# `Spark project`

***Release 0.0.1***

**Bram Vonk**

**Nov 04, 2021**

# CONTENTS:


**Note:**

This project is an Enexis case on forecasting load demand for distributon transformers.

The results should support grid planners with timely grid reinforcements and replacements.

The case is used as a graduation project for the Professional Education program of the Jheronimus Academy of Data Science

CHAPTER

# ONE

# GENERAL

## 1.1 About JADS


This project is done in the context of the Professional Education Lead Program of the Jheronimus Academy of Data Science [JADS] as graduation project.

## 1.2 About Enexis

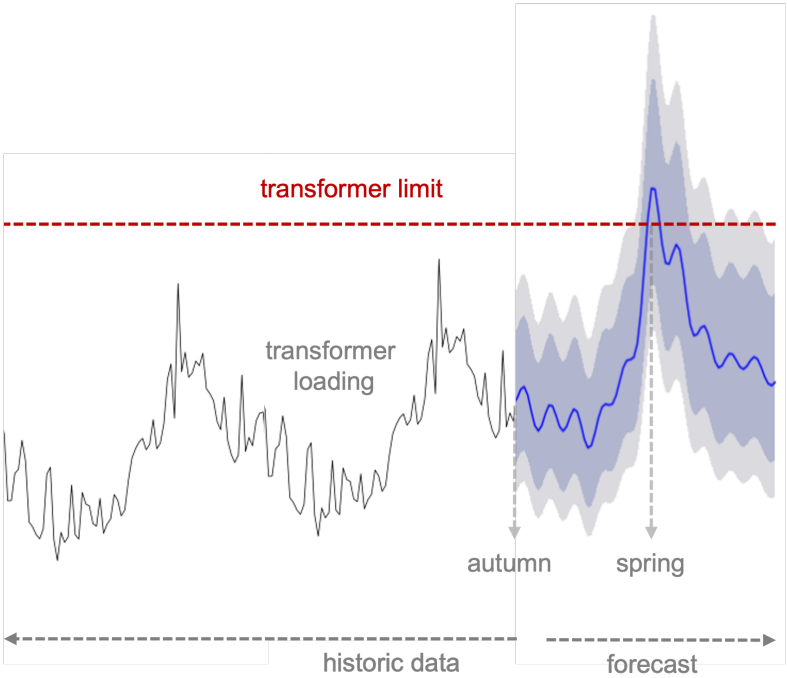

Enexis is regulated regional distribution network operator (DNO) in the Netherlands, responsible for transporting electricity and gas to 2.6 million customers. Enexis aims to keep the delivery af gas and electricity reliable, affordable and sustainable for all its customers.

## 1.3 About the current situation

Power is delivered via grid components as cables and transformers (for electricity) to residential and industrial connections of our customers. These grid components have to have the needed capacity. This is monitored by Grid Planners and if needed transformers are swapped for heavier ones, or cable connections are strengthened. For decennia this monitoring was done by mainly looking to historical yearly extremes. However, the increasing growth rate of power demand and supply driven by emerging technologies as electrical vehicles (EV) and photovoltaics (PV), requires shorter monitoring periods and nowadays even forecasts. Otherwise, there is no time for mitigating actions and customers will be out of power.

