Márton Kovács
18 March 2024

User Manual

Time Series Data Cleaning User Interface

1 Table of Contents

2 Introduction

2.1 The Purpose of the User Interface

The user interface that this document aims to serve as a manual for was made for a master's thesis in the 2024. The aim of the program is to aid the user in the cleaning of time series data, without the need for the user to be knowledgeable in the area of data science.

2.2 What is Time Series Data?

Time series data consists of floating-point numbers with a constant sampling frequency, meaning that the time-difference between two measurement points is the same for each subsequent point. Aside from the constraint on sampling rate, time series data should have smooth changes between any two given points, and may have a periodic behaviour as well, meaning that there is a repeating behaviour over a certain number of measurement points (i.e., a daily periodicity in household power consumption, with power consumption being low during the day and high in the evening).

The input time series data always consists of a timestamp column, and multiple floating-point number columns, henceforth called features.

Based on the above, time series data can always be decomposed into a trend and a residual component, and into multiple seasonal components, depending on the number of relevant periodicities in the data, in a manner similar to the one shown on Figure 2.1. By summing these components together, the original signal can be rebuilt. The trend component should consist of a smooth change with no periodic behaviour, the seasonal component should have a repeating behaviour and an average value of zero, and the residual component or the noise should also have an average value of zero, however there should be no repeating behaviour in it, it ought to be random.

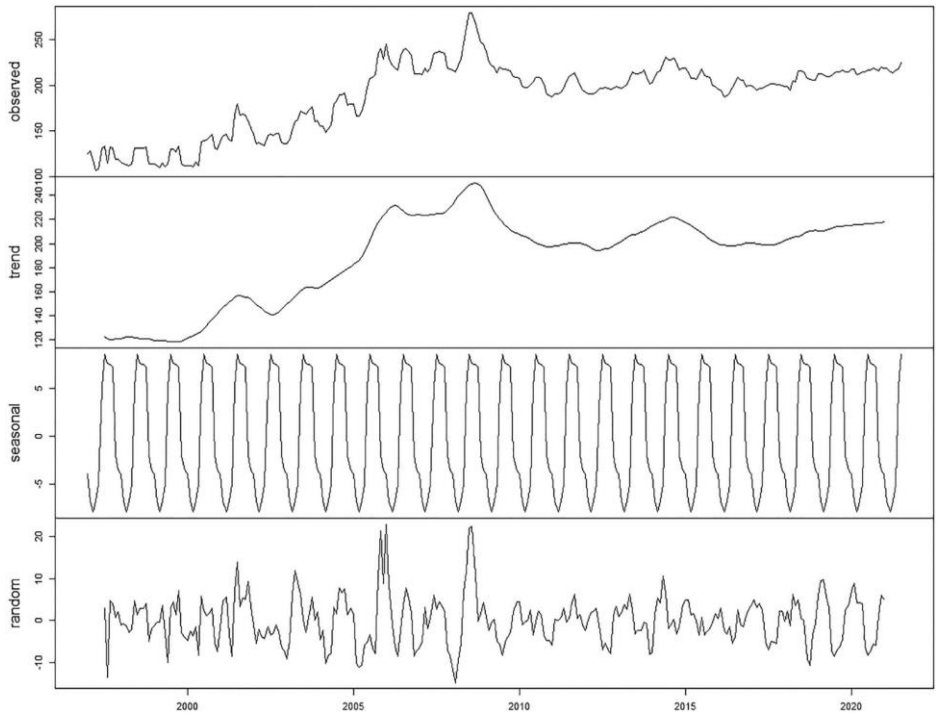


Figure 2.1: Electricity and piped gas costs for consumers in Texas [1]

Based on literature, three kinds of outliers can be defined in the context of time series data. These are point outliers, wherein a single point is an outlier (Figure 2.2), continuous outliers, wherein after a certain point a long-term behaviour shift occurs (Figure 2.3), and inter-series outliers, wherein the behaviour in one time series feature does not correspond to the behaviour in another. It should be noted that each of the aforementioned components has different prominent error types.

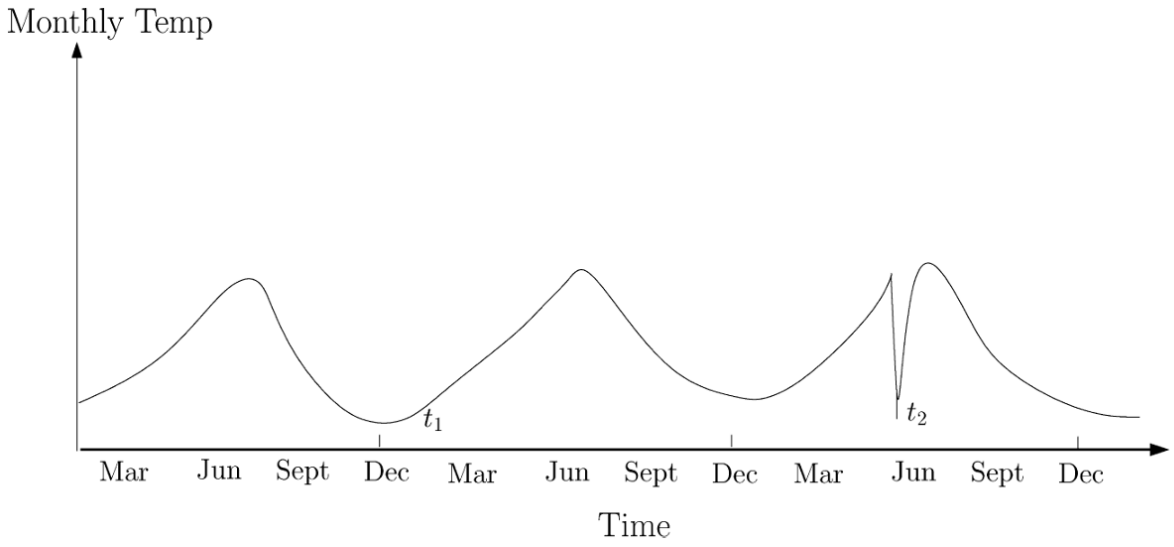


Figure 2.2: Point anomaly in a temperature time-series [2]

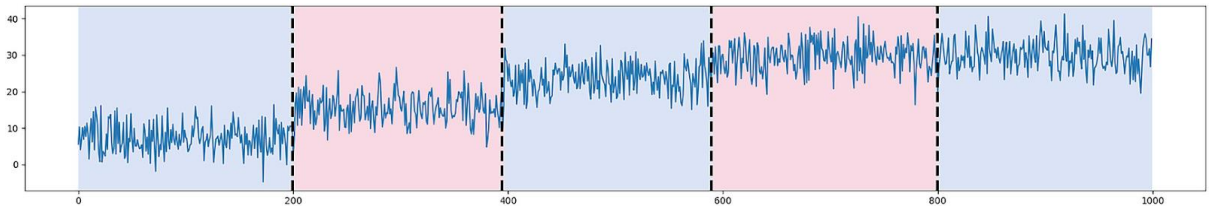


Figure 2.3: Long-term behavior shifts in time-series data [1]

A final key concept from literature worth considering is the separation of anomalies into events and errors. An anomaly can be any form of outlier described above, and events are separated from errors based on anomaly context, which in the case of the user interface is defined by looking at relationships between series. In short, this means that an outlier is only flagged as an error if a similar kind of outlier does not occur in any other feature (i.e., if I have a voltage and a current signal, and I detect an outlier in only voltage, it is deemed an error (meaning for example a sensor error), whereas if both are detected as an outlier in the same timestamp, it is flagged as an event (e.g., if my power consumption increases, both of these features will increase)).

2.3 An Overview of the User Interface

The ultimate goal of the python-based user interface (UI) is to produce a version of the original data where either errors are identified at each timestamp, or they are corrected at each timestamp. As each component has different prominent error types, outlier detection requires the breakdown of the signal to its components through data transformation. Since the data transformation method used in the user interface does not accept missing values, missing values ought to be imputed beforehand, which requires knowledge of the relevant periodicities in the data, and naturally the reading of the data.

A flowchart describing this methodology can be seen on Figure 2.4, wherein the modules of the user interface are put in order in the middle of the figure. The figure also shows the output plots for each module on the top, which may serve as a control for recalculating certain steps of the UI (e.g., since missing value imputation depends on the results of periodicity estimation, the parameters of periodicity estimation may be adjusted after observing the missing value imputation). Aside from output plots, output data is also shown on the figure on the bottom. Here it can be seen that aside from error detection and rectification results, the user interface also facilitates the imputation of missing values, which can find its use in periodic data.

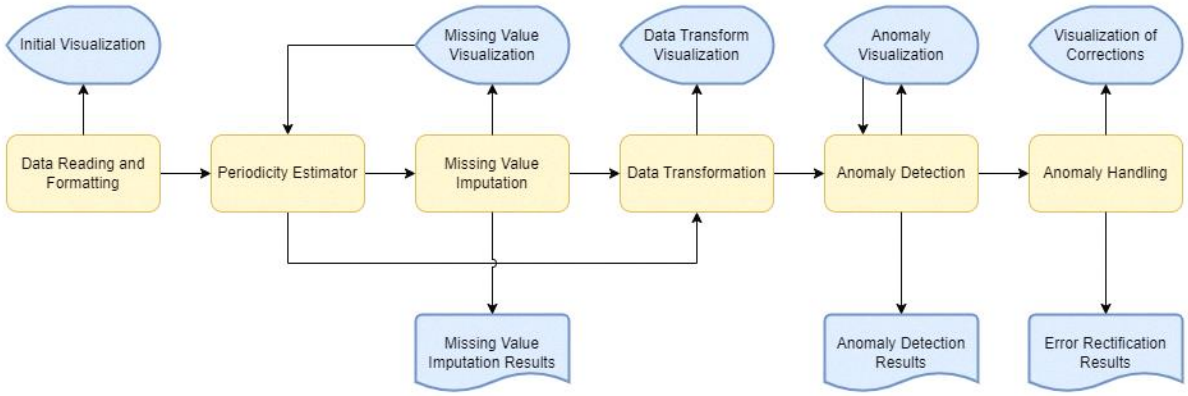


Figure 2.4: The flowchart of the data cleaning user interface

3 The Data Input Module

3.1 Overview

The purpose of the module discussed in the following paragraphs is to read the data to be cleaned, and to format the data in a manner that calculations can be done on it. Aside from these tasks, the periodicity estimator is also in this module, which can tell the user of all relevant periodicities detected in the data and is required for future calculations. Finally, the initial visualization is also found in this module, where the user can take a look at the formatted data.

