structure. Since outliers can be isolated with fewer splits from the rest of the data, a data point with a shorter path length in the tree will likely be anomalous.

We focus on two hyperparameters of the algorithm that we can set manually: `n_estimators`, which is basically the number of trees in the forest; and `max_samples`, which is the number of samples that are used to train each estimator. Instead of choosing a fixed value, we can also set a share of the input data - which will come in handy for our own data, since the number of measurements can greatly differ between different time series. We can identify the splitting process by looking at our example in Figure 3.15.

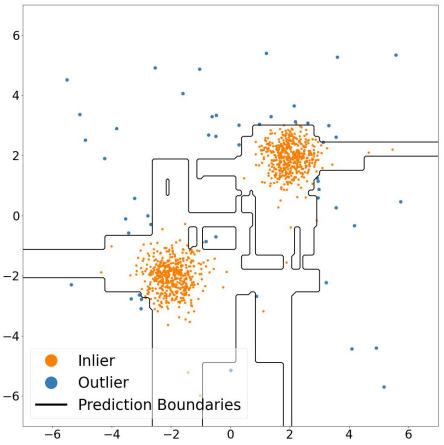
In general, we can say that:

- If we choose lower values for both hyperparameters, the decision boundaries are assessed quite generously.
- For a low number of base estimators, different choices of the number of samples have a low impact.
- Choosing higher values for the parameters will make the decision boundaries finer, although the effect of increasing the number of base estimators has a larger effect.
- If we set both parameters relatively high, we might actually get worse results, since the decision boundaries become too specific.

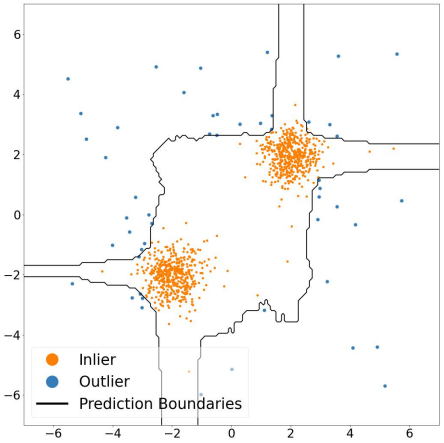
[25] `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html`



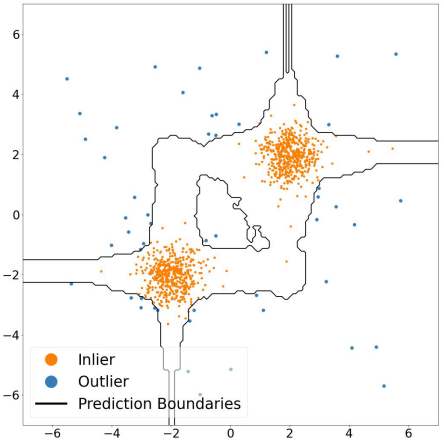
(a) `n_estimators` = 10
`max_samples` = 0.02



(b) `n_estimators` = 10
`max_samples` = 0.05



(c) `n_estimators` = 150
`max_samples` = 0.02



(d) `n_estimators` = 150
`max_samples` = 0.05

Figure 3.15: Outlier Detection with `IsolationForest` for the same toy dataset and 5% outliers, but with different hyperparameters.

We can also see that relatively large areas are inside the decision boundaries, while they clearly do not belong to one of the two blobs. But the reason for this is simple: These areas are empty, so there aren't any samples to choose in it. This may be a problem for novelty detection, i.e. outlier detection on new, unseen data; then, the training set has to be chosen very carefully.

But this problem does not need to concern us - we would just use it for outlier detection, where we already have all data available. We cannot (mis)identify data points as outliers, if they do not exist.

Moreover, the `IsolationForest` approach compliments our kind of data. Those outliers that are very far away from the rest of the data, i.e. those that we definitely want to remove, can be separated from the rest of the data with a single horizontal split. And since their values are so different, it is quite probable that we randomly select such a split. In general, it will be easier to isolate already quite isolated data points, than it will be to isolate data points from in between our measurement curve.

For outlier detection on our data, we will set the number of base estimators to the default 100 and draw 3% of the samples for each estimator, i.e. `max_samples` $= 0.03$. And indeed, we find the results to be quite promising, as we can see in Figure 3.16.

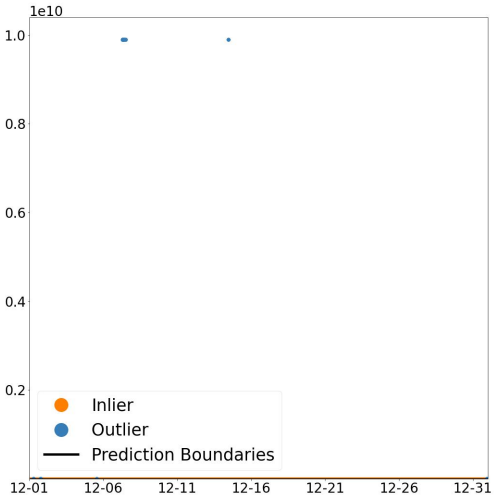
For the `FRE_FLOW` data, as well as for any other time series that we have tried, the outlying outliers (cf. Figure 3.5a) are identified correctly, as we can see in Figure 3.16a. In addition, we can see in Figure 3.16b that the decision boundaries nicely adjust to the distribution of the data. All obvious outliers are correctly identified, while all data points from the center of the distribution are correctly identified as inliers.

We might still argue here, that the algorithm "cuts away" too large a share of the variance of the data. However, we can attribute that to the fact that we estimated the share of outliers to be 2%. If we choose a lower value, here 0.5%, we find the results in Figure 3.16c quite satisfying. The algorithm still identifies the outliers as outliers, while keeping more of the "regular" peaks and troughs of the data. Even on more complicated data, for instance the supply temperature data measurements in Figure 3.16d, the isolation forest yields reasonable results.

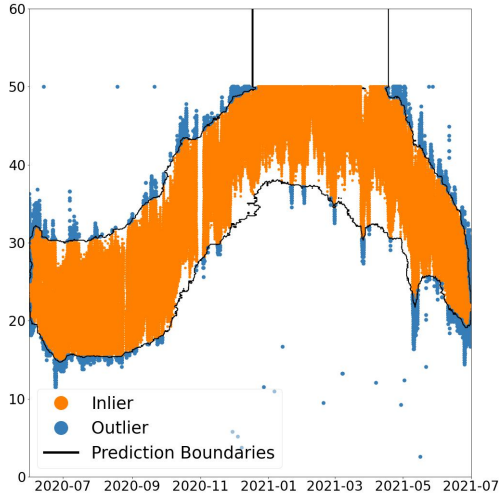
Unfortunately, there is still data on our hands which the isolation forest struggles with. It fails to identify a few of the infamous outdoor temperature values of 6,553.5°C. Again, the reason can be found in the algorithm's design. These outliers make up about 7% of the whole dataset (see Figure 3.17a). Therefore, they are neither few, nor different, so the algorithm's presumptions do not hold.

We can also consider the `ELM_TF` measurement for the first quarter of 2022 in Figure 3.17b. This contains fluctuations in the data between different temperature levels, similar to the kind that we have identified earlier in Subsection 3.1.4. In this example, it is definitely difficult to identify any data points as outliers in particular; there are many possible candidates, but all of them are at most questionable. Anyway, if the temperature changes between two different levels, there will be some (correct) measurements in between - but exactly those measurements will be identified as outliers by the `IsolationForest`.

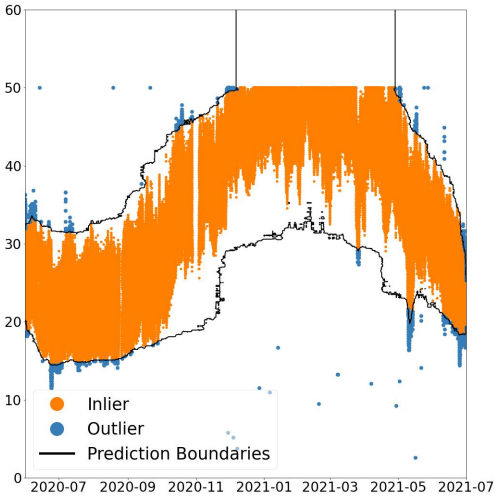
Nevertheless, it yields the best detection mechanism on our data so far.



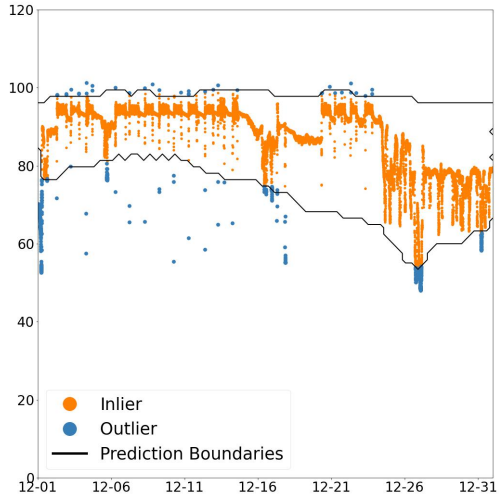
(a) `FRE_FLOW` data for December 2021.



(b) The familiar `GOERLOESE_BY_FLOW` data.

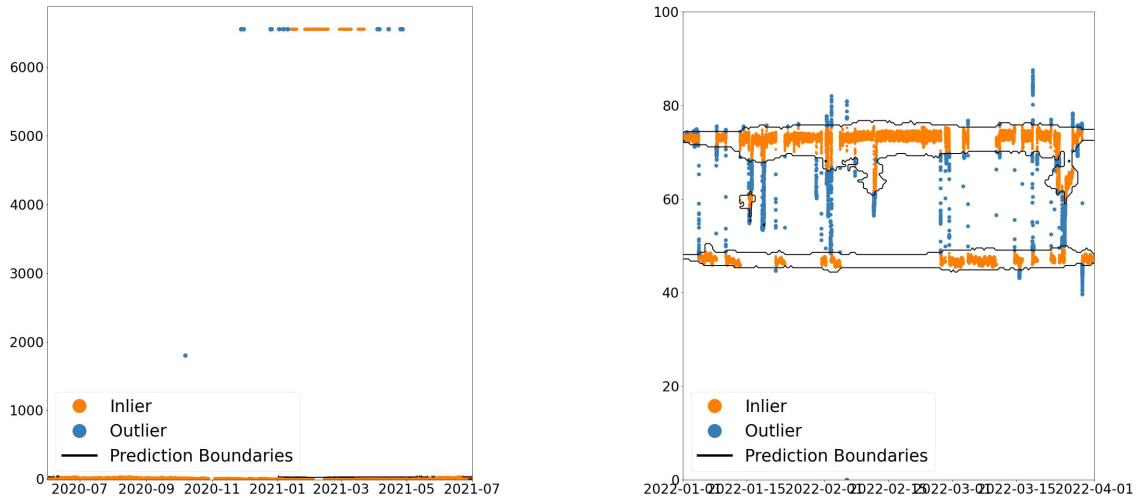


(c) The familiar `GOERLOESE_BY_FLOW` data. Here, we have chosen a smaller share of outliers, namely 0.5%.



(d) The `HKV` supply temperature time series during December 2021.

Figure 3.16: Outlier Detection with `IsolationForest` for some of our datasets.



(a) Not all of the extreme outliers of our infamous outdoor temperature measurement are detected.

(b) The `ELM_TF` supply temperature during the first quarter of 2022.

Figure 3.17: Outlier Detection with `IsolationForest` for some of our datasets, where the outlier detection fails.

3.2.5 ECOD

With PyOD, there is another open-source API for outlier detection in Python [ZNL19][26] that provides additional options for outlier detection. Our experiences with the `LocalOutlierFactory` in Subsection 3.2.3 show us, that proximity-based algorithms might not be the best choice for our data. A short trial of a standard KNN approach[27] supports that claim, since the results are similarly unsatisfying. Linear models do not strike us as particularly useful either, which we can confirm after a short trial of PCA.

But PyOD also provides probabilistic models. In particular, we have examined the very recent `ECOD`[28] algorithm, which is based on cumulative distribution functions [LZH+22]. It is especially charming, because beside our (estimated) share of outliers, we do not need to set any other parameters for it.

If we try to detect outliers on our sample dataset again, we can see in Figure 3.18 that `ECOD` seems to fit a rectangle with rounded vertices, where all the data points outside of the rectangle are identified as outliers. Hence, we might end up with similar results as our `EllipticEnvelope` from Subsection 3.2.2. Unfortunately, we had to draw this rectangle ourselves, because our beloved decision boundaries do not really work for the probabilistic PyOD methods. A boundary is drawn, also one in the shape of a rectangle with rounded vertices, yet it does not fit the displayed data.