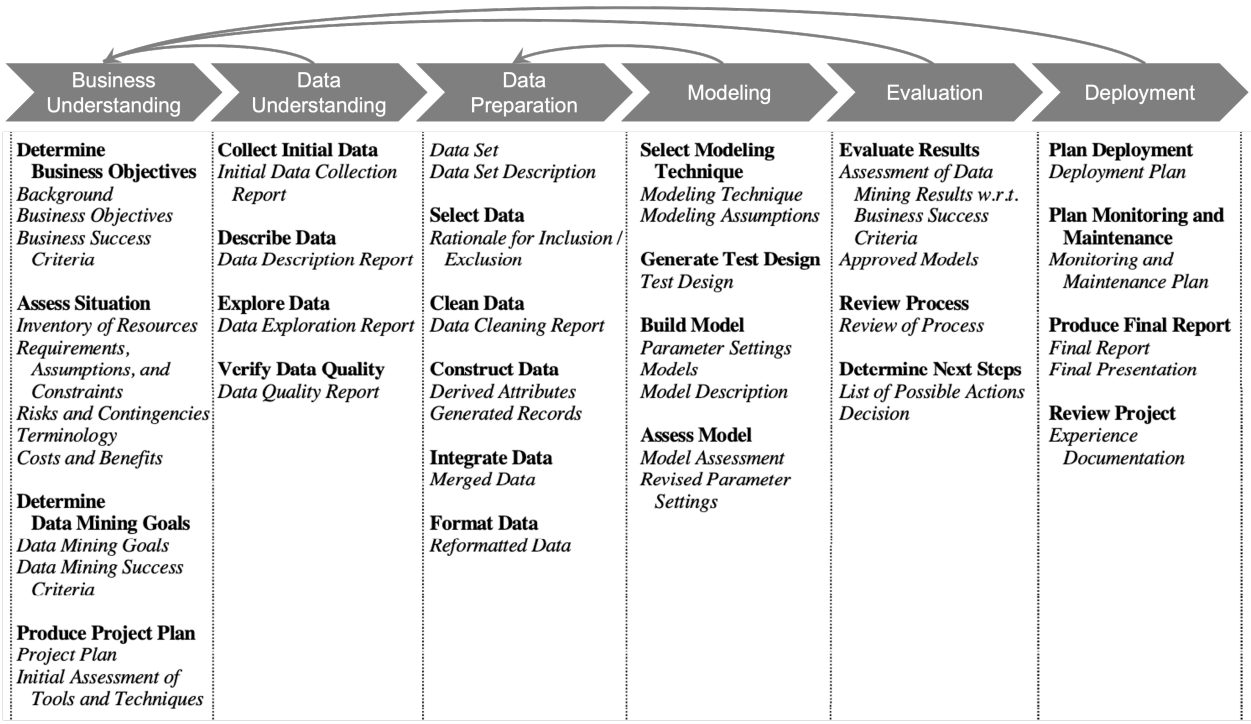
## 1.4 About this projects aim

Enexis started a few years ago with the measurement of its distribution transformer population (in the project "distribution automation light" (DALI)). Power measurements are available that enable monitoring ad hoc by the grid planners. This project takes the next step by forecasting on that data in autumn if a transformer overloads in spring.




## 1.5 About the structure of this project

This project and its documentation are set up around the CRoss Industry Standard Process for Data Mining [CRISP-DM]. It is an iterative process and this documentation focuses on the end result without ignoring the lessons learned along the way. The latter is generally noted at the end of every chapter.




## 1.6 Acknowledgements

I would like to thank Jeroen the Mast for the valuable feedback and supervision.

Special thanks also goes out to PDEng candidate Akshaya Ravi for her technical support. Together with my buddy David Rijlaarsdam she provided me with helpful insights and discussions on the project.

CHAPTER

TWO

# BUSINESS UNDERSTANDING

## 2.1 Business Objectives

Enexis is a regulated DNO with a monopoly position regarding the distribution of electricity and gas. The company wants to keep the delivery of electricity and gas:

- reliable
- affordable
- sustainable

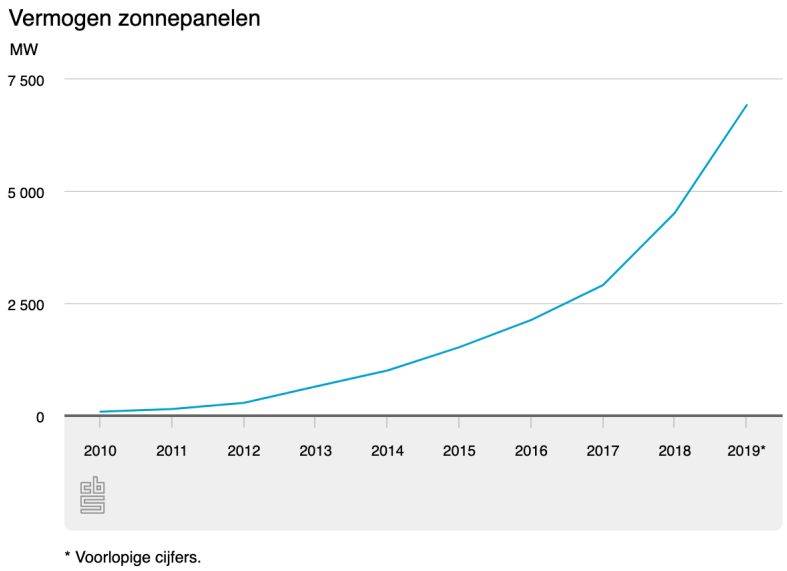
This is realised by doing effective and efficient grid investments according to the Risk and Opportunity Based Asset Management method [ROBAM].