3.2 Data Reading and Formatting

The first page of the data input module is file reading, as shown on Figure 3.1. Here, by clicking the browse button, a csv format file can be selected (not necessarily a .csv file, can also be .txt for example). Selecting a file will upload it to the user interface. In the next card, the csv separator can be set, which is the text that separates the columns within a csv. Within this page, only 50 rows of the read data are shown in a table as to minimize rendering time, however, the user can parse through the data with the slider on the right, rendering the data from a new index. This rendering does not affect the read data itself and is only for visualization purposes.

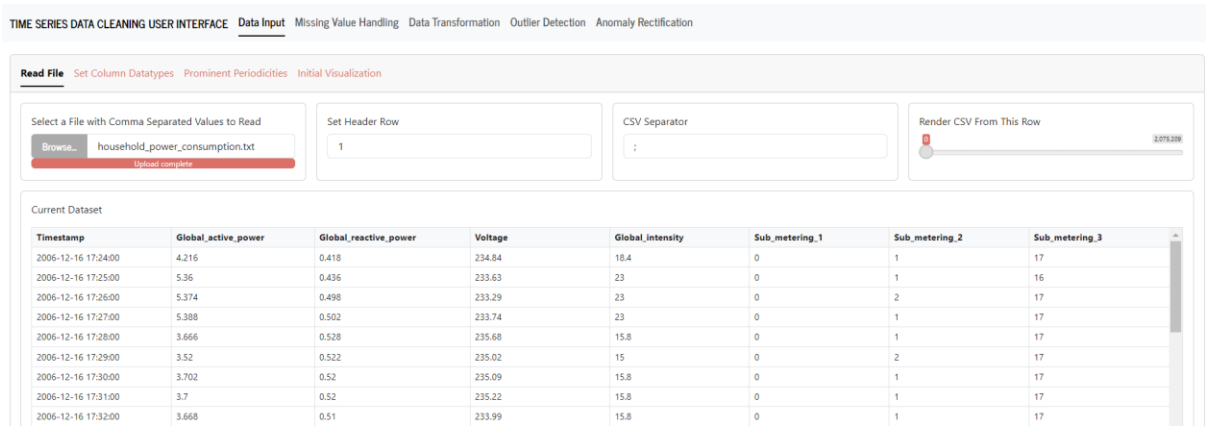


Figure 3.1: The file reading page in the data input module

The second page of the data input module is datatype setting, as shown on Figure 3.2. The screen is divided into two parts, the one on the left showing the current datatypes of the previously read file, and the one on the right being for control purposes.

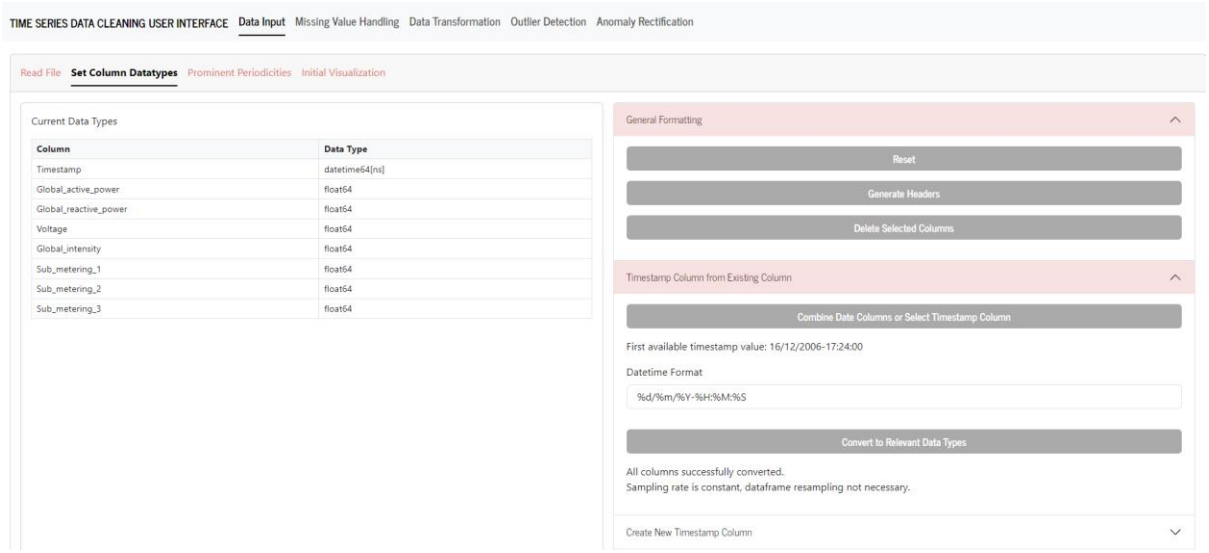


Figure 3.2: The datatype setting page in the data input module

Looking at the settings, it can be seen that it is divided into four segments. The first segment, general formatting (Figure 3.3) allows the user to reset their formatting progress, generate generic headers (such as Column 1, 2, etc.) in case they are not available in the data, or to delete columns selected on the table on the left side of the page. Following this step, the user has two formatting options available: The first is to select a timestamp column from the data available in the read csv file, and the second is to create a timestamp column from scratch.

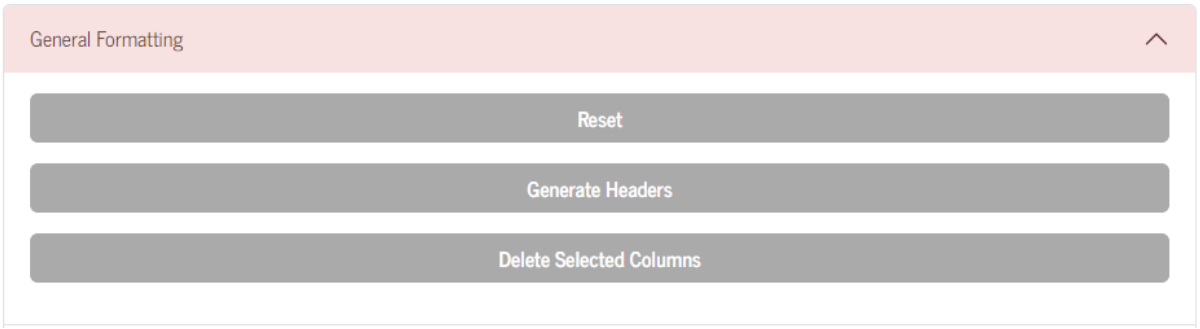


Figure 3.3: The general formatting segment in datatype setting page

Concerning selecting a timestamp column from existing column(s), the available options can be seen on Figure 3.4. First the user has to select one or multiple columns from the table on the left side of the page to create the timestamp column. Once these columns are merged, the timestamp column is created, and the text below the button will display the first available timestamp value. Based on this first available value, the user can set the datetime format, and click on the button on the bottom to convert the timestamp column to the datetime format, and all other columns to a numeric format. Setting the datetime format is assisted by a tooltip when hovering over the textbox. Once the conversion is completed, the user will get feedback about the results of the conversion below the conversion button.

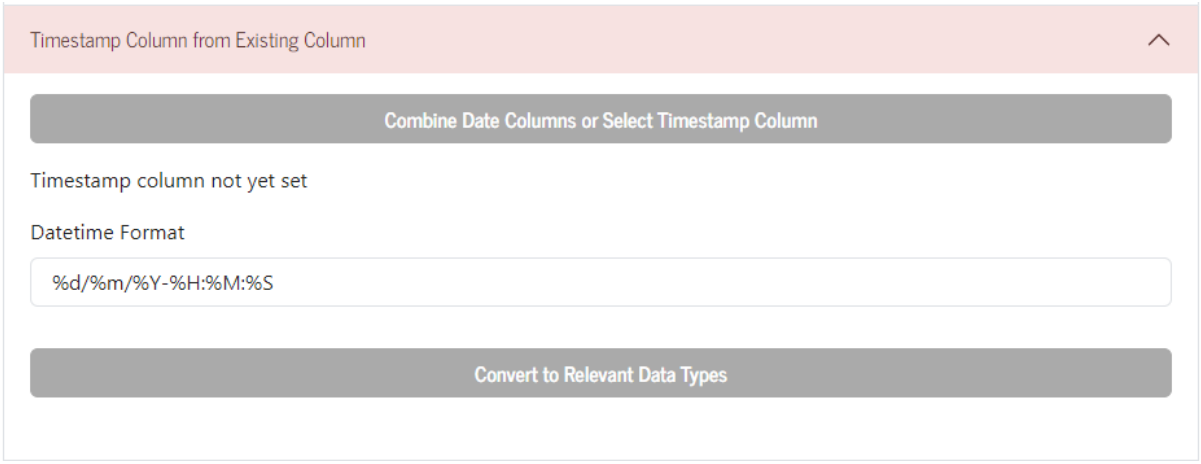


Figure 3.4: The timestamp from existing column segment in datatype setting page

Concerning new timestamp column creation, the available settings can be seen on Figure 3.5. Here the user can create a column with an end date as selected on the top with a sampling frequency based on the two middle input fields (in this case it would generate a timestamp every second with the final timestamp being the date selected). Once again, a conversion button can be clicked, creating the timestamp column and converting all other columns to a numeric format. The outcome of the conversion can be read below the conversion button once it is finished.