And indeed, while the results turn out to be better in general and are being delivered much faster than the elliptical envelope, the detection pattern is similar. On our

[26] `https://pyod.readthedocs.io/`
[27] `https://pyod.readthedocs.io/en/latest/index.html#implemented-algorithms`
[28] `https://pyod.readthedocs.io/en/latest/pyod.models.html#pyod.models.ecod.ECOD`

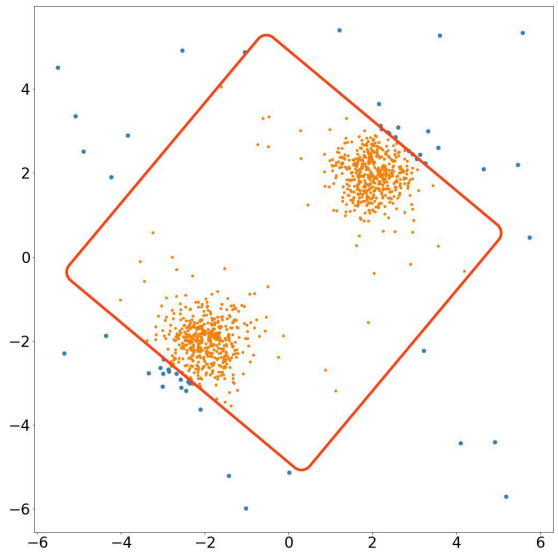


Figure 3.18: Outlier detection with `ECOD` on our familiar example dataset.

`GOERLOESE_BY_FLOW` time series, it identifies all except one of the single-point outliers in the bottom right corner. Unfortunately, it fails to identify their counterparts in the top left corner. Instead, it cuts away data at the beginning and end of the time series - a behavior that we can always detect to some extent, e.g. in Figure 3.19b.

For time series with more variance though, for instance the `ELM_TF` measurement in Figure 3.19c again, `ECOD`'s results look "nicer" than those of the forest. However, `ECOD` would simply not detect outliers somewhere in between at all, even if they were quite obvious. Therefore, its performance is inferior in comparison to the `IsolationForest`.

3.2.6 Identifying Extreme Outliers

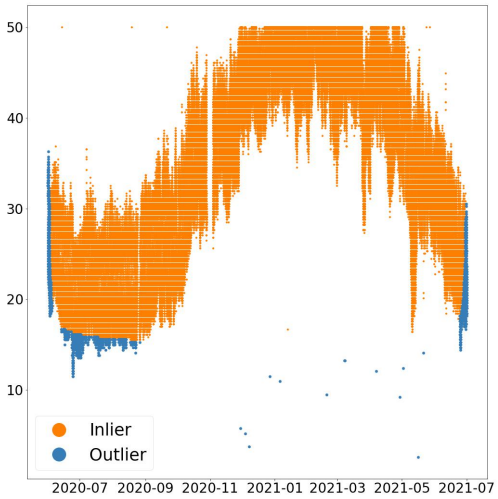
In the previous sections, we have examined multiple algorithms for anomaly detection. Out of those, the isolation forest provided the best results in general.

However, all of the mechanisms we have considered so far failed to reliably identify extreme outliers, as long as these outliers are not isolated, but appear in bulk - which they often do, since those measurements are clearly faulty. Unfortunately, we cannot simply filter out specific values, since the exact value can differ from time series to time series. Nevertheless, those faulty measurements have at least two things in common that we can use:

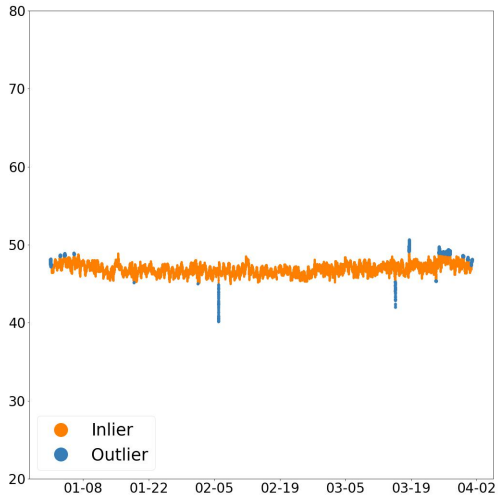
1. They are far away from the rest of the measurements, value-wise.
2. They all take on the same value *within the same series*.

Hence, if a series contains such extreme outliers,

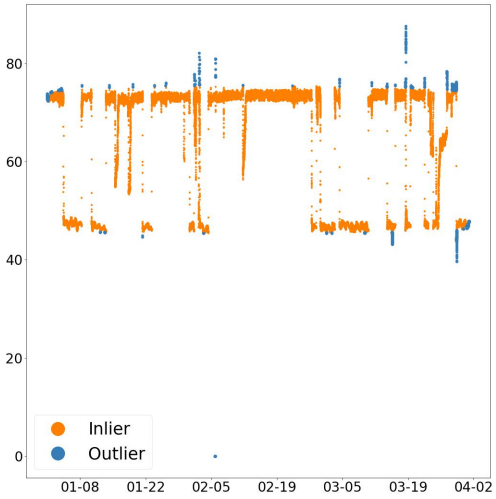
1. their value will be the maximum value of the series and
2. if we remove them, the remaining values will be significantly lower.



(a) The `GOERLOESE_BY_FLOW` measurements.



(b) The `ELM_TR` supply temperature during the first quarter of 2022.



(c) The `ELM_TF` supply temperature during the first quarter of 2022.

Figure 3.19: Outlier Detection with `ECOD` for some of our datasets.

Therefore, we propose the very simple Algorithm 1 for each time series. Let us denote our time series with $X = \{x_i\}_{i=1}^{n}$, its maximum value with $m_1 := max(X)$, and let $m_2 := max(X \setminus \{m_1\})$ be the maximum value of the time series without m_1.

Now, if a value m_1 is an extreme outlier of a time series, that means that it will be significantly larger than all other data points in that time series. Specifically, it is also larger than the next-largest data point m_2. We just have to define how much larger it has to be to make it an extreme outlier, and set this necessary distance to

$$m_1 > m_2 + 3\sigma \tag{3.1}$$

where σ is the standard deviation of $X \setminus \{m_1\}$. In particular, we do not choose the standard deviation of X here, since that is largely influenced by m_1, if m_1 is an extreme outlier.

Algorithm 1: Remove extreme outliers from a single time series

Data: $X_i, i \in I \subset \mathbb{N}$ time series, with $|I| = n$
Result: $X_j, j \in J \subset I$ time series without extreme outliers.

```
m_1 := max(X)
X' := {x ∈ X : x < m_1}
m_2 := max(X')
if m_1 > m_2 + 3 * std(X') then
    return X'
else
    return X
```

Note, that according to Algorithm 1, we remove all measurements that are equal to the maximum value of the time series. But for time series of constants[29], the computation of the maximum of X' will fail, since $X' = \emptyset$ in this case. Similarly, in time series where the maximum value occurs often (a time series with categorical values, for instance), we would remove most values, without them actually being outliers. In particular, values that appear very often cannot be considered outliers by definition, since the occurrence of that value would be an expected behavior.

What *often* means in this case, is up for interpretation though. We will arbitrarily set this threshold to 15%. One might argue, that a lower value could already qualify to be considered *often*. However, as we have pointed out, some of our datasets contain a considerable amount of these extreme outliers. For example, the Hillerød outside temperature measurement has a share of about 7.3% over the course of one year; if we just look at the winter months, it will be even larger.

In addition, we do not have to rely on the fact that all of these extreme outliers assume the same value. In fact, we have noticed extreme outliers on different levels. To find all of them, we can simply run our algorithm again recursively with the reduced time series. Overall, we'll end up with a slightly modified Algorithm 2.

Another commonly used option would be to simply remove all values x_i of a time series where $x_i > a\mu$ holds, where a is some constant and μ the mean or median of the time series.

[29] Yes, those exist in our datasets.

Algorithm 2: Remove extreme outliers from a single time series

Data: $X_i, i \in I \subset \mathbb{N}$ time series, with $X.std() > 0$, $|I| = n$
Result: $X_j, j \in J \subset I$ time series without extreme outliers.

if $|\{x \in X : x = m_1\}| > 0.15n$ **then**
    return X
$m_1 := max(X)$
$X' := \{x \in X : x < m_1\}$
$m_2 := max(X')$
if $m_1 > m_2 + 3 * std(X')$ **then**
    return Algorithm(X')
else
    return X

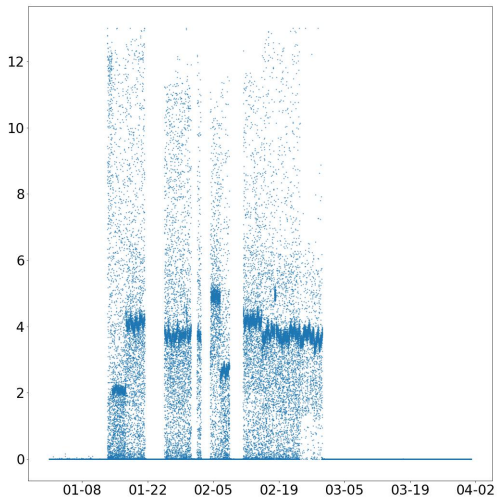


Figure 3.20: Some time series contain a significant amount of time period with constant values, skewing the mean and median.

However, there are multiple time series in our data lake that contain a constant value over a longer period of time, for example the one in Figure 3.20. More than 61% of its data is equal to or just slightly less than 0. Here, the median is 0 and the mean is roughly 1.3. That means, if we chose $a = 3$, we will remove all data points larger than 0 or larger than 3.9 - i.e. most or all of the interesting data. In addition, we would end up with completely different results if we choose a different time period - most of the zero-measurements are from January and March. So if we only perform outlier detection during February, we will identify less outliers, although we are basically looking at the same data.

Arguably, the measurements that are different from 0 in Figure 3.20 are, indeed, outliers, because most measurements are actually (almost) 0, so that should be the expected behavior. But this is counterintuitive - we could just get rid of the sensor completely then, because that would mean that the sensor is predictable and its information entropy would be 0 [Sha48]. If anything, a long period of constant values, as in this example, seems suspicious in and of

itself.

But we also cannot simply remove this without knowing more about the sensor. Maybe the sensor is defective and transmits erroneous measurements. But maybe the sensor is working fine and there is something wrong with its environment. Thus, removing constant values may not be advisable as a general solution.

3.2.7 Evaluation

After examining many anomaly detection algorithms in their functionality, a clear strategy has emerged:

1. Remove the extreme (and nonsensical) outliers with our own algorithm from Subsection 3.2.6.
2. Detect (and remove) further outliers with the isolation forest from Subsection 3.2.4.

With regard to the isolation forest, it seems to be a good idea to change some parameters. The authors of [LTZ08] underline that the method works better with lower sampling sizes. Therefore, we reduce the `max_samples` parameter to 0.01, i.e. to 1% of the data. We should also lower the number of expected outliers to 0.5%, so that the `IsolationForest` does not get too eager to identify more outliers than necessary, but only the most obvious ones.

We will now see that this seems to be a good approach. In Figure 3.21, the extreme outliers were reliably removed from the `ELM_TR` measurements, before the isolation forest does the rest. Even the infamous Hillerød outdoor temperature outliers - and only the extreme outliers - get removed now, thanks to our Algorithm 2.

The parameter tuning of the isolation forest also yields good results for more complicated data as the `ELM_TF` dataset. If we compare Figure 3.21b with the previous version in Figure 3.17b, the decision boundaries seem to be more chaotic on first impression. However, the selection of outliers seems much more reasonable, particularly in the center of the plot.

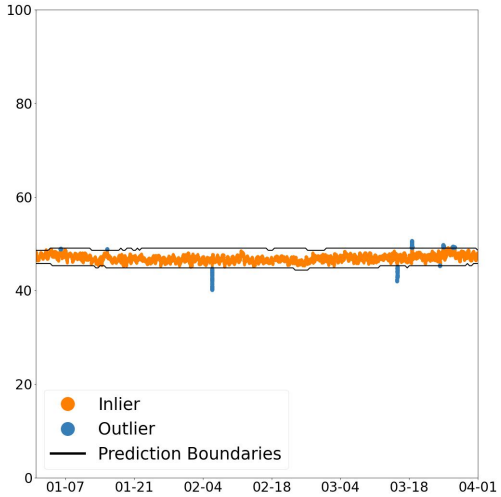
To evaluate the outlier detection more thoroughly, we add outliers to a measurement ourselves and see, whether they are identified correctly. That means, we

1. load a regular time series measurement from the first quarter of 2022,
2. generate additional data points, uniformly distributed[30] in the value range of the data,
3. add additional outliers that are larger than the maximum value of the time series,
4. perform an outlier detection and remove all points that were identified as outliers and
5. check the results.