## 2.2 Situation Assessment

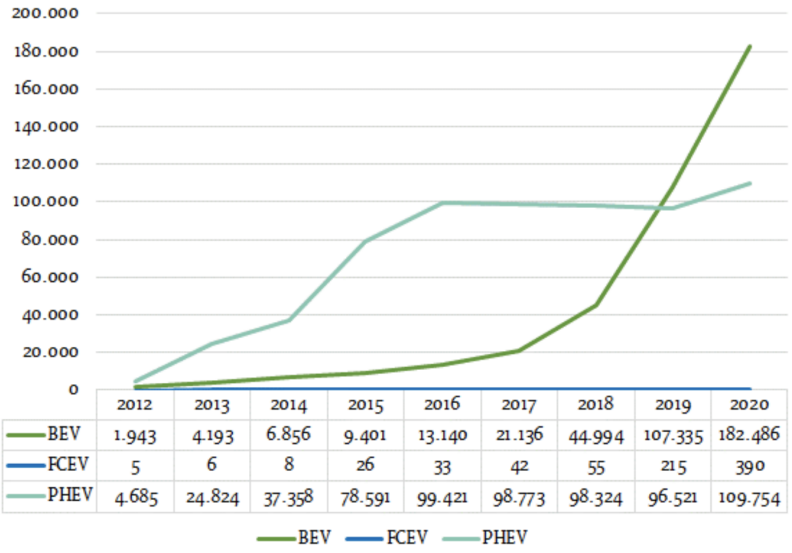
### 2.2.1 A changing environment

Power is delivered via grid components as cables and transformers (for electricity) to residential and industrial connections of our customers.
These grid components have to have the needed capacity. This is monitored by Grid Planners and if needed transformers are swapped for heavier ones, or cable connections are strengthened.
For decennia this monitoring was done by mainly looking to historical yearly extremes.

However, the increasing growth rate of power demand and supply driven by emerging technologies and electrification of transportation and heating, require a more frequent and pro-active electricity grid management.

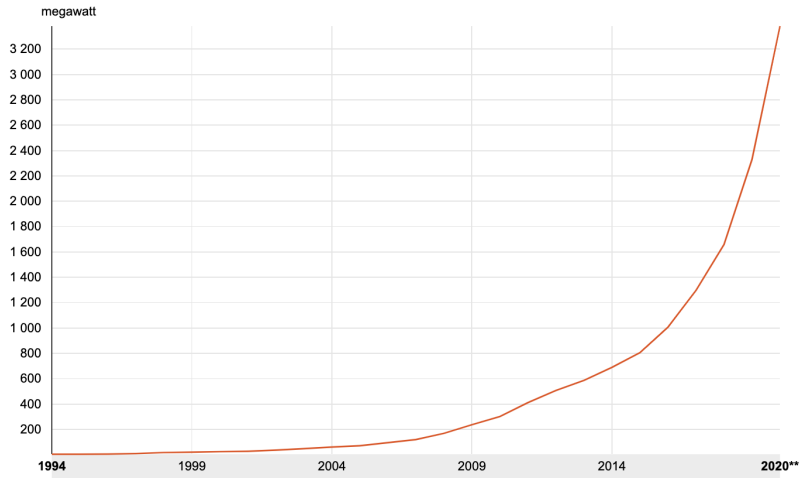



Solar power (PV) in The Netherlands (source: CBS).




| | 2012 | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 |
|---|---|---|---|---|---|---|---|---|---|
| BEV | 1.943 | 4.193 | 6.856 | 9.401 | 13.140 | 21.136 | 44.994 | 107.335 | 182.486 |
| FCEV | 5 | 6 | 8 | 26 | 33 | 42 | 55 | 215 | 390 |
| PHEV | 4.685 | 24.824 | 37.358 | 78.591 | 99.421 | 98.773 | 98.324 | 96.521 | 109.754 |

Battery (B), Fuel Cell (FC) and Plugg-in Hybride (PH) Electric Vehicles (EV) in The Netherlands (source: RVO).




Installed power of heat pumps (HP) in The Netherlands (source: CBS).

### 2.2.2 Impact on grid management

The more volatile and more rapidly increasing power flows require that Grid Planners more often monitor if the minimum and maximum power is still withing the transformer's capacity. Otherwise, this will cause unsafe operating situations and it can lead to power outages.

Grid Planners expect a big increase of the number of transformers that reach their operating capacity in the upcoming years. This in a environment were they are expected to make sure that Enexis is an enabler of the energy transition.

### 2.2.3 Opportunity for data science

Since a few years Enexis started to deploy so called Distribution Automation Light Boxes. With these measurement apparatus it is possible to monitor our distribution transformers near real time.

At the moment of writing 11k distribution transformers (of the total population of 35k) are equipped with these apparatus which are measuring 15 minute average voltages, currents and powers of each transformer.

This data (together with transformer metadata) enables to automatically detect or even foresee earlier overloading of transformers. It gives grid planners the opportunity to timely mitigate upcoming issues.

## 2.3 Data Mining Goals

### 2.3.1 Primary objective

More volatile power flows require a monitoring tool that forecasts transformer overloading.

The goal is to timely identify future overloading of transformers.

### 2.3.2 General description based on an example case

The project should result in a tool that is able to predict in autumn that a transformer will be overloaded in spring due to EV with a prediction interval.