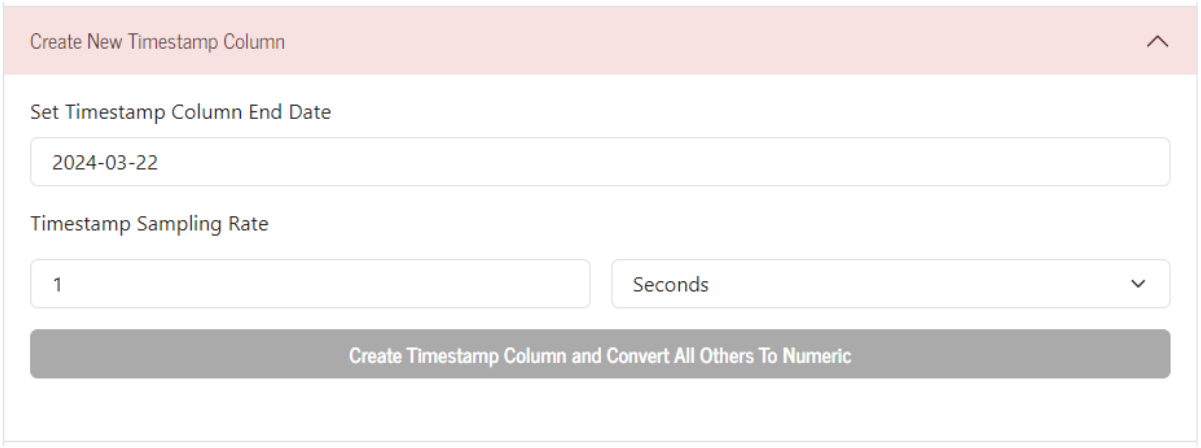


Figure 3.5: The new timestamp column segment in datatype setting page

In certain cases, the user may want to resample their dataframe, and this can be accomplished with the resampling segment on the page, as seen on Figure 3.6. Reasons to do this may be to limit the amount of data, or to achieve a constant sampling frequency. In case the time difference between any two neighboring datapoints is not constant, the user will receive feedback following datatype conversion that resampling is recommended, and the recommended resampling rate will also be written at the resampling segment. This will result in the dataframe achieving a constant sampling frequency, which is of special importance for missing value imputation and data transformation later down the line.

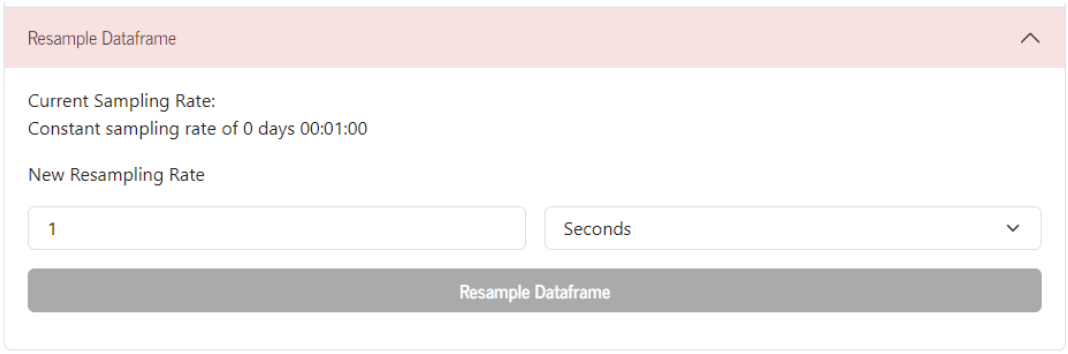


Figure 3.6: The resampling segment in datatype setting page

3.3 Periodicity Estimation

The third page of the data input module is periodicity estimation, which is a required step for future calculations such as missing value imputation and data transformation (Figure 3.7). Here, before calculating the prominent periodicities, two settings can be given, both affecting the outcome of the calculation. The samples in the (P)ACF calculation (described in depth in Section 3.5) affects runtime, with a larger value lowering runtime, but a smaller value increasing the odds of detecting more periodicities that are only there for a shorter timeframe. The second setting is the strength cutoff. A higher strength cutoff will result in a stricter periodicity detection.

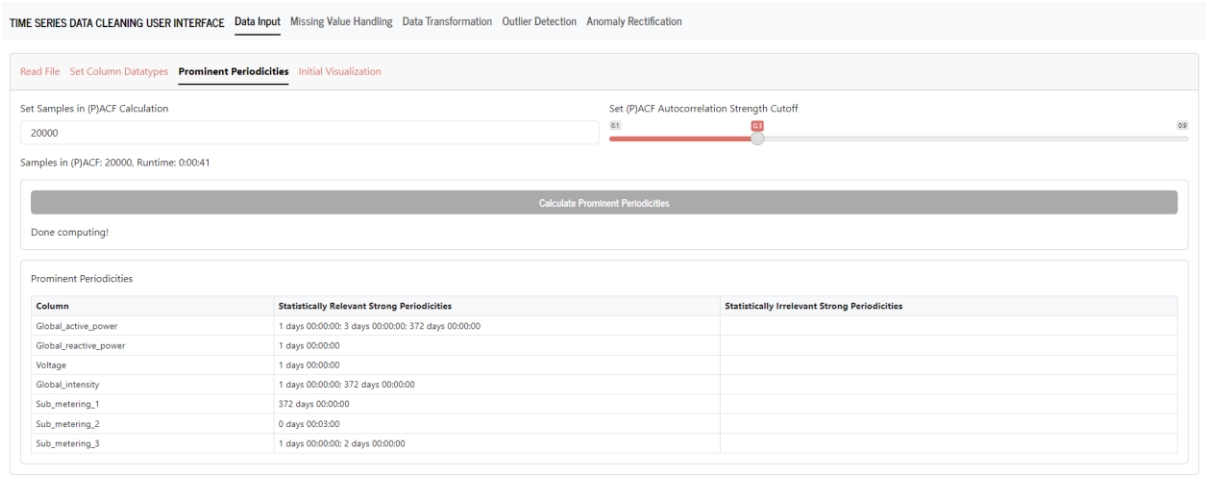


Figure 3.7: Periodicity estimation in the data input module

The recommended values for the periodicity estimation:

- Samples in (P)ACF calculation
  - 20000 for large datasets
  - With smaller datasets, some fraction of the entire length of the data
- (P)ACF autocorrelation strength cutoff
  - 0.3 as a default value
  - 0.5 as a more strict value, in case too many periodicities are detected

Following the calculation, the results are shown on the table below the calculation settings. Here, the statistically relevant strong periodicities will be used for future calculations, whereas the statistically irrelevant periodicities will be discarded, but may serve as useful information to the user (i.e., the algorithm detects a yearly period that the user knows is in the data, however, it will choose not to use it for calculations).

3.4 Initial Visualization

The final page of the data input module contains the initial visualization. This page has two tabs, single plot (Figure 3.8) and paired plot (Figure 3.9). Both of these plots settings are as follows. A resampling rate can be given as to look at the data over a larger timeframe than the original sampling rate. The maximum samples on the plot determines how many samples with the previously determined sample rate are shown, serving as a type of zooming method (e.g., 24 samples with an hourly sampling rate will result in a plot showing a single day). The column to visualize on the plot can also be determined. Finally, in case the maximum number of samples is smaller than the total number of samples in the resampled data, the user can use a slider to pan along the data on the time axis, and pressing the play button below the slider will animate the plot, making it pan along the time axis automatically.

Figure 3.8: The single plot tab on the initial visualization page of the data input module

Figure 3.9: The paired plot tab on the initial visualization page of the data input module

3.5 In Depth on the Periodicity Estimation Algorithm

This final section of the data input module aims to go in depth on the workings of the periodicity estimation algorithm used in the user interface. An overview of the algorithm is as follows:

1. Determine the original sampling frequency of the data
2. Determine the timespan of the data, and create a list of resampling frequencies between the original sampling frequency and the total timespan
3. Iterate through the resampling frequency list
    a. Select resampling frequency
    b. Resample data
    c. Break down data into chunks of size given in the samples slider
    d. Iterate through slices of data
        i. Get slice of data
        ii. Use Partial Autocorrelation Function (PACF) and Autocorrelation Function (ACF) on the data chunk (Figure 3.10)
        iii. Get peaks of both ACF and PACF exceeding the strength cutoff value set on the slider (Figure 3.10 – blue points)
        iv. Filter peaks as to have a lag of at least 3 (determined through testing, the PACF and ACF detects periodicities well with this setting) (Figure 3.10 – x axis at least 3)
        v. Split peaks into ones where the 95% confidence interval is exceeded and to ones where it is not (hence statistically relevant and irrelevant periodicities) (Figure 3.10 – blue points exceeding the orange area are statistically relevant)
        vi. Multiply lag by resampling rate of the chunk to get the prominent periodicity value (e.g., 1 day)
4. Cluster all detected periodicities into clusters having a maximum range of plus or minus 10% from the cluster mean (i.e., a chunk has 1 day, 1.01 day, 1 day, all of these are put in the same chunk with a mean of 1 day), and refer to periodicities as cluster mean henceforth
5. If a cluster is present in statistically irrelevant periods but is not present in relevant periods, add a new periodicity to irrelevant periods

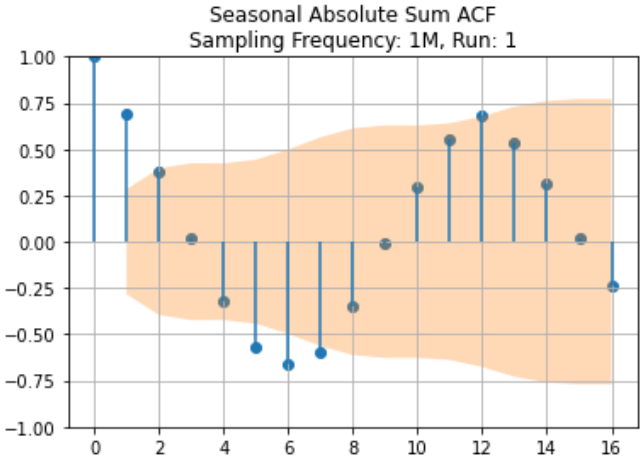


Figure 3.10: An ACF decomposition of a chunk of data

4 The Missing Value Handling Module

4.1 Overview

The purpose of the module discussed in the following paragraphs is to give the user an overview of the amount of missing values in their data, how these missing values are linked to each other, and finally to impute these missing values, meaning to put something in their place, resulting in a dataset that should be free of missing values.