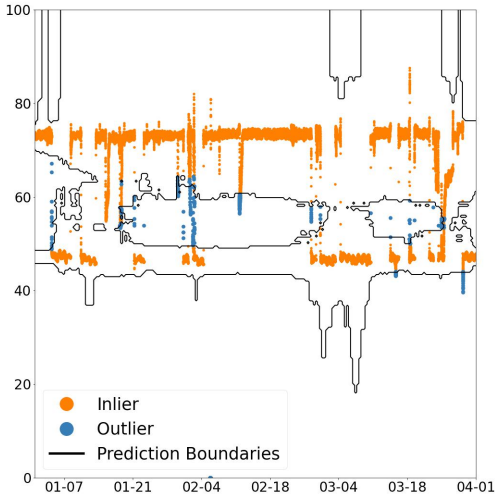
We would like to ensure three things:

1. All extreme outliers are removed from the data, i.e. those that are far away from the rest of the dataset.
2. All obvious outliers are removed from the data, i.e. those values that are in the possible range of values, but are far away from the nearest data points in time.
3. If a value is removed, it is not an obvious inlier.

[30] `https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.uniform.html`



(a) The ELM_TR return temperature for the first quarter of 2022. The extreme outliers have been removed by our Algorithm 2.



(b) The ELM_TF supply temperature during the first quarter of 2022. Outliers in blue. The decision boundaries are somewhat chaotic, but the results are reasonable.

Figure 3.21: Outlier detection with removing extreme outliers first and then identifying outliers with an isolation forest, with the share of outliers preset to 0.5%.

Hillerød

Given these three criteria, our detection mechanism works quite to our satisfaction. We can see in Figure 3.22 that of our generated outliers, both the extreme and the non-extreme but still obvious outliers are detected and removed. In contrast, those generated data points that fit into the distribution of the data are *not* removed, as we would have hoped for.

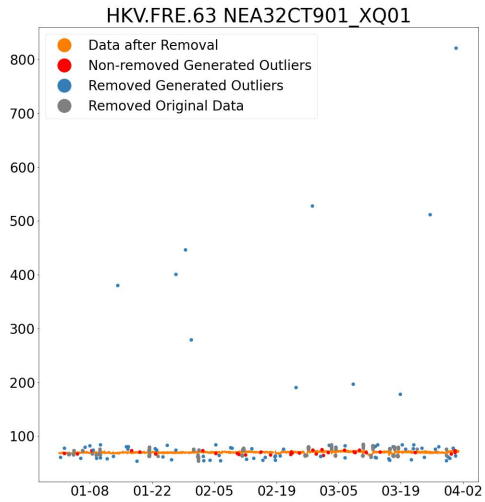
Unfortunately, the measurements from the original data that were removed are not necessarily outliers. While most of the data fluctuates around a supply temperature of 70°C, there are some spikes up- and downwards; but since these values are not just isolated measurements, these measurements can very well be correct. The reason is simple: we force the isolation forest to find some (small) share of outliers, so it finds them. Therefore, we should retain the option to just remove the extreme outliers, while leaving the rest intact. Nevertheless, the data points that were removed are also not obviously inliers; i.e. if we must remove any 0.5% of the time series, we should choose exactly those that have already been removed.

If we take another look at the decision boundaries of the isolation forest in Figure 3.23, we can confirm our findings from Subsection 3.2.4, namely that it will construct a band that clings to our time series, but is still broad enough to cover its variance.

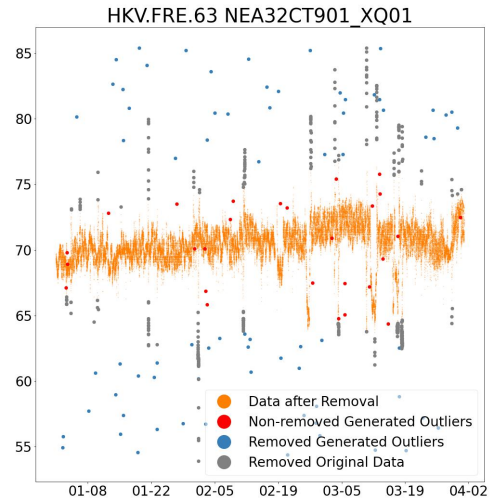
Brønderslev

In most of this chapter, we have relied on time series from the Hillerød dataset so far, since that is the most organized one. Now, we can check the two other datasets as well.

The Bronderslev dataset is more chaotic and so are the time series in it (cf. Subsec-



(a) The extreme outliers are identified correctly.



(b) Only generated data points that fit into the data distribution are left intact, the outliers are removed.

Figure 3.22: FRE_TF measurement after outlier detection and removal.

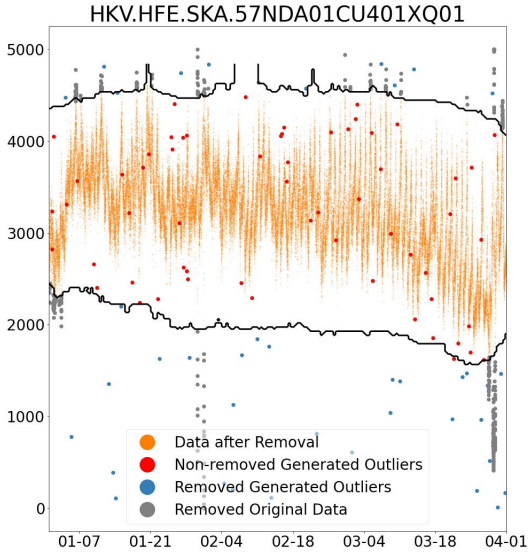


Figure 3.23: The SKAEV_EFF_PRIM dataset after outlier detection and removal, including the decision boundaries of the isolation forest.

tion 3.1.3). Nevertheless, the detection of extreme outliers works without problems. Whether the isolation forest works good as well highly depends on the time series under consideration, though.

For the return temperature measurement in Figure 3.24a, the outlier detection works similarly well as for the temperature measurements from the Hillerød data.

In contrast, Figure 3.24b contains power measurements from a heating circuit in the power plant. Most of the measurements are somewhere in the ballpark of 1-2 MW, but there are also a few chunks of measurements close or equal to 0. Without further information, we cannot say whether those low measurements are legitimate or not. But either they are erroneous, then we should remove them as well; or they are correct measurements, then we cannot remove all those measurements in between either. Given that there are so many measurements in between in the first place, likely neither those nor the zero-measurements should be removed.

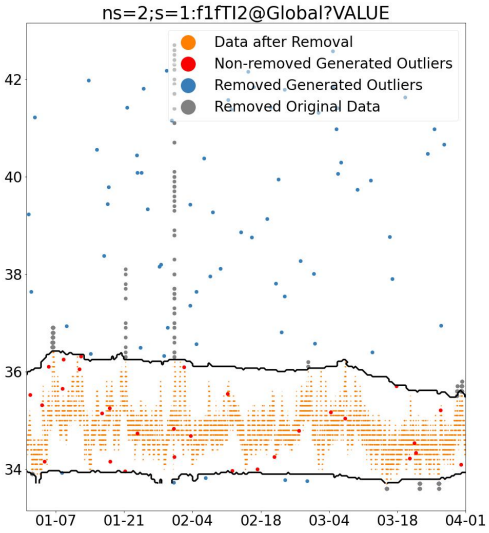
There are also time series as the one in Figure 3.24c that contain solar irradiation values. Here, the outlier detection works as before and as reasonable as we could expect. However, that does not necessarily make sense for this particular measurement. This time series contains measurements of the global solar irradiation from January to March - global meaning "overall" here, not "worldwide". It should come as no surprise that there is an upwards trend in general; the decision boundary of the isolation forest follows this trend nicely. However, there are large differences in the data from day to day, since the solar irradiation depends on weather conditions to a large extent [Sed21]. That means, on some days, the decision boundary is much too high - while a solar irradiation of 100 $\frac{\mathrm{W}}{\mathrm{m}^2}$ would be feasible for a generic day in January, it would be suspiciously high for the 9^{th} of January 2022 in particular. In contrast, removing the values above $100\frac{\mathrm{W}}{\mathrm{m}^2}$ on January 4 does not seem right either. It seems like this was just an unusually sunny day for this time of year, but not an erroneous measurement worthy of removal. And although I cannot remember seeing the sun a lot during January in Denmark, I do have photographic evidence that the sun was shining at least on the morning of January 6 in the capital region, so a sunny January 4 is not completely out of the ordinary.

We can use this opportunity to check the measurements for soundness as well. Via the Copernicus Atmosphere Monitoring Service (CAMS)[31], the ECMWF provides detailed solar irradiation data, based on satellite measurements (see [Sed21] for details). There, we can download solar irradiation measurements from January through March 2022 for the Brønderslev area and compare them with the measurements from our dataset.

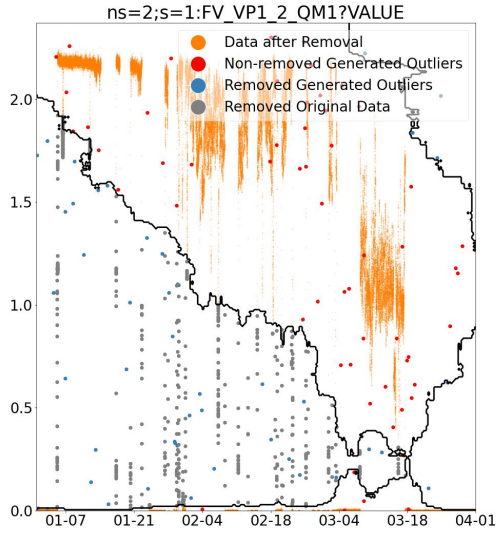
We can see the results in Figure 3.25, with the CAMS data in blue and our time series in orange. The difference between both measurements is additionally plotted in black. Clearly, the overall trend of our data matches the satellite data.

There is *some* difference in the values, but this can have different reasons. For one, we can check that our measurements of direct solar irradiation differ from the satellite measurements to a much greater extent; but the direct irradiation depends on, well, the ability to be directly irradiated by the sun. If the sensor is placed unfavorably, this can impair the direct and, consequently, the global irradiation. Also, the satellite measurements are interpolated to some degree, so they are not and cannot be completely precise for a specific location. It is not unheard of, that satellite data can strictly over- or underestimate ground truth

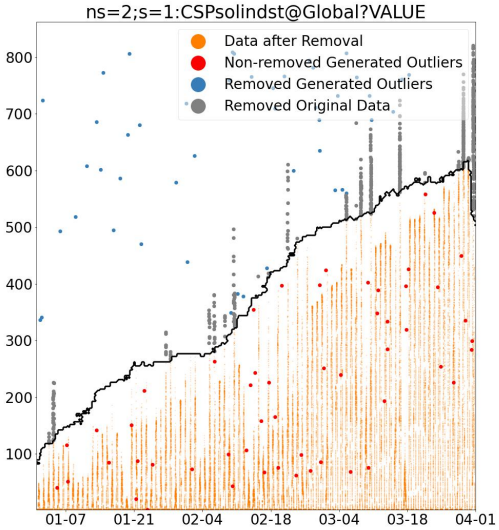
[31] `https://ads.atmosphere.copernicus.eu/`



(a) Return temperature measurement `f1fTI2`



(b) Power measurements in MW from a heating circuit in the power plant



(c) Solar irradiation measurement

Figure 3.24: Outlier detection and removal on three different time series from Bronderslev.

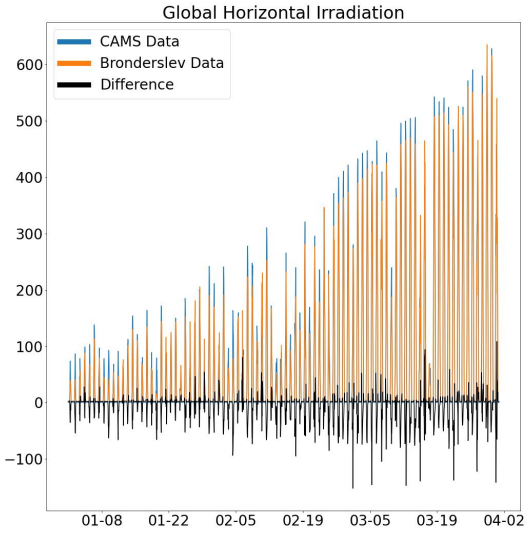


Figure 3.25: Comparison between global solar irradiation from the Bronderslev measurement (orange) and satellite data (blue), with the difference between the two in black.