4.2 Descriptive Dataframes

The first page of the missing value (MV) handling module are the descriptive dataframes as seen on Figure 4.1. The table on the left shows the user how many continuous occurrences of missing values there are in the data, what is the start index of these occurrences, and how many points are missing from this start index.

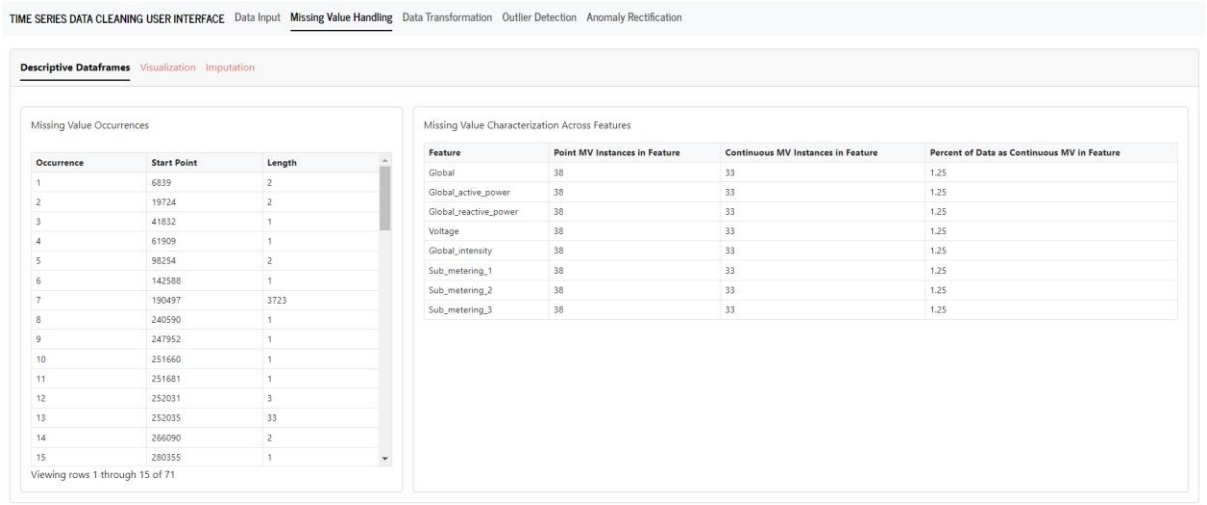


Figure 4.1: The descriptive dataframes page on the missing value handling module

The table on the right tells the user how many points are missing in each column of the data (i.e., how many rows in the MV occurrences dataframe that have a length equal to 1), how many continuous missing value instances there are (i.e., how many rows in the MV occurrences dataframe that have a length greater than 1), and what percent of the data is continuous missing values (this basically tells the user that the average continuous missing value length is x% long, meaning that on average, there is (100-x)% of data having a value, followed by x% of data not having a value). This dataframe begins with the global column, which looks at all columns of the dataset to see if any feature has a missing value in a given timestamp.

4.3 Missing Value Visualization

The second page of the missing value handling module contains the visualization of missing value occurrences between features. This page consists of three tabs, which each contain a different plot type, aiming to give the user information on how missing values occur. Common settings between these plots are whether to plot all features or only selected features as to potentially limit the information on each plot.

Of the three tabs of data in missing value visualization, the first tab plots the cumulative sum of missing values across selected features, as shown on Figure 4.2. This plot shows the user how

many missing values there are in total as time moves forward. In this case, there are in total about 27000 missing values, and it can be seen that the greatest jump in their number occurs around autumn of 2010.

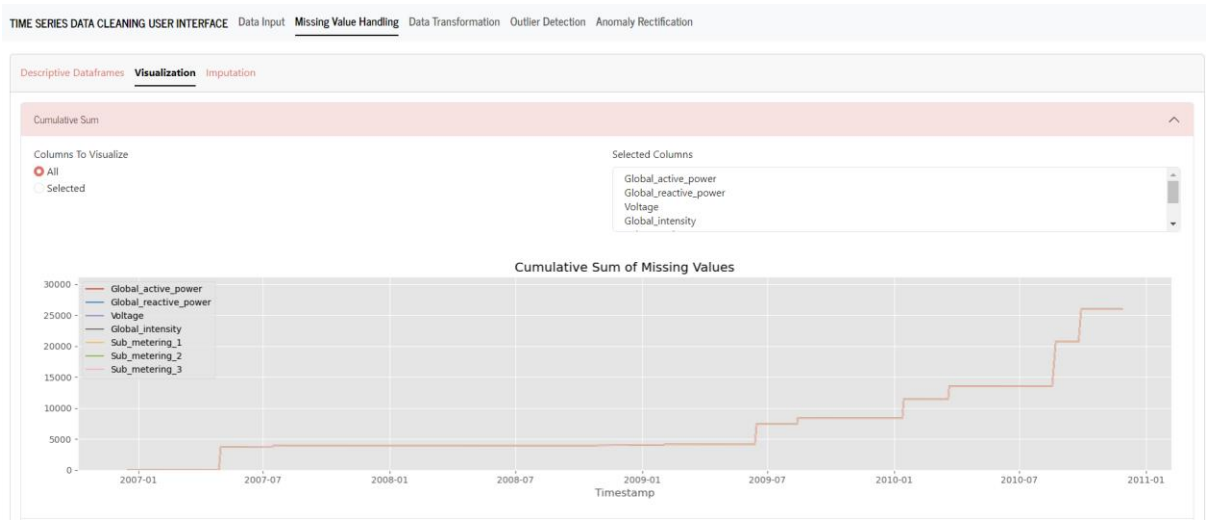


Figure 4.2: Plotting the cumulative sum of missing values

The second tab in missing value visualization allows the user to plot spanwise proportions of missing values, as seen on Figure 4.3. This breaks the data down into a given number of spans of equal length in time, and shows the user what percentage of all missing values are in a given span. In case of the figure, 10 spans were selected, and below the selection it can be observed that a span consists of 144 days. Based on the plot, it can be observed that the final 144 days has the highest amount of missing values. Lowering the number of spans to 4 results in a span length of 360 days which is about one year: this plot tells the user that there are barely any missing values in the second year of measurements.

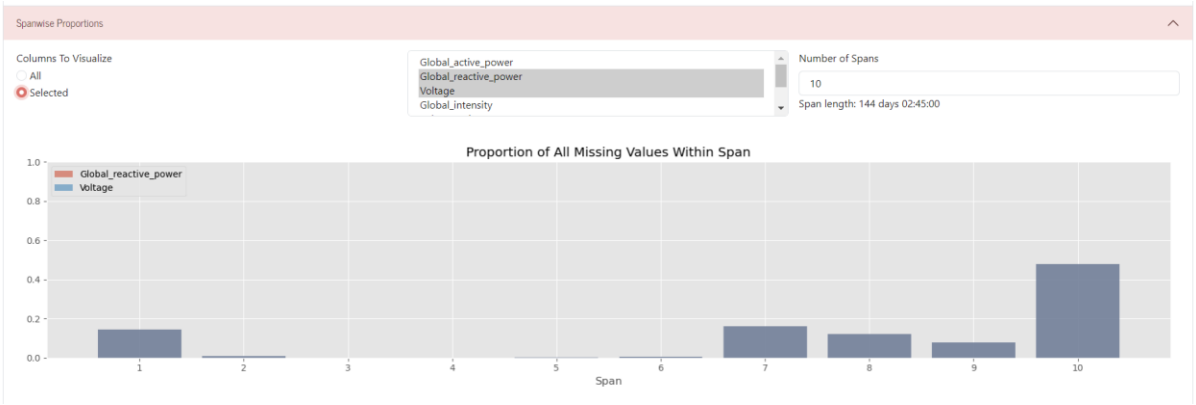


Figure 4.3: Plotting the spanwise proportions of missing values

The third and final tab in missing value visualization plots the correlation heatmap between features' missing values, as seen on Figure 4.4. This plot tells the user how the missing values are overlapped between features. In case of the figure, all features are perfectly overlapped, they all have a correlation of 1, meaning that if one feature has a missing value, all others will have one in that timestamp as well.

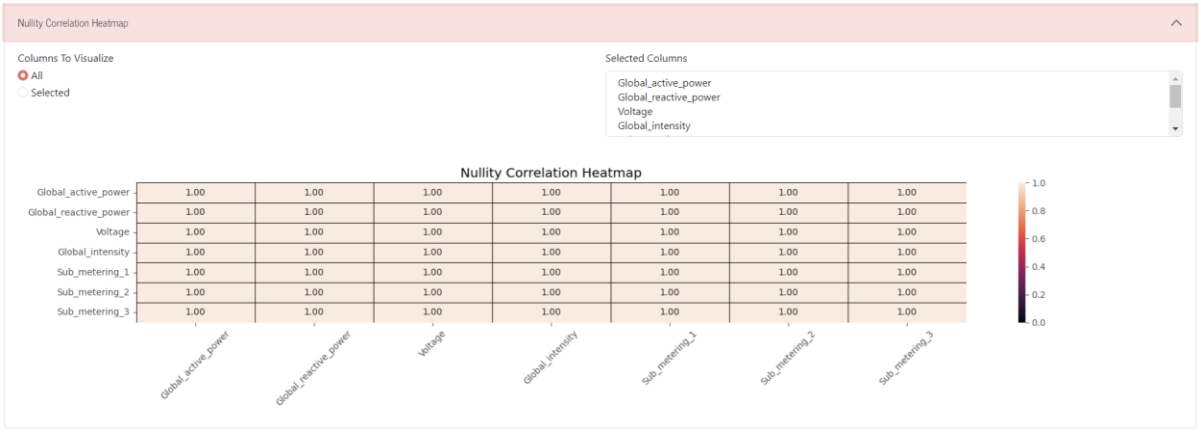


Figure 4.4: Plotting the correlation heatmap of missing values

4.4 Missing Value Imputation

The third and final page of the missing value handling tab allows the user to impute missing values based on the periodicity estimator results. As seen on Figure 4.5, this page consists of three parts: calculation settings, plot settings, and a plot.