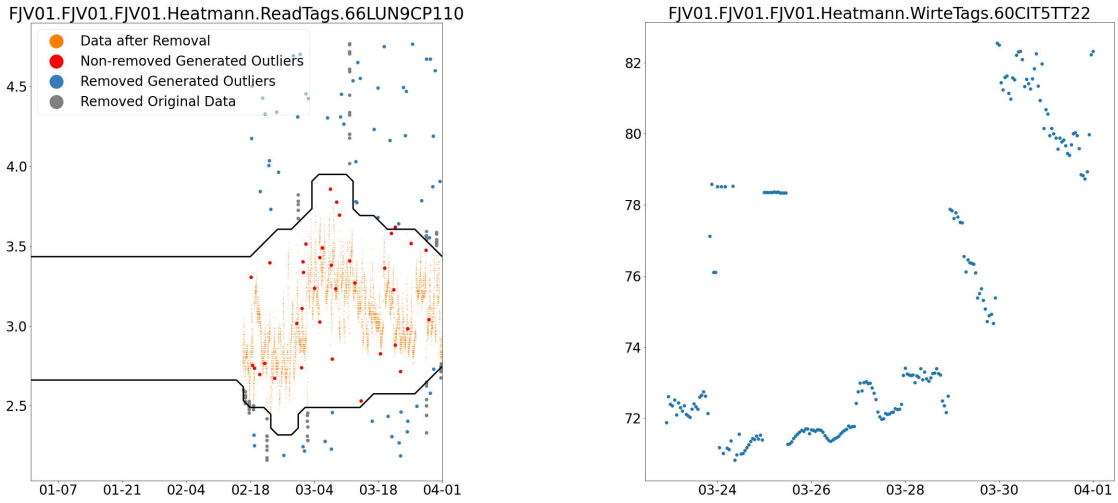
measurements [Sed21]. Therefore, we cannot say whether the satellite or our measurements are imprecise, or both.

Nevertheless, we do note that the trend coincides not only over the whole 3-month period, but also from day to day. That means, for example, the radiation measurements for the 18^{th} of January are low in our time series, when compared to the days before and after. But the radiation measurements from CAMS for the 18^{th} of January are low in comparison to the other CAMS measurements as well. Hence, these measurements seem to be quite sound.

Trefor

For the sake of completeness, let us also have a look at some time series from the Trefor dataset. To some extent, this is an adventure, as it contains many undocumented time series, while many documented time series do not actually exist. Despite - or rather, because - of this, we are pleased to find that our approach also works on randomly selected time series from the Trefor dataset. In Figure 3.26a, we see a measurement that is completely lacking data for the first $1\frac{1}{2}$ months of 2022. Afterwards, our outlier detection works just as fine as it did for the other time series in this section.

Another randomly selected measurement can be seen in Figure 3.26b. Interestingly enough, it only contains data for the last week of March, in a very interesting pattern. But what are the outliers here? Usually, outliers depart from expected behavior, but we definitely didn't expect anything like this at all. Thus, we don't know. Luckily, our Algorithm 2 and the isolation forest do not know either. Hence, they leave this dataset untouched, which is arguably the best solution.



(a) Mystery measurement with a long period of no data at the beginning.

(b) Mystery measurement with data only for one week in artistical patterns.

Figure 3.26: Outlier detection on time series from the Trefor data.

3.3 Data Reconstruction

A - or, one might argue, **The** - method to handle missing data is to avoid missing data in the first place. And, apparently, we are also not the first to argue in this manner [Kan13]. That, unfortunately, is not always in ones power to do. Therefore, we will examine techniques to recover, or rather reconstruct, missing data points in this section. To be able to do so, we should determine two things for a start.

First, we will have to determine the type of the missing data, charmingly termed as its *missingness mechanism* [Wil20]. Since the data that is missing is lost during transmission or due to sensor failures, we can simply regard it as *missing completely at random* [Kan13] - there is no apparent reason why we should assume otherwise. But this is great, since it allows us to ignore the process that causes the missingness of data [Rub76] and concentrate on the data itself.

Second, we need to determine how the reconstructed data should look like. We have examined in Subsection 3.1.3 how far regular data points in our time series are apart, in order to determine when data points are missing. We have found that the data points are usually 36 seconds, 90 seconds and 5 minutes apart for the Bronderslev, Hillerød and Trefor datasets, respectively (cf. Table 3.2). But we have also found that the measurements in the Trefor datasets often arrive delayed or prematurely, and that there is even more variance in the Bronderslev data.

At least the Hillerød measurements are 90 seconds apart from each other in more than 97% of the case in a particular time series. But these 90-second marks are not necessarily the same for all time series from that dataset. And if there is a gap in the data somewhere, the measurements are not necessarily picked up again at the same pace - i.e. if the measurements' time stamps are 00:30s, 02:00s, 03:30s, ... before the gap, they might very well be 00:15s, 01:45s, 03:15s, ... afterwards.

Considering all of this, we cannot simply insert the missing values where they would have been, because we do not know when that would need to be. Instead, we can construct a completely new time series with predefined timestamps, i.e. we start at a fixed date and then add a measurement after every d seconds. Taking our observations from Subsection 3.1.3 into account, we can choose d differently for every dataset. The median distances between two datapoints from Table 3.2 seem to be reasonable choices for d - for the Hillerød dataset, because it is overwhelmingly predominant anyway; for the Trefor dataset, because it can correct the late and early arrivals - except for those time series that contain hourly measurements. We can either try to interpolate them to 5-minutely values as well, or just choose a different time period. For the Bronderslev dataset, the median distance is not such a natural choice, but we need to make *some* choice for a value. This should neither discard too much available details, nor make everything up. Therefore, choices of some value between 30 and 40 seconds, i.e. close to the median value, still seem reasonable.

We also have to define, up to which length of a gap we want to recover the missing data. We have mentioned, that Johra et al. deem gaps of more than 9 consecutive hours of missing data for too long for interpolate. But we cannot apply this criterion to our data par for par. Apparently, a gap of more than an hour in one of their time series occurs when only a single data point is missing [JLH+20]. Thus, choosing 9 hours as an upper threshold for possible interpolation seems to be too high for our data, since we would already be missing much more data at that point. Instead, we might consider using 9 times the distance between

consecutive measurements as an upper threshold. But this would be 45 minutes for the Trefor data, but only $4\frac{1}{2}$ minutes for the Bronderslev data, although the laws of physics, including the time it takes for water to heat up and cool down are (approximately) the same for both.

As a result from conversations on this topic, we decide that it will be sufficient for now to consider only shorter gaps; therefore, we set the upper threshold of data gaps where imputation is still feasible to 2 hours for now. We can deal with this mainly in two ways. Either, we impute the data anyway, but use it with caution; or we split the time series at longer gaps and only impute those shorter slices.

3.3.1 Interpolation

A very easy way to impute missing values is mean imputation or mean substitution, i.e. to set missing values of a time series to the mean of that time series. Apart from the previously stated fact that we cannot simply replace missing values, but rather have to create new time series with measurements at new time stamps, this is not a good solution for our data in any way. Just a quick look at some of the previous plots suggests that the mean of a time series is often a very bad approximation. Especially for time series that switch between different value levels frequently, such as the one depicted in Figure 3.6a, the mean lies between those levels and thus is seldom or never a value of the time series itself, i.e. a bad approximation.

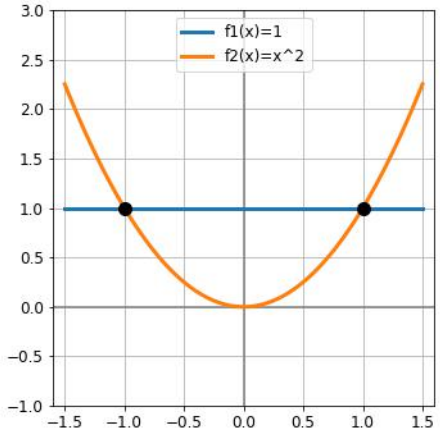
Polynomial Interpolation

Another easy, yet much more effective way to reconstruct the gaps in our time series is interpolation. Interpolation has the big advantage that it is independent from the missingness pattern of our available data, as well as from our desired target configuration. By independent, we mean that we can *apply* interpolation regardless of how often and for how long the input data is missing, and regardless of which time stamps we choose for our target time series. We can achieve this simply by using all available data points as supporting points of the interpolation, and by evaluating the interpolated function at our target dates. We do **not** mean by independent, that the results will stay the same. In fact, we expect the interpolation results to be closer to the original function, the more supporting (i.e. original data) points we have and the closer the evaluation points are to those supporting points.

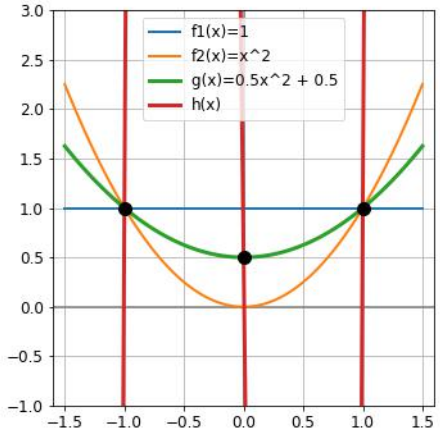
Unless this is already clear intuitively, consider a small example: Let us assume, that we have two pairs of (x, y), namely $p_1 = (-1, 1)$ and $p_2 = (1, 1)$, and we are looking for a function f such that $y = f(x)$ for both pairs. From p_1 and p_2, two possible interpolations are immediate, namely $f_1(x) = 1$ and $f_2(x) = x^2$, as we can see in Figure 3.27a. In fact, $f_1(x) = 1$ is the only linear - and in this case, constant - function that satisfies $f(-1) = f(1) = 1$.

If our data contains another point $p_3 = (0, 0.5)$, we can see in Figure 3.27b that both of our previous interpolations cannot be true. Instead, for example by solving the corresponding system of linear equations, we find that the - now unique - quadratic interpolation has become $g(x) = \frac{1}{2}x^2 + 0.5$.

But this is not necessarily the true underlying function that our data points follow. We can still interpolate our three points with an infinite number of polynomials of a higher degree - for example, $h(x) = 100x^3 + \frac{1}{2}x^2 - 100x + 0.5$ would also be a viable choice. While these functions can yield very different results between the supporting points (e.g.



(a) Supporting points at $(-1, 1)$ and $(1, 1)$, with two possible interpolations $f_1(x) = 1$ and $f_2(x) = x^2$.



(b) With another supporting point at $(0, 0.5)$, the (now unique) quadratic polynomial is $g(x) = \frac{1}{2}x^2 + 0.5$. Another valid interpolation would be $h(x) = 100x^3 + \frac{1}{2}x^2 - 100x + 0.5$.

Figure 3.27: Exemplary interpolation between supporting points. The closer to a supporting point we evaluate the different functions, the closer the results will be.

$g(0.5) = \frac{5}{8}, h(0.5) = -\frac{295}{8}$), these results will become eventually close as long as we are close enough to one of our supporting points, because our interpolation functions are continuous. In other terms, for a supporting point p_i it holds that $\lim_{x \to p_i} g(x) = \lim_{x \to p_i} h(x)$.

Yet, this very small example already shows us that interpolation with a single polynomial can yield virtually unusable results with just a handful of datapoints. Our goal is to is to get good estimates for the gaps between the data points that we already have, i.e. the supporting points - but the resulting polynomial exhibits large oscillations in these gaps, especially towards the end of our interpolation interval, therefore leaving us with very bad estimates in general (see also *Runge's phenomenon*). But in interpolation that only provides good estimates at places where we do not need estimates, because we have the original data, is basically useless.

Spline Interpolation

To remedy this problem, we can use linear or spline interpolation instead.

Linear interpolation simply computes a direct line between each pair of supporting points separately. That means, for two points (p_1, p_2) and (q_1, q_2), it yields

$$y = \frac{p_2(q_1 - x) + q_2(x - p_1)}{q_1 - p_1}$$

for any $p_1 \leq x \leq q_1$.

Spline interpolation uses piecewise polynomials of a low degree d, usually $d \in \{2, 3\}$, and chooses them in such a way that they fit together smoothly at the supporting points. That means, the resulting polynomial is continuously differentiable at the supporting points as well, which is not the case for linear interpolation.