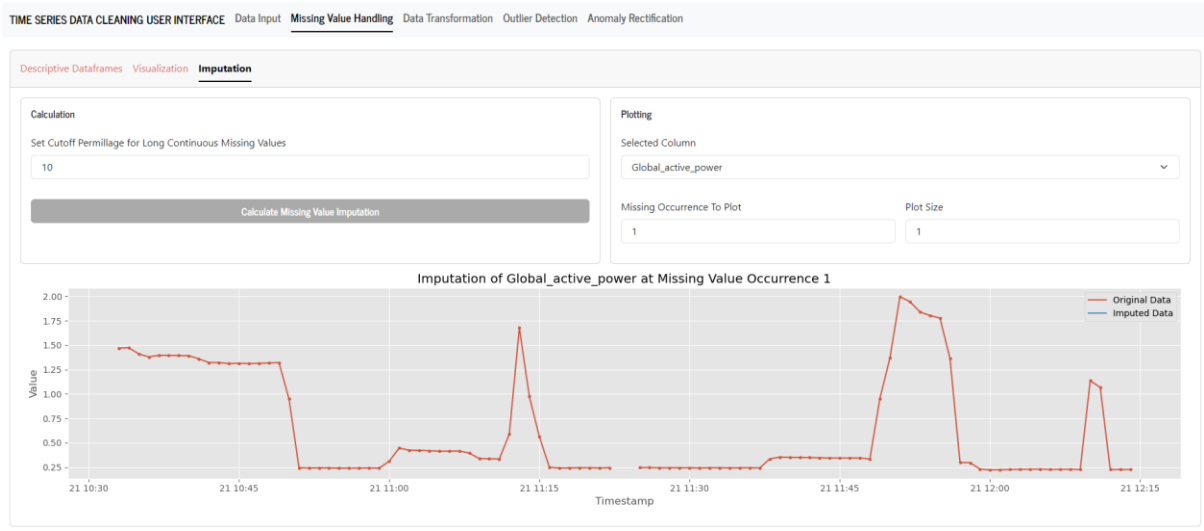


Figure 4.5: The missing value imputation page in the missing value handling module

Concerning calculation settings, the cutoff permillage allows the user to set from what continuous missing value length should a given period be taken into account (in case of a periodicity of 100 minutes and a permillage of 10, the periodicity of 100 minutes will be added to the imputation in case that the missing value is longer than 1 minute). Once the permillage is set, the calculation may commence by clicking on the button. Following a successful calculation, a text will appear beneath the button, informing the user that there are no missing values left in the dataset.

The recommended values for the missing value imputation:

- Cutoff permillage
  - 10 as an all-round value
  - In case the user is unsatisfied with the results, the periodicity estimation results should be recalculated before the effects of this value are tested

Concerning plot settings, a dataset feature may be selected along with a missing value occurrence, as seen in Section 4.2. Finally the plot size serves as a kind of zooming function, plotting n-times the continuous missing value length in both directions of the continuous missing value.

A resulting plot following missing value imputation can be seen on Figure 4.6. Here the original data is plotted in red, and the newly imputed data is plotted in blue, showing the user how each missing value is filled in within the context of the neighboring data.

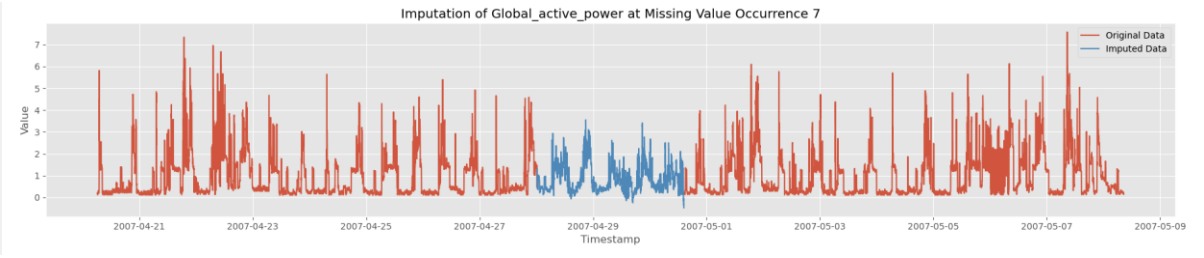


Figure 4.6: The resulting plot from a successful missing value imputation

As shown on the flowchart on Figure 2.4, the periodicity estimator results may be recalculated depending on the missing value visualization. This means that if the user is unsatisfied with the results shown on the imputation result plot, they may recalibrate the settings of the periodicity estimator, and recalculate the missing value imputation until the results are satisfactory.

4.5 In Depth on the Missing Value Imputation Algorithm

This final section of the missing value handling module aims to go in depth on the workings of the missing value imputation algorithm used in the user interface. An overview of the algorithm is as follows:

1. Iterate through features
    a. Create missing value occurrence dataframe for feature
    b. Get statistically relevant periodicities for feature
    c. Depending on the smallest periodicity, split the MV occurrence dataframe into short continuous (SC) and long continuous (LC) segment
        i. If feature is aperiodic or if the MV length is smaller than permillage times smallest periodicity length, then treat it as a SCMV, otherwise treat it as an LCMV
    d. SCMV imputation
        i. Iterate through SCMV occurrences
            1. Get occurrence start and end index
            2. Exponential moving average (EWMA) on maximum 1000 points before and after the CMV
                a. EWMA filters out the noise, meaning that the imputation result will not be affected by it
            3. Get EWMA start and endpoints for SCMV imputation
            4. Do SCMV imputation through linear interpolation between EWMA points
    e. LCMV imputation
        i. Get optimal LCMV occurrence
            1. The one with the most available space

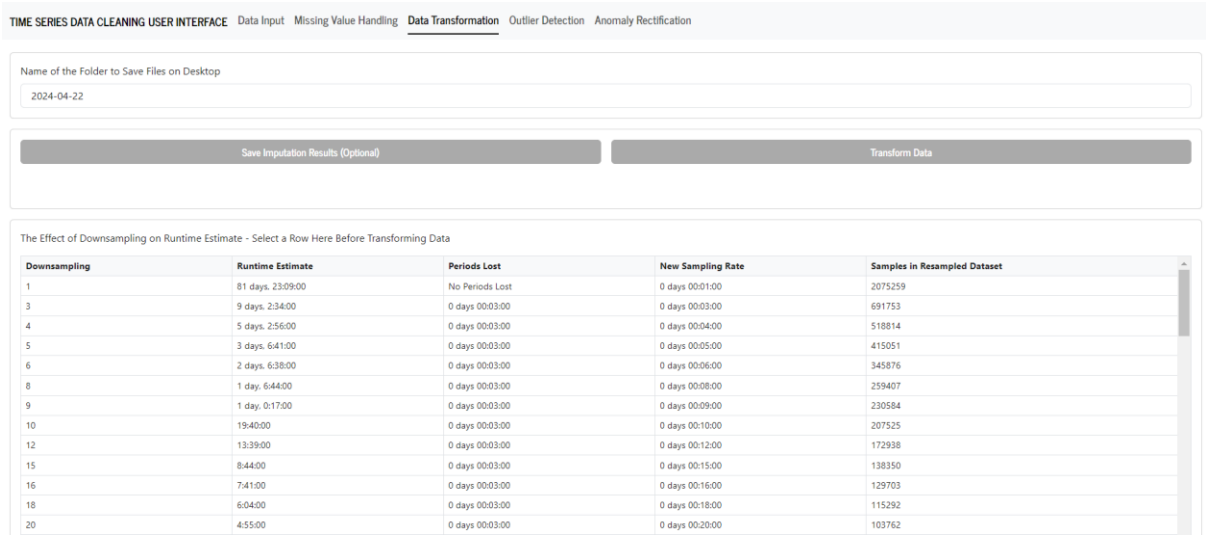
2. Use seasonal decomposition on this occurrence, iterating by giving a seasonal result based on fewer periods in the slice a greater weight
    a. Seasonal decomposition preferred over (M)STL, as the runtime is negligible
3. Completely deseasonalize the data before and after the LCMV, getting the EWMA start and endpoints for linear interpolation
4. Add the the linear interpolation each seasonal component multiplied by a distance factor
    a. At the start of the CMV the left side seasonal component has a weight of 1 and at the end it has a weight of 0, unless the right side decomposition could not be done, in which case the distance factor is constant 1

f. Leftover SCMV imputation
    i. The LCMV imputation may have certain occurrences where periodicity can not be taken into account, since there is insufficient data before and after the CMV, meaning that these missing values need to be filled in with the SCMV method, through simple linear interpolation

5 The Data Transformation Module

5.1 Overview

The purpose of the module discussed in the following paragraphs is to decompose the missing value-free data into trend, seasonal, and residual components, which will then serve as an input to outlier detection. The only page of the module can be seen on Figure 5.1. As the results will be saved to a folder on the desktop, the user can name said folder using the textbox on the top of the page. Following this, the user can choose to save the missing value imputation results to this folder (the saving of these results is not necessary for moving forward) or move forward with the outlier detection task by transforming the data.



Downsampling	Runtime Estimate	Periods Lost	New Sampling Rate	Samples in Resampled Dataset
1	81 days. 23:09:00	No Periods Lost	0 days 00:01:00	2075259
3	9 days. 2:34:00	0 days 00:03:00	0 days 00:03:00	691753
4	5 days. 2:56:00	0 days 00:03:00	0 days 00:04:00	518814
5	3 days. 6:41:00	0 days 00:03:00	0 days 00:05:00	415051
6	2 days. 6:38:00	0 days 00:03:00	0 days 00:06:00	345876
8	1 day. 6:44:00	0 days 00:03:00	0 days 00:08:00	259407
9	1 day. 0:17:00	0 days 00:03:00	0 days 00:09:00	230584
10	19:40:00	0 days 00:03:00	0 days 00:10:00	207525
12	13:39:00	0 days 00:03:00	0 days 00:12:00	172938
15	8:44:00	0 days 00:03:00	0 days 00:15:00	138350
16	7:41:00	0 days 00:03:00	0 days 00:16:00	129703
18	6:04:00	0 days 00:03:00	0 days 00:18:00	115292
20	4:55:00	0 days 00:03:00	0 days 00:20:00	103762

Figure 5.1: The data transformation module

As the data transformation was identified as the bottleneck of the user interface, in certain cases it may be necessary to resample the data before calculating the data transforms. Selecting a resample rate can be done by selecting a row on the resampling table in a similar way to the one described in Section 3.2. Once a resample rate is selected, clicking the transform data will begin the calculation, and finally save the result on the previously defined folder.