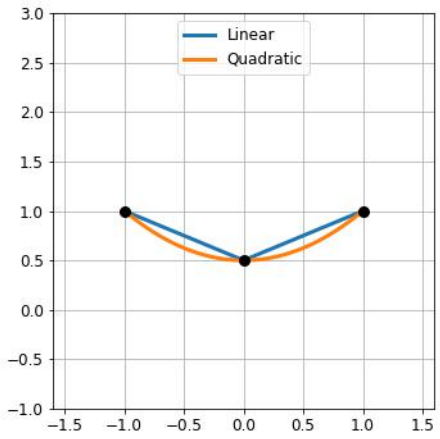


Figure 3.28: Example supporting points, with linear and quadratic spline interpolation. We can see that the linear interpolation is continuous, but not differentiable in $(0, 0.5)$.

Luckily, SciPy already provides tools for this[32], so dealing with Pandas and NumPy timestamps is, again, the largest struggle. If we apply a linear and a quadratic spline interpolation on our small example from before, we can see in Figure 3.28 that the result of the quadratic interpolation is very similar to our solution for $g(x)$ from before.

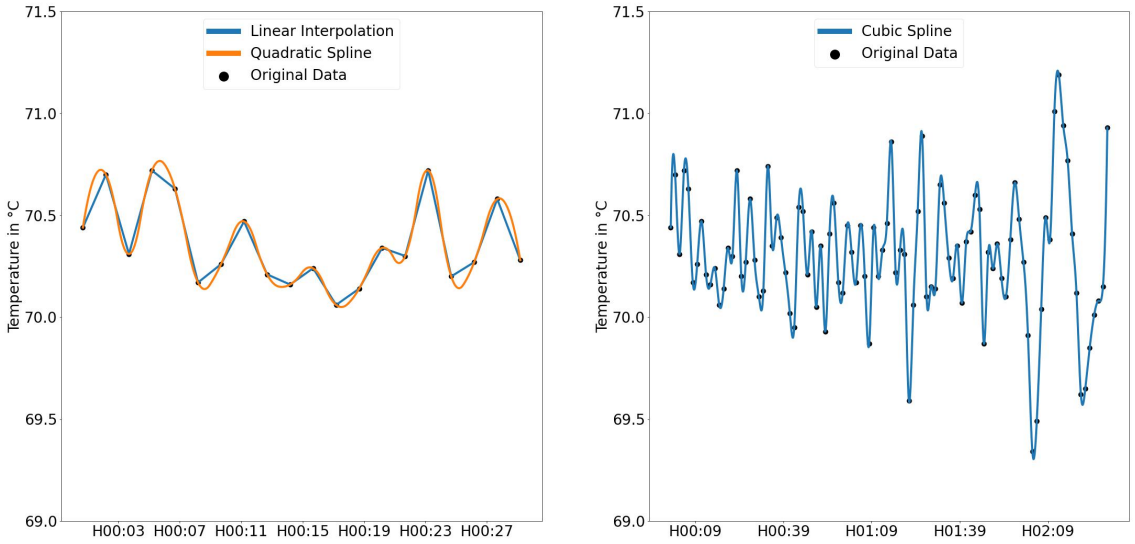
After we have figured out all timestamp-related programming issues, we can now try this with some of our real data as well. If we consider the first measurements of the `GOERLOESE_BY_TF` time series in 2022, we can see that the measurements oscillate approximately in the range of 69-71°C during the first two hours of 2022. In Figure 3.29a, we can see a quadratic spline interpolation of the first 20 values in comparison to a linear interpolation of the same values. Both interpolation methods yield very similar results. The only recognizable differences appear where the curve changes direction. Here, the linear interpolation cuts short of the spline interpolation curve. But even these differences are small when we consider how small the difference in temperature and time is here.

This should become clearer in Figure 3.29b, which shows a cubic spline interpolation of the first 100 datapoints. If we pick any pair of consecutive data points and imagine a straight line between them - and nothing else is linear interpolation - this line would closely follow the spline interpolation that is depicted. If we plot the linear interpolation on top again, we would see nothing, really; so we leave this up to the reader's imagination. Thus, we can see no reason why we should prefer one over the other here. Having said this, the spline interpolation seems to be a more natural fit than the linear interpolation - water temperature cooling or warming that abruptly contradicts our own experience.

Let us look at a similar plot for our infamous Hillerød outdoor temperature measurements in Figure 3.30. If we take a similarly detailed look on just a handful of measurements, we will see that the values in this time series are actually not continuous, but discrete measurements, i.e. the temperature is measured in steps of 0.1°C.

Here, both methods seem at least odd, since they transform a discrete measurement into

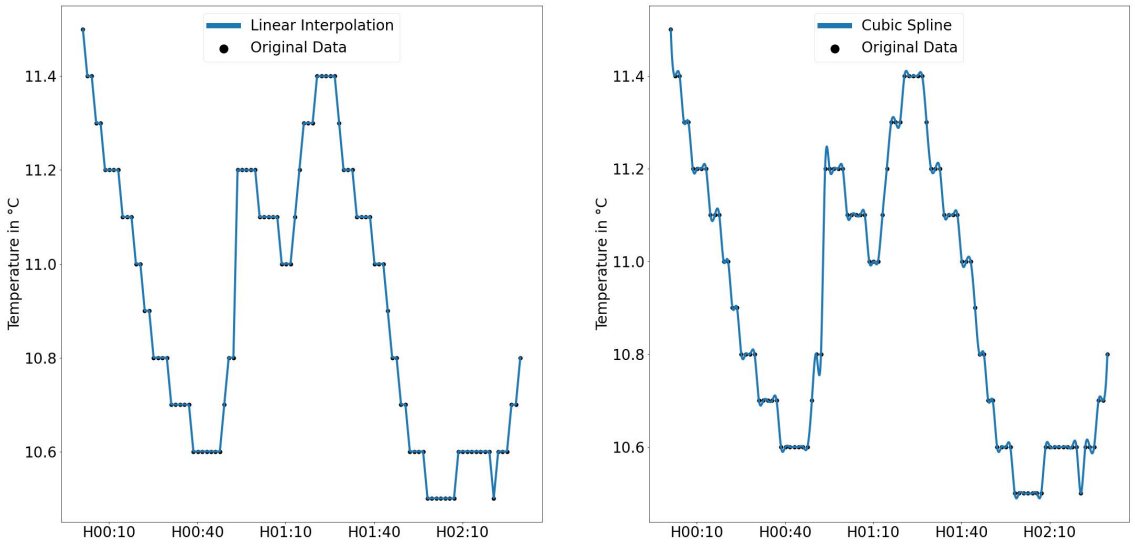
[32] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp1d.html`



(a) Linear and quadratic spline interpolation of the first 20 measurements in 2022.

(b) Cubic spline interpolation of the first 100 measurements in 2022.

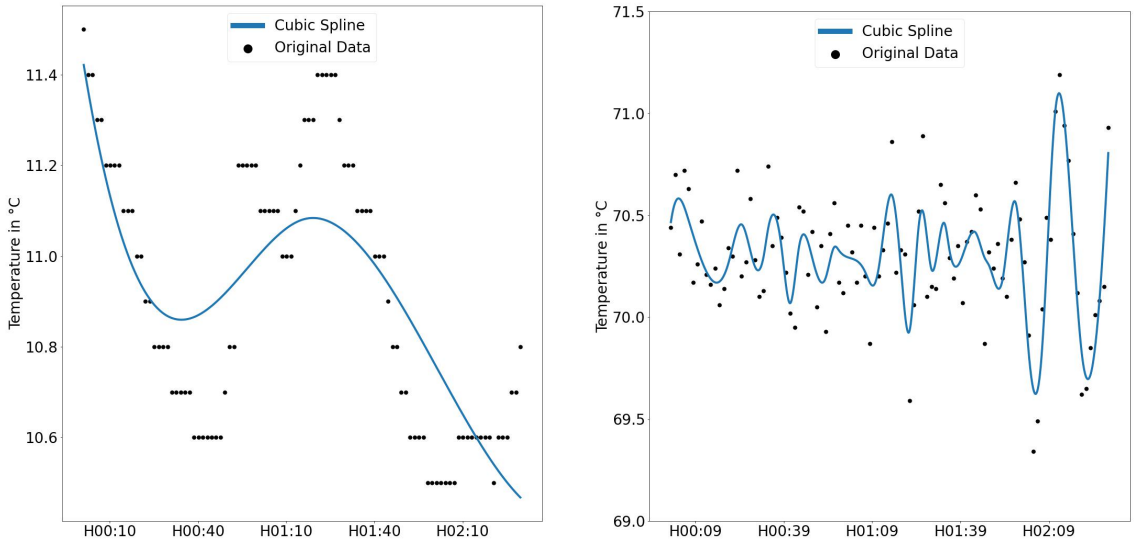
Figure 3.29: Linear and spline interpolation on a snippet of the `GOERLOESE_BY_TF` time series. Interpolated curves in blue and orange, while the original datapoints are scattered across in black.



(a) Linear interpolation

(b) Cubic spline interpolation

Figure 3.30: Interpolation of 100 measurements of the Hillerød outdoor temperature, which is measured in discrete 0.1°C-steps. Interpolated curves in blue, while the original datapoints are scattered across in black.



(a) Outdoor Temperature

(b) `GOERLOESE_BY_TF`

Figure 3.31: Smoothed cubic spline interpolation of 100 measurements of two different time series, with smoothing factor `s=3`.

a continuous one. The linear interpolation seems to make more sense in this light: Where the original value stays constant, the linear interpolation stays constant as well, while the cubic spline reminds of sawtooth waves.

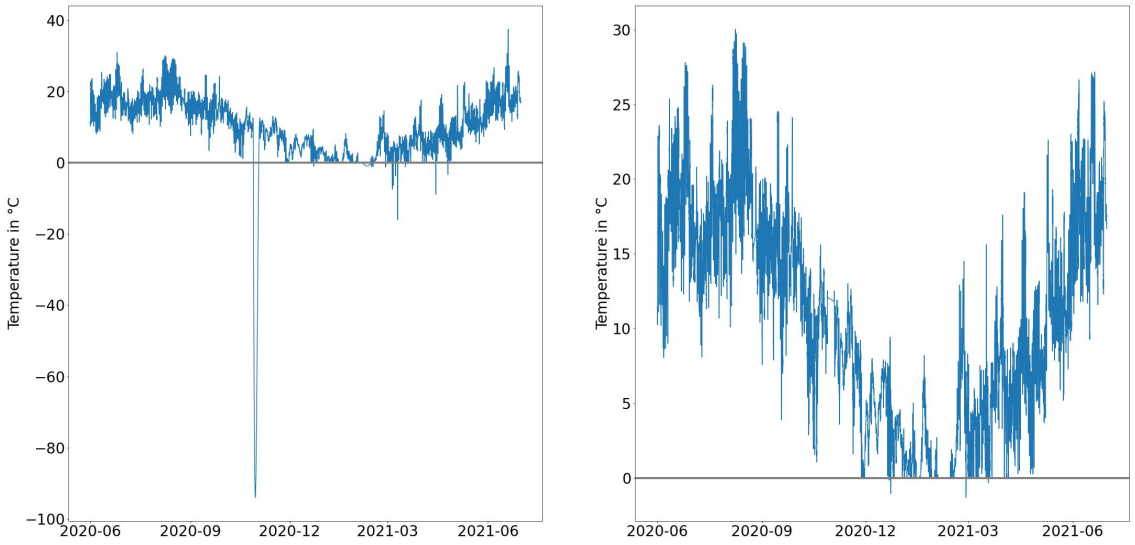
In both cases, we can also re-discretize the interpolated function again, simply by rounding any interpolated value to the nearest multiple of 0.1°C. This is simple in this case, since we already know that this particular time series provides data in multiples of 0.1°C. Generalizing this to all our time series is much less trivial, since we neither know which time series store values only in a discrete range, nor what the difference between separate values is. Note, that we cannot change the function directly due to its design, we can just round the result of the evaluation.

Instead of discretizing the results again so that they match the original function, we can also try to generalize them - the real outdoor temperature is also not discrete-valued. We can do this, for example, by smoothing our spline interpolation. There is also a SciPy function available for that[33]. In Figure 3.31, we can see the resulting interpolation for our previous data.

On the one hand, the outdoor temperature graph looks now much nicer and, well, smoother. On the other hand, even with low choices for the smoothing factor, i.e. `s=2` or `s=3`, the smoothing gets rid of a very large amount of variance in the data. For example, it completely withholds evidence of a drop and rise in temperature of roughly half of a degree over the course of approximately less than half an hour.