The resampling table below contains information as to what is the estimated runtime of the data transformation with each given downsampling, which prominent periodicities are lost through this downsampling, what the new sampling rate will be, and how many samples will be in the transformed dataset. In case of the figure, it can be seen how no downsampling (meaning downsampling of 1) results in a runtime estimate of 81 days, meaning that the data should probably be resampled.

5.2 In Depth on the Data Transformation Algorithm

This section of the data transformation module aims to go in depth on the workings of the data transformation algorithm used in the user interface. An overview of the algorithm is as follows:

1. Resample data based on the selected downsampling
2. Iterate through features
    a. Get missing value-free resampled feature data
    b. Get list of statistically relevant prominent periodicities with the original sampling rate
    c. Get list of statistically relevant prominent periodicities with the selected downsampling
        i. Divide all periodicities by the downsampling
        ii. Discard any periodicity less than 3
        iii. Iterate
    d. If number of periods is 0
        i. Decomposition into trend and residual
            1. Trend from exponentially weighted moving average (EWMA)
            2. Residual from original signal – trend
    e. If number of periods is 1
        i. Decomposition into trend, seasonal, and residual with STL
    f. If number of periods is greater than 1
        i. Decomposition into trend, seasonal components, and residual with MSTL
    g. Save results to previously determined folder
    h. Update runtime estimate

6 The Outlier Detection Module

6.1 Overview

The purpose of the module discussed in the following paragraphs is to detect all outliers within the previously transformed data. This is done by reading the transformed data, calculating a set of outlier detection results along certain settings, and finally selecting optimal settings to move forward with based on visualization results.

On this page, the user will receive two outputs, one being a Boolean outlier detection result, and the other being a relative outlier detection result. Both results contain all timestamps, measured values, and point anomalies for all signals, but they are different inasmuch the Boolean result has columns for the continuous errors being average, increased, or decreased, whereas the relative result only has columns for the continuous errors with the changepoint value relative to the series-wide value (for example a relative value of 1.2 means that the average (or standard deviation in case of residual) value is 20% greater than the series average (or standard deviation)).

6.2 Data Transformation Results

The first page of the outlier detection module allows the user to read the transformation results and to visualize it, as shown on Figure 6.1. The user should select all relevant files with their filenames ending in “Data Transform” as an input by clicking the browse button. After this, the results can be visualized in a manner similar to the ones discussed in Section 3.4, allowing the user to select a feature to visualize, to resample it, to zoom into the plot, and to scroll along the time axis or to animate this scrolling. The only novelty when compared to the aforementioned section is the possibility to select components to visualize. The user can select multiple components to add together at each timestamp, and selecting all components will result in the plot showing the original, missing value-free signal (i.e., the results obtained after missing value imputation).

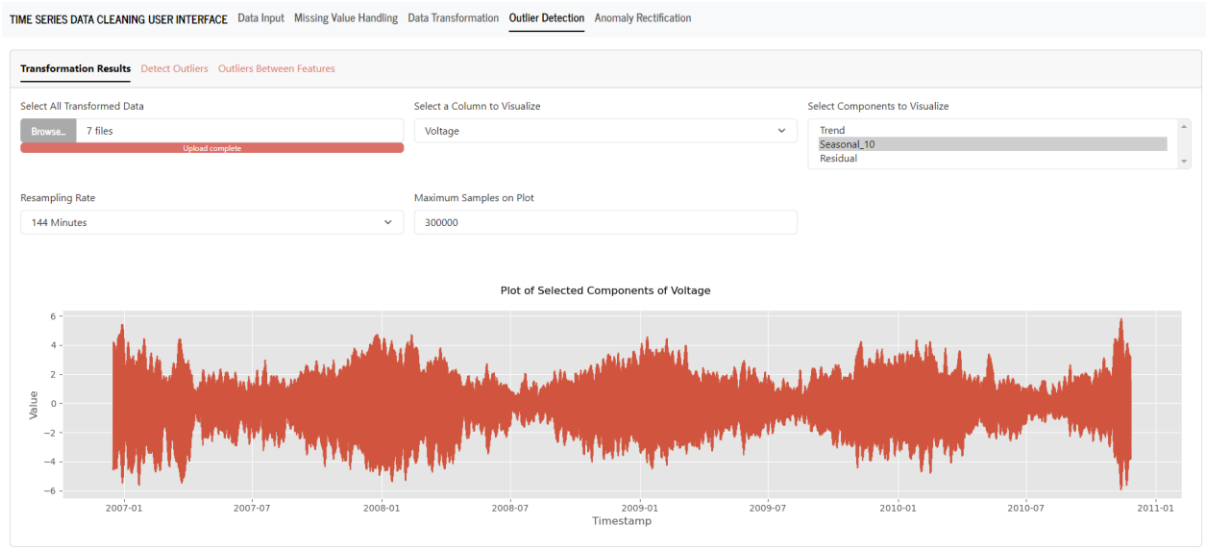


Figure 6.1: The transformation results page on the outlier detection module

6.3 Outlier detection

The second page of the outlier detection module contains two tabs, one for outlier detection calculation, and the second for outlier detection plotting.

The first tab (Figure 6.2) allows the user to adjust the settings of the outlier detection calculation based on the previously read data transforms. Alternatively, previously calculated outlier detection results can be selected in a manner similar to the one discussed in Section 6.2, except that the selected files ought to end with "Outlier detection". Before clicking on the calculation button, certain features may be given: these include a sliding window size, which determines the sensitivity of the results, a minimum changepoint length, which determines the minimum number of points that should be between two changepoints (the section between two changepoints ought to be homogeneous), and the jump parameter, increasing which will lower the computational time but will decrease the accuracy of changepoint detection. The used strictness types can also be set on this panel. Clicking the compute button will save the results to the folder defined in Section 5.1 on the desktop.

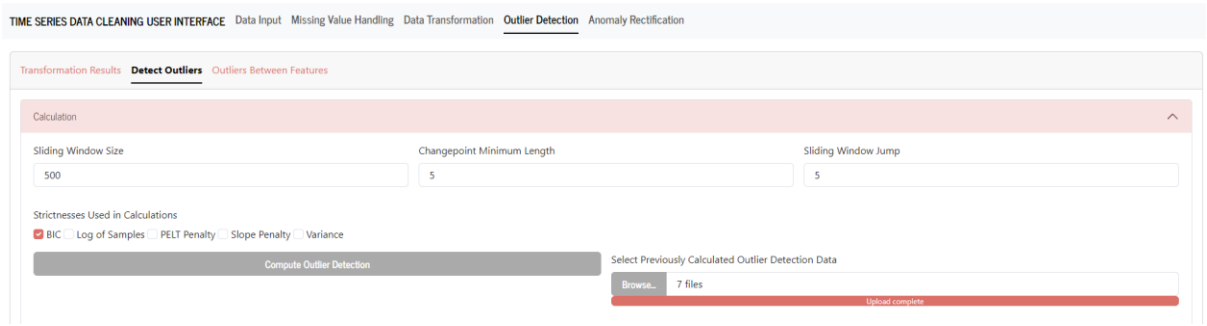


Figure 6.2: The outlier detection calculation tab in the outlier detection module

The recommended values for the outlier detection:

- Sliding window size
  - 500 as an all-round value
  - In case there are less than about 5000 data transformation timestamps, it should be lowered relative to data size
- Changepoint minimum length
  - Minimum 2
  - 5 as an all-round value
  - A larger value can filter out the short changepoints
- Sliding window jump
  - 1 if the changepoint location has to be determined perfectly
  - Otherwise, an increase in this value will lower accuracy (e.g., a jump of 5 means that the actual changepoint can be anywhere in a +-5 point radius from the one determined), but speed up the computation
- Strictness type
  - Only using BIC at the start is a good all-rounder and lowers computational time
  - However, if the user does not find any good outlier detection results, they can test other penalty calculation methods

Once the outlier detection calculation is completed or the user uploaded the outlier detection files, the user may move forward to the anomaly visualization tab. Here each component within

every feature can be selected, along with the setting of a penalty calculation method and strictness value. The user can select error types to visualize on the plot, and can observe the changepoints in the black dashed lines. For all penalty methods aside from PELT Penalty and Slope Penalty, a larger strictness value means a greater strictness, whereas for the aforementioned two, a smaller strictness value means a less strict result when it comes to changepoint detection.

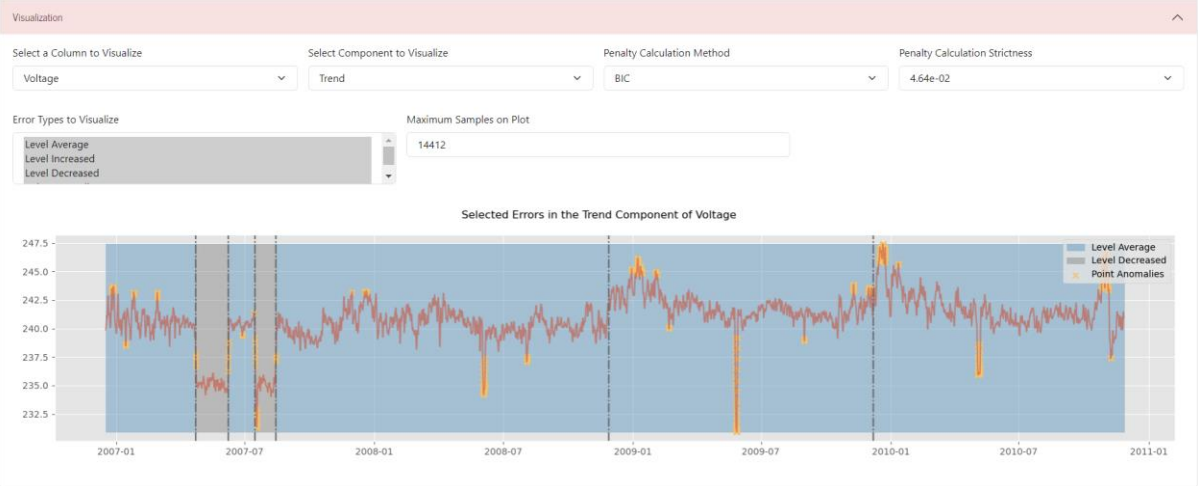


Figure 6.3: The outlier detection visualization in the outlier detection module

6.4 Outliers Between Features

As the current outlier detection result includes a large number of strictness calculation methods and values, the purpose of the following page of the outlier detection module is to select a single pairing of method and value for each of the components to use as the error detection method. Selecting these values is done by going through plot options shown on Figure 6.3, and selecting settings for each component where the user is satisfied with the changepoint results. These methods and values can be set on the plot settings tab of the anomalies between features page as seen on Figure 6.4.