Similarly, it neglects some amplitudes in the supply temperature measurements. Additionally, the `UnivariateSpline` is **very** slow in computing the interpolation. Therefore, we

[33] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.UnivariateSpline.html`



(a) If we interpolate between all data points, we create extreme outliers.

(b) If we only interpolate gaps that are shorter than 2 hours, we avoid the creation of outliers.

Figure 3.32: Cubic spline interpolation of the outdoor temperature.

will not pursue this approach further.

We cannot use the spline interpolation without caveats, though: As we can see in Figure 3.32, it can create outliers, as the outdoor measurement that contains only positive values is suddenly interpolated to values close to -100°C. We can remedy this to a large extent by only interpolating between values that are at most 2 hours apart from each other.

3.3.2 Polynomial Regression

Somewhat related in the idea behind it is the approach of linear or polynomial regression. Here, we also try to find a mapping f from inputs x_i to targets y_i such that $\forall i \in \{1, \ldots, N\}$: $y_i \approx f(x_i)$.

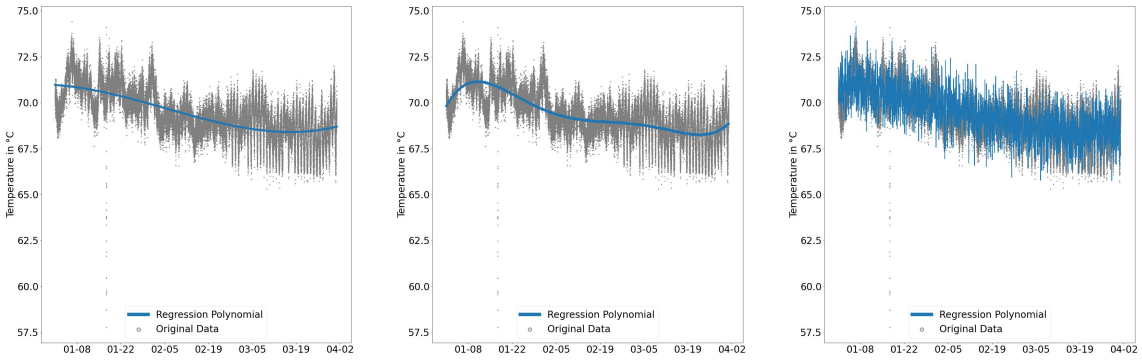
We assume that $y_i = f(x_i) + \epsilon_i$, where the ϵ_i is normally distributed noise with a mean of 0. We try to find this function f by minimizing the error between our model and the available data, usually by minimizing the least squares error.

There is already a NumPy method that does this for us[34]. However, a trial run on one of our favorite datasets does not really yield great results, as we can see in Figure 3.33.

Standard Approach

First, we try to fit a polynomial to our data. It turns out, that one of degree 3 is the best we can do for this data, with the coefficient matrix of the polynomial still having full rank. We can get a slightly better overall fit with a polynomial of degree 6, if we lower the regularization strength, but we still end up with a great oversimplification of the data.

[34] `https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html`



(a) The best fit polynomial of degree 3.

(b) The best fit polynomial of degree 6, with very low regularization.

(c) Best fit polynomial of degree 3, but with added Gaussian noise.

Figure 3.33: Polynomial Regression for the `GOERLOESE_BY_TF` measurement, with the original data in grey.

This comes as no surprise, since regression explicitly aims for the *removal* of noise from the data. This leaves us with two options to reconstruct missing data. Either, we use the values from the regression curve, thus removing all noise from the data. Or we add Gaussian noise again, as we did in Figure 3.33, which preserves the value range of the data. We observe that with both options, our reconstructed time series will not have the same values as our original one, even at the same timestamps. That means, even if no data is missing at all, we would (unnecessarily) alter the time series to some non-trivial extent.

Another difficulty: the result of the regression depends on two parameters, namely the degree of the target polynomial and the regularization strength. But always choosing the same parameters will yield results of different quality for different time series. We could try to remedy this by selecting the degree of the polynomial dynamically, for instance by always starting with degree 1 and then increasing it step by step, until the polynomial's coefficient matrix does not have full rank anymore. Nevertheless, the "pure" regression approach seems to be inferior to the spline interpolation for our application in general.

Slicing the Data

Having said this, the polynomial regression can be a very useful tool with some modifications. We have two main motivations for this.

First, when we checked more time series to evaluate our interpolation and regression approaches, we found that the cubic spline interpolation can add a significant amount of outliers to our data - even when we only interpolate shorter gaps. Similar to Subsection 3.2.7, we checked for random time series from all three datasets, whether our interpolation and regression mechanisms yield meaningful results. Coincidentally, we stumbled across the power measurement from Figure 3.24b again.

As we can see in Figure 3.34, the spline interpolation adds outliers to the dataset that have not been there before. While the power production fluctuates somewhere between 0 and 2 MW in the original data, our spline interpolation suddenly yields values of up to

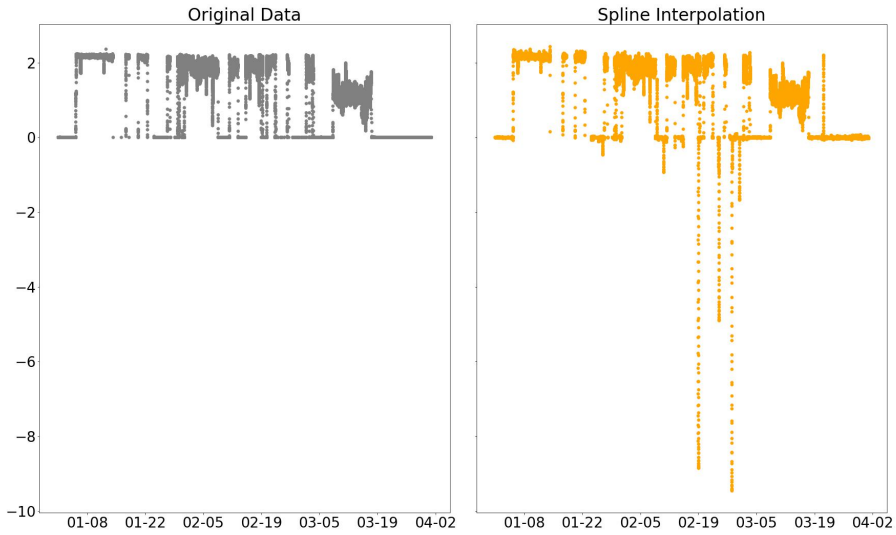


Figure 3.34: Original data (left) and its cubic spline interpolation (right) for the `FV_VP1_2_QM1` measurement. The spline interpolation adds outliers to the dataset.

negative 10 MW. It is not entirely clear, whether this measurement contains the power that is produced or the power that is needed to produce energy - but in both of those cases, we can safely assume that these negative values are false.

Second, we have determined at the beginning that there is an upper limit to the length of a gap in the data that still allows us to reconstruct data in a meaningful way. We have deliberately set this length to 2 hours, although the following considerations are independent of the concrete length we choose.

If we interpolate longer gaps as well, linear interpolation estimates of the shorter gaps are not affected by this at all. For the cubic spline interpolation, there may be some influence on the results, but this is still mainly noticeable right next to the edges of the longer gaps.

In contrast, because polynomial regression optimizes the least-squares error over all data, it does get influenced by the lack of data in a large gap. We can consider this as an advantage or a disadvantage: the computed regression polynomial can be somewhat imprecise in the gap, but we still have estimates that are based on real observations from other parts of the time series - presumably yielding better estimates than the technique of guessing at random.

Still, we might get even better estimates for all shorter gaps, i.e. those that we actually want to consider and reconstruct, if we split the time series into shorter slices that only contain gaps that are shorter than 2 hours, as we have proposed at the beginning of this chapter. Then, we can reconstruct the data in each of these slices separately and put them back together afterwards. That means, the gap will remain in place in our reconstructed data, while we have replaced everything else with our newly created time series (see Figure 3.35 for a visual approximation on how this is supposed to work).

Let us consider a real example from the Trefor dataset. The time series in Figure 3.36 is discontinuous for about half a day in mid-March. Therefore, we split it into two time series and perform polynomial regression with a polynomial of degree 5 on each part separately. We can see in the figure that the results of this approach are slightly better than with

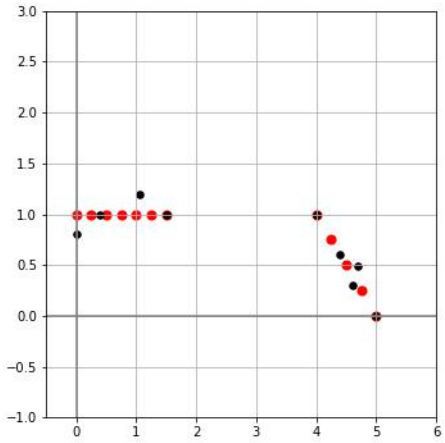


Figure 3.35: Artificial example to clarify how the reconstruction mechanism is supposed to work after splitting up the time series across large gaps. Original data in black, reconstructed data (with a fixed time distance) in red. Note, that the reconstruction yields separate results for the slices of the time series.

polynomial regression on the whole dataset - especially for the part that comes after the gap.

This gives us the idea to split up the time series into slices in general and to perform polynomial regression on each of those slices, instead of only performing a single regression. We find, that this can be a vast improvement over our initial regression approach. For example, let us consider the `KRA15VEKS_TR` return temperature from the Hillerød dataset.

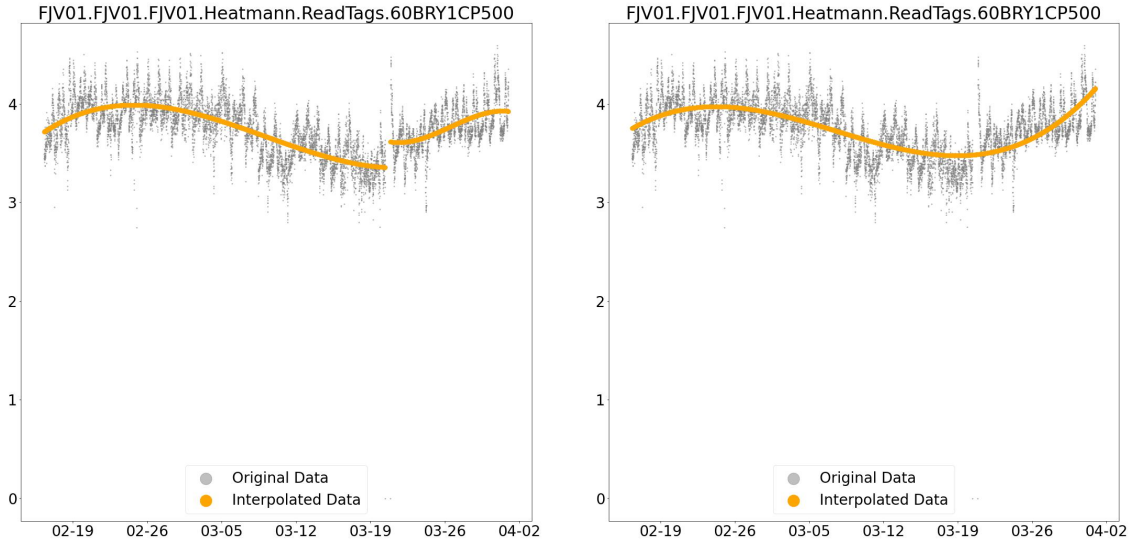
If we perform polynomial regression for the whole three-month long time series as a whole, we can see in Figure 3.37a that the resulting polynomial does not fit the data satisfyingly virtually anywhere. But if we split the time series into pieces of 6 hours length each, and perform polynomial regression separately on each of these pieces, our regression model fits the original data much better, apart from a few overshooting lines. We can also try to fit a polynomial with a lower degree, but as we can see in Figure 3.37c this does not get rid of the overshooting lines, but introduces some imprecision instead.

In this case, we can argue, that we actually prefer **less** generalization anyway: we try to reconstruct one time series specifically, so overfitting is desired here! We do not aim to find a generalized model for all return temperature time series, because we can fit separate polynomials to those as well - we have already seen in Figure 3.1 that generalizations will be difficult or impossible.

Instead of comparing the approaches only visually, we can also do this numerically. For that, we compute the regression polynomial(s) as before, but we do not evaluate them at our equidistant, self-chosen timestamps, but at those of the original data. For the eye-minded reader, we plot those differences in Figure 3.37d. While the mean difference is close to 0 for both our standard and our slicing approach[35], the standard deviation greatly differs: while it is only about 0.83 for our slicing approach with multiple polynomials, it is a multiple of that for the estimation with just a single polynomial, namely roughly 12.48.

In Figure 3.37d we can see that most of this deviation comes from just a few spikes, which

[35] -0.370162 for the regular and -0.005672 for the slicing approach, to be exact.



(a) We split the time series and performed polynomial regression on the two parts separately.

(b) We have not split the time series at the gap, but performed a single polynomial regression on all of the data at once.

Figure 3.36: Time series from Trefor which includes a gap that is longer than 2 hours. Original data in grey, regression polynomials in orange.

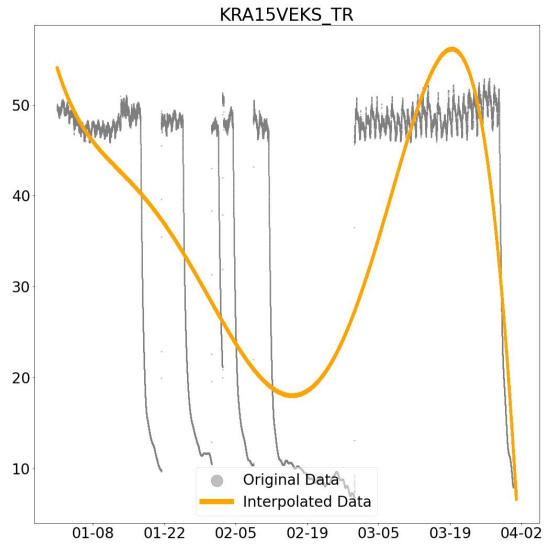
we already have noticed in our original plot. Numerically speaking, while only about 6.5% of our estimations differ by an absolute value of 1 or less from the original data, we can raise this share to a staggering 98.73% by choosing multiple polynomials[36]. This yields a root mean squared error of approximately 0.825, which is somewhere in the ballpark of the results reported by [JLH+20].

Including Outlier Detection

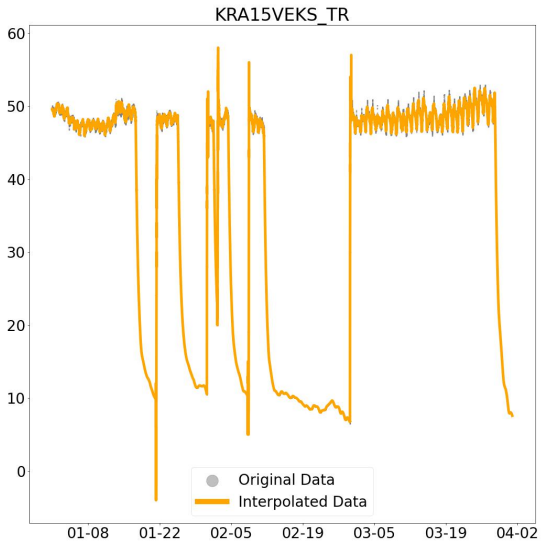
We can now ask ourselves, whether we can improve the results further by applying our anomaly detection mechanism from Section 3.2. Somewhat unsurprisingly, removing the extreme outliers with our Algorithm 2 will drastically improve the results - which is why we were looking for ways to remove them in the first place. If we do that for the Hillerød outdoor temperature measurements, we end up with a root mean squared error of about 0.34.

Whether everything on top of that, i.e. the outlier identification with the `IsolationForest`, influences the results significantly is difficult to say. For multiple time series, we trained one polynomial regression models on the original data and one on the same data, but with the outliers removed beforehand. We then computed and compared the estimations on both the original timestamps, as well as only on timestamps of non-outlier-identified values. In all cases, the root mean squared error were almost identical, usually only starting to diverge at the second decimal place.

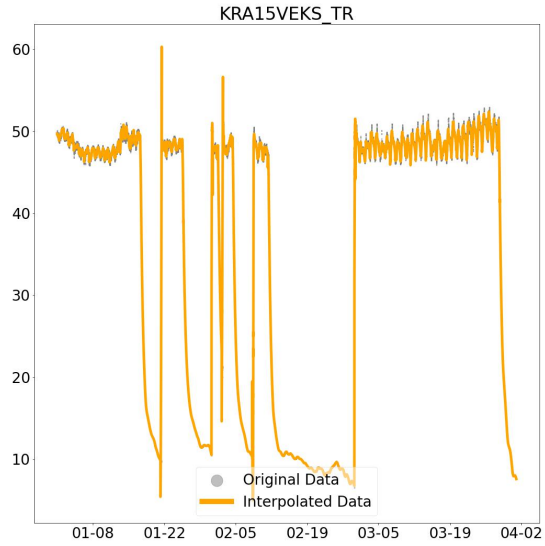
[36] see also `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.percentileofscore.html`



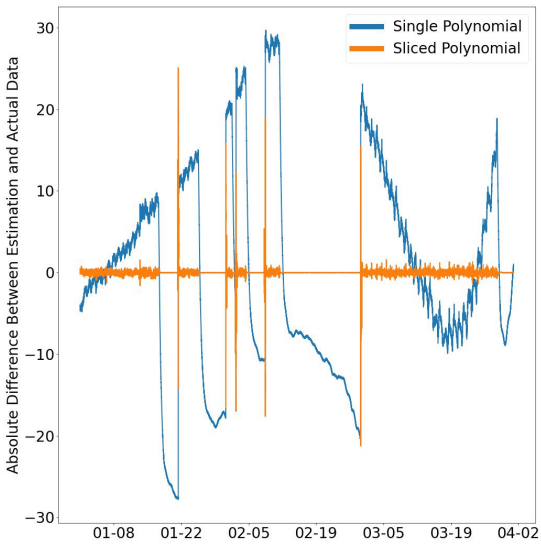
(a) We compute one regression polynomial for the whole 3-month period, which yields a very bad fit for the data.



(b) We split the time series into 6-hour slices and compute a regression polynomial for each of them. This fits the data perfectly.



(c) Polynomial regression on 6-hour slices as well, but with polynomials of degree 3.



(d) Difference between our estimations and actual data, if we use a single (blue) and multiple (orange) polynomials.

Figure 3.37: Return temperature measurement from Hillerød. Original data in grey, regression polynomials in orange. In the first two cases, we fit a regression polynomial of degree 9 with `rcond=1e-20`.

3.3.3 Correlation

Given that we have multiple time series of the same kind over the same time period in each dataset, we can try to impute multiple time series at once. This way, we might be able to derive values from similar time series. As long as not all sensors suffer from a failure at the same time, the available data from one series might give us a good estimate for the missing data of another time series.

An approach to this would be to join multiple time series into a single matrix. Even if no values are missing from each of them, the timestamps will differ between the time series to some extent. Thus, if we use a full outer join with the timestamps as predicate, we will end up with gaps in our matrix. Then, we can try to impute these gaps, for example as described in `https://scikit-learn.org/stable/modules/impute.html`.

This might work, if similar time series are correlated. Unfortunately, this is not the case - or at least not to some useful extent, as we will se shortly.

From Figure 3.1 and Figure 3.2, we can already suspect the correlation among supply temperature measurements to be quite low, but higher for return temperature measurements. Figure 3.38 shows the pairwise absolute correlation[37] between selected temperature measurements from the Hillerød dataset. Darker colors connote low correlation, brighter colors connote high correlation. Unfortunately, this figure is quite dark overall.

A look at the values confirms that the supply temperature measurements are very uncorrelated to the rest of the data. The maximum value we can find there is 0.5 between `KGV_TF` and `GOERLOESE_BY_TF`, but that is about it. For the return temperatures, we can find some more correlation, with a maximum of 0.74 between `FRE_TR` and `KGV_TR`, but the overall picture is not very promising either. Overall, we can compute a mean (absolute) correlation of 0.199, with a standard deviation of 0.171.

That is not the say that all measurements are as uncorrelated. If we look at all measurements from two different stations in Figure 3.39, we can see that some time series are actually highly correlated.

In general, we find that power (`EFFEKT`) and flow measurements are highly correlated - within and across sensor stations. Apart from that, we cannot detect a meaningful pattern here in any way that we could exploit. For example, while the `GOERLOESE_BY_TF` supply temperature is correlated with the `ELM_EFF` power measurement, this does not work the other way round. If we add more stations and time series to the graph, this also adds more to the confusion. Thus, we cannot make use of correlation between time series in a methodical way.

Nevertheless, we tried our previously mentioned approach of matrix imputation with the available SciKit models[38]. As expected, this did not yield any meaningful results so far, although the `IterativeImputer` at least produced bad results, while the `KNNImputer` did not.

Autocorrelation

District heating networks are inherently periodical: more heat is needed during winter, less heat is needed during summer. Therefore, another interesting thing to consider is not only

[37] `https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html`
[38] `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.impute`

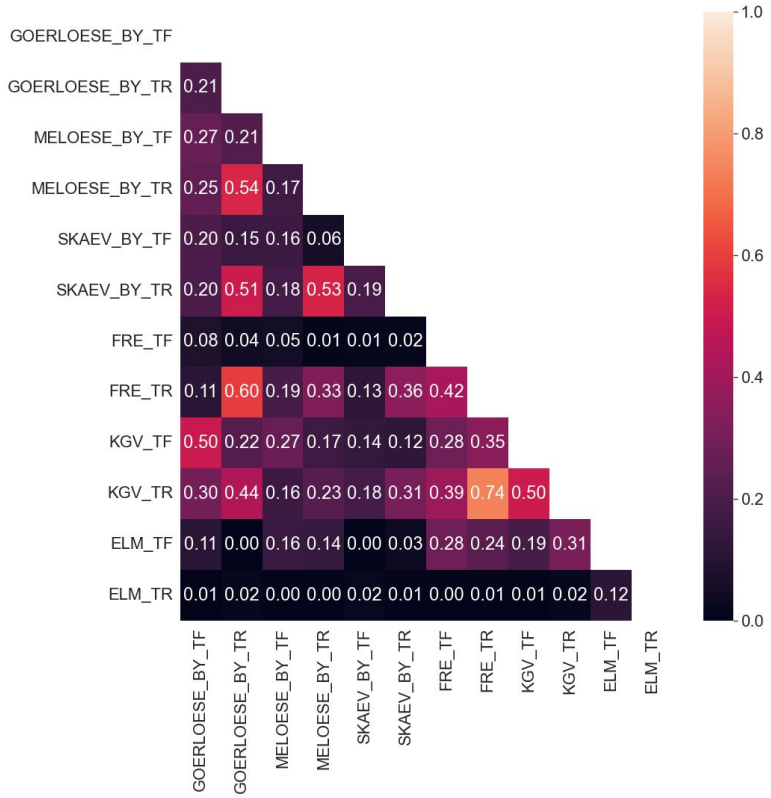


Figure 3.38: Absolute correlation values between some supply (`TF`) and return (`TR`) temperature measurements from the Hillerød dataset. The brighter the color, the larger the correlation between the two time series.

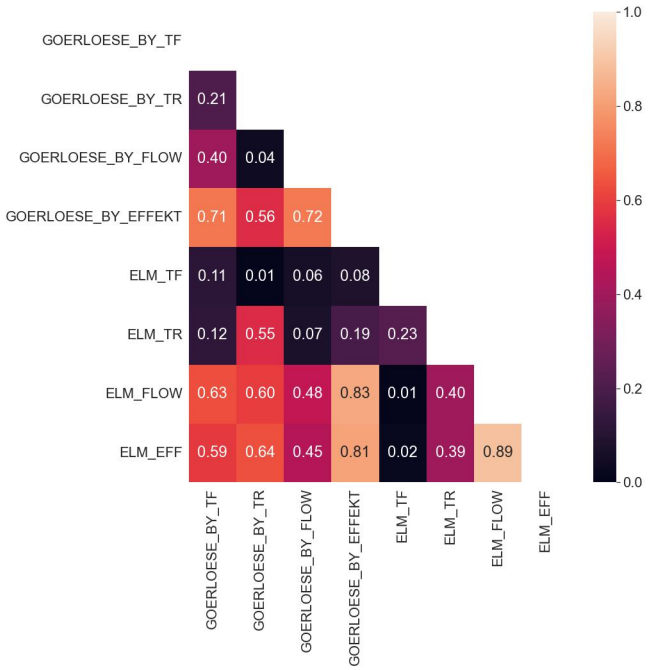


Figure 3.39: Absolute correlation values between all `GOERLOESE` and `ELM` measurements from the Hillerød dataset, after removing extreme outliers. The brighter the color, the larger the correlation between the two time series.

the correlation between different time series, but also the correlation within a time series, i.e. autocorrelation.