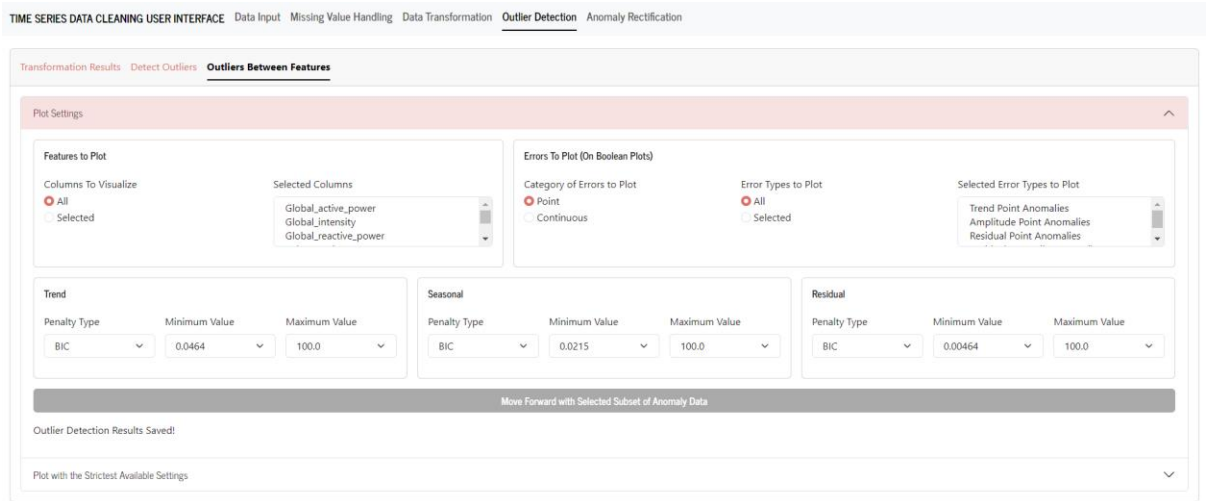


Figure 6.4: The plot settings tab of the outliers between features page

Once the strictness methods and values are set for each component, a result can be calculated and saved to the folder defined in Section 5.1 on the desktop. This result will only contain a single

strictness method and value pair for each component, and also includes the original, missing value-free data. It should be noted that clicking this button will result in selecting the highest possible changepoint strictness within the given limits, meaning that for PELT and Slope penalty the penalty value closest to the maximum, whereas with all other penalties the penalty value closest to the minimum will be selected.

Once a single set of penalty methods and values has been defined, the user may move on the plot tab of the anomalies between features page. Here, similarly to Section 4.3, a Boolean cumulative sum plot (Figure 6.5), a Boolean spanwise plot (Figure 6.6), a Boolean correlation heatmap (Figure 6.7), and a relative correlation heatmap (Figure 6.8) can be generated for the selected penalty values. As these plots can become very crowded, it may be worthwhile to adjust the plot settings in the previous tab, selecting a certain subset of features, or a certain subset of error types to visualize on them. Concerning how the Boolean and the relative anomalies are different, more is written on the overview section of this chapter (Section 6.1).

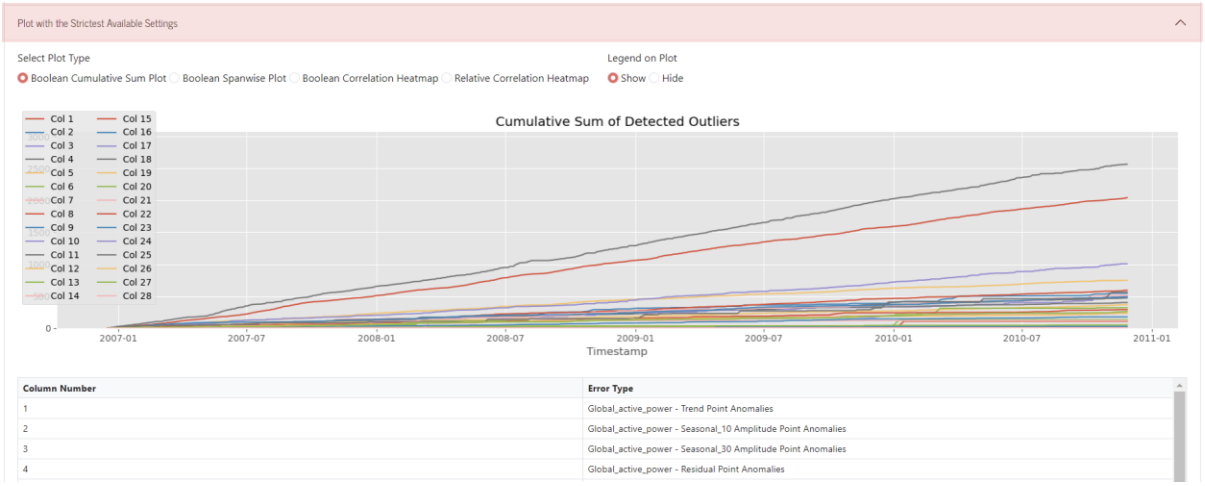


Figure 6.5: The Boolean cumulative sum of detected outliers

Figure 6.6: The Boolean spanwise plot of detected outliers

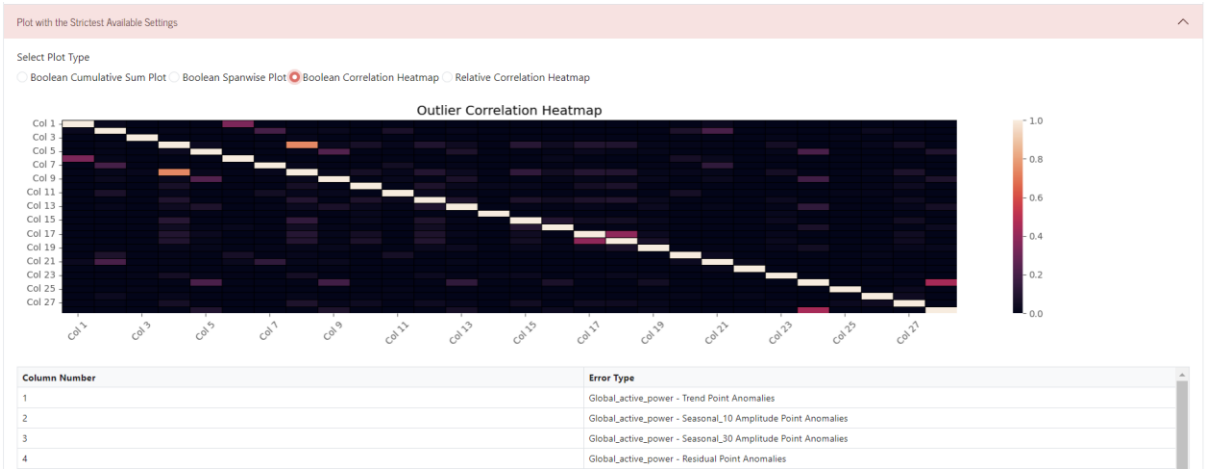


Figure 6.7: The Boolean correlation heatmap of detected outliers

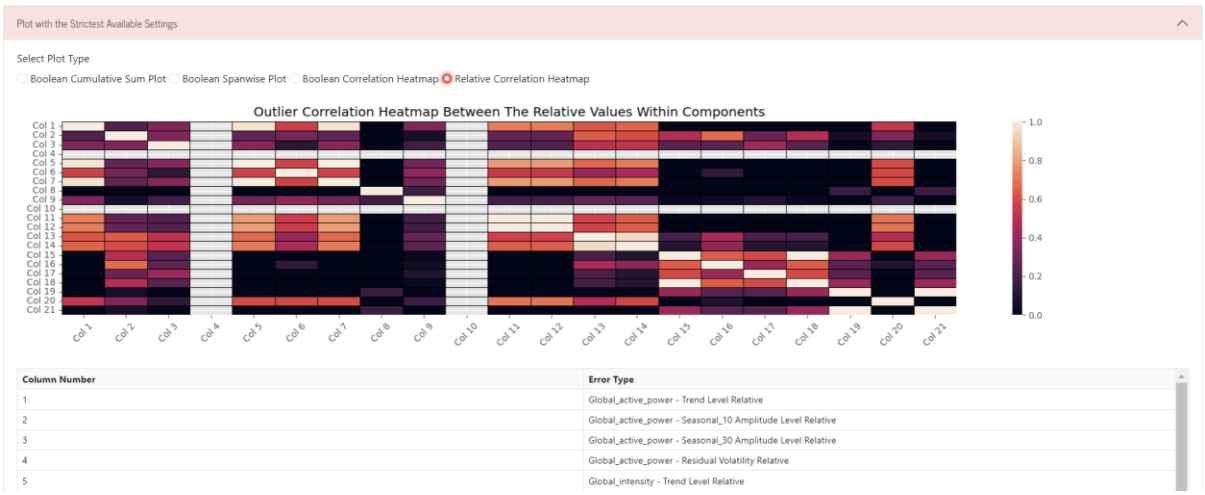


Figure 6.8: The relative correlation heatmap of detected outliers

Ultimately, these plots should serve as a useful tool for the user in seeing the relationships between detected outliers, and specifically the correlation heatmap can give the user an idea on what anomalies occur frequently together.

Of these plots perhaps the two correlation heatmaps contain the most useful information. The Boolean correlation heatmap (Figure 6.7) tells the user that if one feature has an anomaly, how high is the chance that another feature will have another anomaly, e.g., how well does the model that if voltage level is increased then the current level is also increased fit. The relative correlation heatmap (Figure 6.8) tells the user how well individual continuous errors values are correlated, e.g., if a section's voltage trend level is 20% greater than the average voltage trend level, how well does the assumption that the same section's current trend level is 20% greater than the average current trend level fit.

6.5 In Depth on the Outlier detection Algorithm

This section of the outlier detection module aims to go in depth on the workings of the outlier detection algorithm used in the user interface. An overview of the algorithm is as follows:

1. Iterate through data transform features
    a. Iterate through feature components
        i. Iterate through strictness combinations (method and value)
            1. Use sliding window segmentation with the parameters defined in calculation settings
            2. For the trend component detect changepoints in level using the l2 model
            3. For the seasonal component detect changepoints in the absolute sum within each period using the l2 model
                a. This will detect changes in amplitude
            4. For the residual component detect changepoints in variance using the normal model
                a. The normal model detects both level shifts and variance changes, however, since level can be assumed to be constant 0, we can just use the normal model
            5. Iterate through changepoint sections
                a. Within each changepoint section find point outliers using the 1.5-times interquartile range (IQR) method
                    i. This is for trend and residual
                    ii. For seasonal, get the quotient of every nth value in the period divided by the absolute sum within each period, and see if there are outliers in this list
                b. For seasonal, also find outliers using the IQR method for the shape parameter
                    i. Shape is squared sum of kurtosis and skewness
                    ii. This should detect abrupt changes in a seasonal shape within changepoints
                c. For residual, detect normality anomalies
                    i. Get a rolling kurtosis and skewness value
                    ii. These should both be between certain values defined by rules of thumb
            6. Define sections between changepoints as either being average, increased, or decreased
                a. For trend and seasonal, compare to series average and plus-minus 1 times standard deviation in level and period absolute sum respectively
                b. For residual, compare to series standard deviation plus-minus 5% series standard deviation

7 The Anomaly Rectification Module

7.1 Overview

The purpose of the module discussed in the following paragraphs is to rectify the anomalies detected with the previous module, and to plot and save the results.

The first page of the module can be seen on Figure 7.1. Whether after saving the set strictness results, or by selecting a previous calculation with the filename beginning with “Boolean” and ending in “Set Strictnesses” by clicking on browse, the user can move forward with anomaly rectification. This module works by looking at all anomaly types across multiple features, and if anomaly context is not given, meaning that if a feature is alone in a given timestamp in having a certain error, it is flagged as an error, the point is deleted, and is imputed in a manner equivalent to the one discussed in Section 4.4.

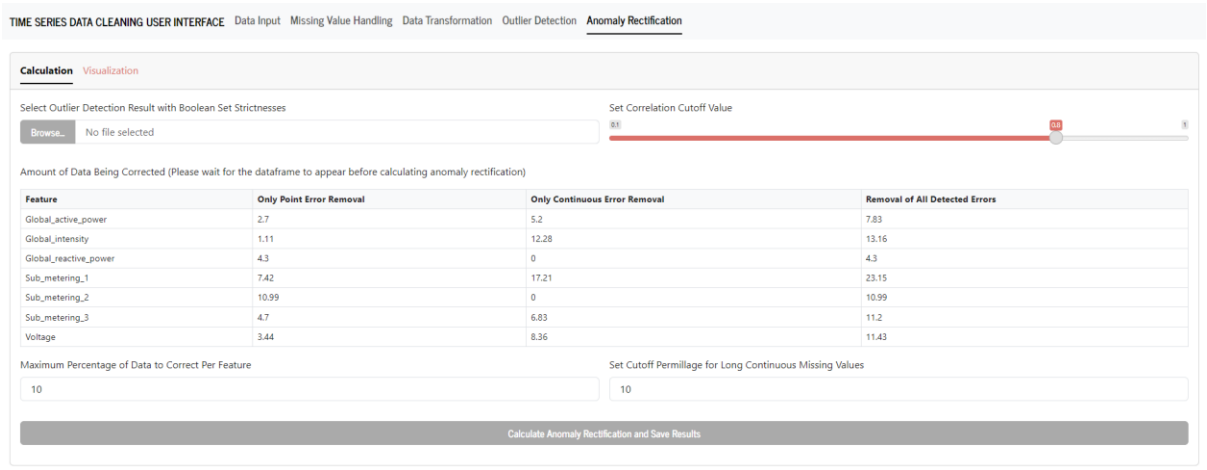


Figure 7.1: The calculation page of the anomaly rectification module

The recommended values for the anomaly rectification:

- Correlation cutoff value
  - 0.6 as a more lenient value
  - 0.8 as a more strict value
- Maximum percentage of data to correct
  - It can depend on the numbers shown on the table on this page, but about 10-20% is recommended, otherwise too much of the original data can be discarded
- Cutoff permillage for long continuous missing values
  - Similar to Section 3.3, 10 is recommended
  - This should be the last variable the user adjusts in case they are unsatisfied with the results

For point anomalies, this context is searched across all features, whereas for continuous anomalies, this is searched across feature errors that have a correlation greater than the correlation cutoff set with the slider on the top of the page, meaning that they frequently occur together. By sliding the correlation cutoff value, the user can see the effect on the amount of data to clean. Finally, before cleaning, the user can set the maximum percentage of data to correct (a too low value may mean that no cleaning is done), and a cutoff permillage can also be set in a manner similar to that in Section 4.4. Clicking the calculate button will save the results to the

desktop folder defined in Section 5.1, and will allow the user to move forward with the cleaning visualization.

The cleaning visualization can be seen on the following page and is conveyed on Figure 7.2. Here, along with the usual settings such as resampling rate, number of samples, feature selection, and scrolling (similar to Section 3.4), the user can choose to show or hide both the original imputed data and the corrected data by clicking on the switches above the plot. An example of a correction can be found from the summer of 2008 on the data, where rather than having an abrupt jump, the data is corrected to have a gradual decrease.

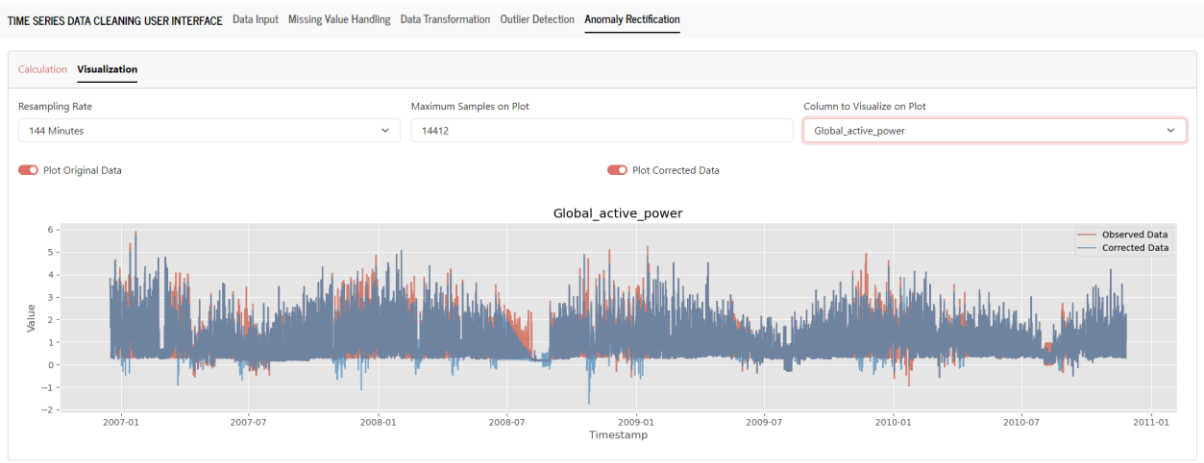


Figure 7.2: The visualization page of the anomaly rectification module

It should be noted that as opposed to the outlier detection results, the results of the anomaly rectification are highly parameter dependent, meaning that moving forward with just the detection results may be more robust in certain cases, and obtaining a good rectification result may require some tweaking of parameters.

7.2 In Depth on the Anomaly Rectification Algorithm

This section of the outlier detection module aims to go in depth on the workings of the anomaly rectification algorithm used in the user interface. An overview of the algorithm is as follows:

1. Get point anomaly context
    a. Label each feature as having point anomaly in each timestamp, yes or no
    b. If only a single feature has a point anomaly in a timestamp, append the feature-index combination to the list of point errors
2. Get continuous anomaly context
    a. Go through the anomaly correlation heatmap as shown on Figure 6.7 row by row
        i. Find elements with correlation greater than the cutoff value set in the calculation page
        ii. If there are at least 2 different features in the list of correlations filtered thus, and there are at least 3 error types correlated, then add this list to the list of correlated errors
        iii. E.g., add voltage – trend average and current – trend average to the list
    b. Go through list of correlated errors
        i. Similar to point anomaly context, find if there is a feature in this subset of continuous errors that has one value while everything else has another
        ii. Append feature-index combinations to list of continuous errors

3. Create point errors, continuous errors, and both errors feature-index combinations
4. Select which of the 3 methods described in 3) to use based on calculation settings
    a. Use maximum percent of data removed while the percentage is less than the cutoff
    b. If everything is greater than the cutoff, do not remove anything
    c. Delete values based on the selected method for each feature
5. Go through missing value imputation in the same way as described in Section 4.5

References

[1] K. Feasel, Finding Ghosts in Your Data Outlier detection Techniques with Examples in Python. Berkeley, Calif: Apress, 2022.

[2] V. Chandola, A. Banerjee, and V. Kumar, “Outlier detection,” ACM Computing Surveys, vol. 41, no. 3, pp. 1–58, Jul. 2009. doi:10.1145/1541880.1541882