In particular, we expect a time series X during the year 2020 to be correlated to the time series X during the year 2021. And for a few selected time series, this was actually the case. However, examining this in more detail is necessary, because the timestamps of the 2020 data are not the same as those of the 2021 data (apart from the year, which is obviously not the same anyway). Therefore, a thorough mapping between timestamps has to be developed first.

Moreover, there is not much periodical data available yet. The time series start around the beginning of 2020 (see Subsection 3.1.2), so we have at most two full periods available - arguably not enough to make any meaningful statements on this.

3.3.4 Variational Autoencoders

In the beginning, we hoped that we could end up with some kind of neural network, where we could just feed our incomplete time series into, which then does its magic and consequently spits out a perfectly reconstructed time series. In particular, Variational Autoencoders came to mind immediately. In an unsupervised fashion, they can learn a lower-dimensional latent representation from the training data that is robust to gaussian noise[39]. Since the latent vectors are forced to closely follow a gaussian distribution, the network generalizes and can

[39] `https://kvfrans.com/variational-autoencoders-explained/`

be used to generate new data.

We have already trained a VAE on the MNIST dataset and successfully used it to generate new images of drawn digits. Therefore, a VAE seemed us to be a viable option to reconstruct our missing data points - even before starting to research for related work and realizing that this approach has gained popularity over the last few years - for anomaly detection even more so than for time series reconstruction.

Using a VAE for reconstructing data requires us to train it first - and its performance on the reconstruction tasks will succeed or fail with the quality of its training data. Just as an example, before Liguori et al. start their model development to reconstruct data, they conduct a data preprocessing step that includes the detection of outliers and the removal of time series with longer periods of missing data - apparently removing about 75% of their daily observations [LMD+21]. While there can be disparities in opinions what actually makes training data to be of high quality, we can be certain that most of our data is not high quality training data. As we have discussed in Section 3.1, there are multi-day gaps in all time series of one dataset, value thresholds and extreme outliers in another - the data preprocessing would either take a significant amount of time or remove a very significant amount of data, or probably both.

That is also why we were looking for additional data sources in Subsection 3.1.5, mostly to no avail. In this case in particular, instead of choosing a different and more reliable data source, we can also look whether we can choose a pre-trained model instead. But while pre-trained models for time series classification and forecasting are not unheard of [YLL+22, SK19], this does not seem to be the case for time series reconstruction.

4 Conclusion

We have begun this thesis with some philosophical contemplations, what actually constitutes anomalous or missing data - to repair something, we need to know what is broken in the first place. For this purpose, we have also thoroughly examined our available data. Subsequently, we have examined multiple techniques and different approaches on data cleaning and anomaly detection. We have looked at what these techniques do in general, and what that means for our data. We applied this same approach to reconstruction mechanisms, in particular interpolation and regression, and deduced necessary modifications from that. We have also shed some light on other available techniques, as matrix imputation and Variational Autoencoders, and what would be necessary for us to use them successfully.

We have already stated in the introduction that no anomaly detection and no time series reconstruction method will be a one-size-fits-all. As our own data is very heterogenous, we have seen that there might not even be the one method for all of *our* data. Nevertheless, we have found some techniques that fit at least most of our data well. Our own algorithm to detect extreme outliers from Subsection 3.2.6 is simple, yet does what it is supposed to do more reliable than state-of-the-art implementations. The isolation forest from Subsection 3.2.4 is mostly a great addition to remove most obvious outliers from our dataset, although its influence on the later reconstruction results is unfortunately quite small. On a much better note, we have seen that our sliced approach to polynomial regression is actually a very good way to reconstruct our missing data, because it stays very close to the original data, while still providing meaningful results in between.

At the end of this project, there are still loose threads that we could tie up in the future. We have already combined different mechanisms for the anomaly detection. We could try to repeat this for the data reconstruction. For example, we can try to use regression polynomials of different degrees and combine the results. We could also try to process our data manually into great training data that we can use for Variational Autoencoders. Another interesting path seems to be the correlation between and within time series, and whether we can use matrix imputation for those time series that are correlated, or on different periods of the same time series.

List of Figures

Bibliography

[AC15] Jinwon An and Sungzoon Cho. Variational Autoencoder based Anomaly Detection using Reconstruction Probability, 2015.

[Ass17] Jan Paul Assendorp. *Deep learning for anomaly detection in multivariate time series data*. Master's thesis, Hamburg University of Applied Sciences, Hamburg, Germany, September 2017.

[AYMTF20] E Afrifa-Yamoah, U. A. Mueller, S. M. Taylor, and A. J. Fisher. Missing data imputation of high-resolution temporal climate time series data. *Meteorological Applications*, 27(1):e1873, 2020.

[Ban21] European Central Bank. *Seventh report on card fraud: October 2021*. October 2021.

[Ben91] Atli Benonysson. *Dynamic modelling and operational optimization of district heating systems*. PhD thesis, Danmarks Tekniske Universitet, Kongens Lyngby, DK, 1991.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying Density-Based Local Outliers. *ACM SIGMOD Record*, 29(2):93–104, May 2000.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3), July 2009.

[CGL+20] Xintao Chai, Hanming Gu, Feng Li, Hongyou Duan, Xiaobo Hu, and Kai Lin. Deep learning for irregularly and regularly missing data reconstruction. *Scientific Reports*, 10(1), 2020.

[CLKB19] Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences of the United States of America*, 116(45):22445–22451, November 2019.

[con22] Wikipedia contributors. Anomaly detection, 2022. [Online; accessed 29-April-2022].

[CZS+16] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30:891–927, 2016.

[Dan17] Danish Energy Agency. *Regulation and planning of district heating in Denmark*, June 2017.

[DF13] Zhiguo Ding and Minrui Fei. An Anomaly Detection Approach Based on Isolation Forest Algorithm for Streaming Data using Sliding Window. *IFAC Proceedings Volumes*, 46(20):12–17, 2013. 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013.

[Die17] Felix Dietrich. *Data-Driven Surrogate Models for Dynamical Systems*. Dissertation, Technische Universität München, München, D, 2017.

[FW20] Chenguang Fang and Chen Wang. Time Series Data Imputation: A Survey on Deep Learning Approaches. *CoRR*, abs/2011.11347, 2020.

[HK11] Stephen Hawking and H. Kober. *Eine kurze Geschichte der Zeit*. Rowohlt-Taschenbuch-Verlag, 2011.

[JLH+20] Hicham Johra, Daniel Leiria, Per Heiselberg, Anna Marszal-Pomianowska, and Torben Tvedebrink. Treatment and analysis of smart energy meter data from a cluster of buildings connected to district heating: A Danish case. In J. Kurnitski and T. Kalamees, editors, *Proceedings of the 12th Nordic Symposium on Building Physics (NSB 2020)*, E3S Web of Conferences, France, June 2020. EDP Sciences.

[Kan13] Hyun Kang. The prevention and handling of the missing data. *Korean Journal of Anesthesiology*, 64(5):402–406, 2013.

[KATCM20] Mourad Khayati, Ines Arous, Zakhar Tymchenko, and Philippe Cudré-Mauroux. ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams. *Proc. VLDB Endow.*, 14(3):294–306, November 2020.

[KCC+22] Siwon Kim, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. Towards a Rigorous Evaluation of Time-series Anomaly Detection. *CoRR*, abs/2109.05257, January 2022.

[KLT04] Eamonn Keogh, Jessica Lin, and Wagner Truppel. Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research. *Knowledge and Information Systems*, 8:154–177, 2004.

[KLTCM20] Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. Mind the Gap: An Experimental Evaluation of Imputation of Missing Values Techniques in Time Series. *Proceedings of the VLDB Endowment*, 13(5):768–782, January 2020.

[LAC17] Mathieu Lepot, Jean-Baptiste Aubin, and François H.L.R. Clemens. Interpolation in Time Series: An Introductive Overview of Existing Methods, Their Performance Criteria and Uncertainty Assessment. *Water*, 9(10), 2017.

[LGG08] Hatem Ltaief, Edgar Gabriel, and Marc Garbey. Fault tolerant algorithms for heat transfer problems. *Journal of Parallel and Distributed Computing*, 68(5):663–677, 2008.

[LKK17] Seungjoon Lee, Ioannis G. Kevrekidis, and George Em Karniadakis. A resilient and efficient CFD framework: Statistical learning tools for multi-fidelity and heterogeneous information fusion. *Journal of Computational Physics*, 344:516–533, 2017.

[LMD+21] Antonio Liguori, Romana Markovic, Thi Thu Ha Dam, Jérôme Frisch, Christoph van Treeck, and Francesco Causone. Indoor environment data time-series reconstruction using autoencoder neural networks. *Building and Environment*, 191, 2021.

[LR02] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, New York, NY, 2002.

[LTZ08] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[LZH+22] Zheng Li, Yue Zhao, Xiyang Hu, Nicola Botta, Cezar Ionescu, and George H. Chen. ECOD: Unsupervised Outlier Detection Using Empirical Cumulative Distribution Functions. *CoRR*, abs/2201.00382, March 2022.

[MBB17] Steffen Moritz and Thomas Bartz-Beielstein. imputeTS: Time Series Missing Value Imputation in R. *The R Journal*, 9(1):207–218, 2017.

[Mei02] Erik Meijering. A Chronology of Interpolation: From Ancient Astronomy to Modern Signal and Image Processing. *Proceedings of the IEEE*, 90(3):319–342, 2002.

[MM99] L Mitas and H Mitasova. Spatial Interpolation. In P. Longley, M.F. Goodchild, D.J. Maguire, and D.W. Rhind, editors, *Geographical Information Systems: Principles, Techniques, Management and Applications*, chapter 34, pages 481–492. Wiley, 1999.

[NHR12] Zineb Noumir, Paul Honeine, and Cédue Richard. On simple one-class classification methods. In *2012 IEEE International Symposium on Information Theory Proceedings*, pages 2022–2026, 2012.

[PLB+99] H. Pálsson, Helge V. Larsen, Benny Bøhm, H.F. Ravn, and J. Zhou. *Equivalent models of district heating systems for on-line minimization of operational costs of the complete district heating system*. DTU-ET-ES-99-03. Technical University of Denmark. Department of Energy Engineering, 1999.

[PS18] João Pereira and Margarida Silveira. Unsupervised Anomaly Detection in Energy Time Series Data Using Variational Recurrent Autoencoders with Attention. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1275–1282, 2018.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron

Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Rub76] Donald B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, December 1976.

[Sch18] Christiane Schlötzer. Griechischer Chefstatistiker verurteilt - wegen Ehrlichkeit. *Süddeutsche Zeitung*, June 2018.

[Sed21] Philipp Sedlmeier. Extending the Calibration of the Transition Model for Calculating Solar Radiation Values. TUM Department of Aerospace and Geodesy, September 2021.

[Sha48] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[Sha19] Janelle Shane. *You Look Like A Thing And I Love You.* Wilfire, Headline Publishing Group, London, UK, 2019.

[SK19] Alaa Sagheer and Mostafa Kotb. Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems. *Scientific Reports*, 9(1), 2019.

[Wil20] Tessa Wilkie. Missing Data: Introducing the Missingness Mechanism. https://www.lancaster.ac.uk/stor-i-student-sites/tessa-wilkie/2020/04/29/missing-data-part-1-introducing-the-missingness-mechanism/, April 2020.

[WLV+09] Joakim Widén, Magdalena Lundh, Iana Vassileva, Erik Dahlquist, Kajsa Ellegård, and Ewa Wäckelgård. Constructing load profiles for household electricity and hot water from time-use data - Modelling approach and validation. *Energy and Buildings*, 41(7):753–768, 2009.

[WW20] Xi Wang and Chen Wang. Time Series Data Cleaning: A Survey. *IEEE Access*, 8:1866–1881, 2020.

[YLL+22] Yuan Yuan, Lei Lin, Qingshan Liu, Renlong Hang, and Zeng-Guang Zhou. SITS-Former: A pre-trained spatio-spectral-temporal representation model for Sentinel-2 time series classification. *International Journal of Applied Earth Observation and Geoinformation*, 106, 2022.

[ZNL19] Yue Zhao, Zain Nasrullah, and Zheng Li. PyOD: A Python Toolbox for Scalable Outlier Detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[Zuc20] Eugenio Zuccarelli. The dos and don'ts of imputation. *Towards Data Science*, November 2020.