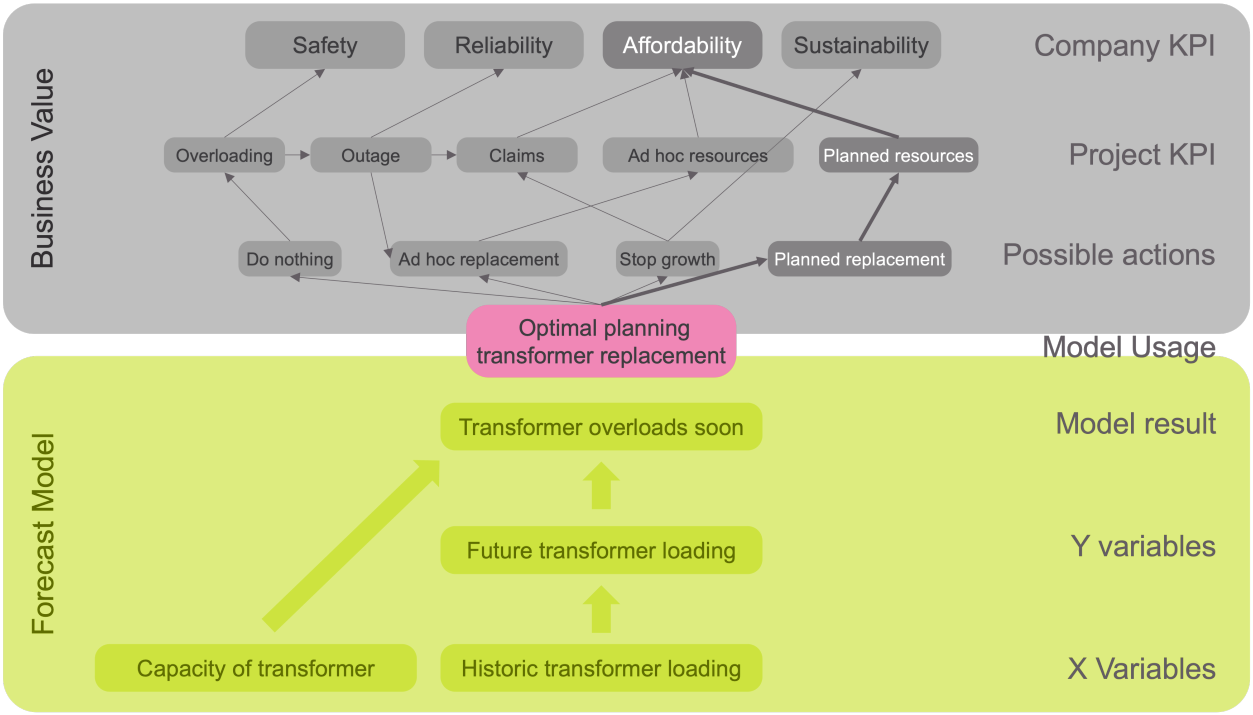
The tool enables better planning of grid strengthening which prevents overloading of transformers (safety and reliability), foreseeing future work (costs) and enabling the energy transition better (sustainability).

### 2.3.3 Business Value Diagram

Business value is created by the forecast model by enabling the preferred path for grid investments on the business value part (upper half) of the Business Flow Down Diagram below.

This path is possible by providing grid planners with a probability that a transformer will overload in the foreseeable future.

The model (lower half) will use historic DALI timeseries data to forecast transformer loading. By comparing this forecast with capacity of the transformer possible overloading can be foreseen.




Business Flow Down Diagram for the project.

### 2.3.4 Requirements from users

The developed product is only valuable if it is use dby the end users, who are the Grid Planners. Therefore the following additional requirements and wishes are important:

- Grid Planners don't want another tool to log into/install..
- The presentation of results has to be quick (no long waiting times).
- Results have to scalable up to 35k transformers.
- The tool should be reliable (stable interface and accurate forecasts).
- The model should be explainable.
- The uncertainty of predictions should be available.
- Results should be available for further use (e.g. importable for load flow tools).
- The results should be sorted based on their urgency.
- Forecast can also be made with limited transformer data.
- Forecast horizon should be six months.

### 2.3.5 Success criteria

The project is a success if there is a tool that is being used by the Grid Planners that accurately forecasts overloading of transformers.

Usage of the tool can be measured by tracking users of the tool and by performing interview with the end users after deployment. Accuracy is assessed by comparing forecasts with measurements.

On more detail the project is a success if:

- The model forecasts prediction intervals.
- The working of the model can be explained clearly.
- The prediction intervals ranges are acceptable to the Grid Planners.
- The model can use prior information of the rest of the population if historic measurements are missing.
- The computational burden is acceptable.

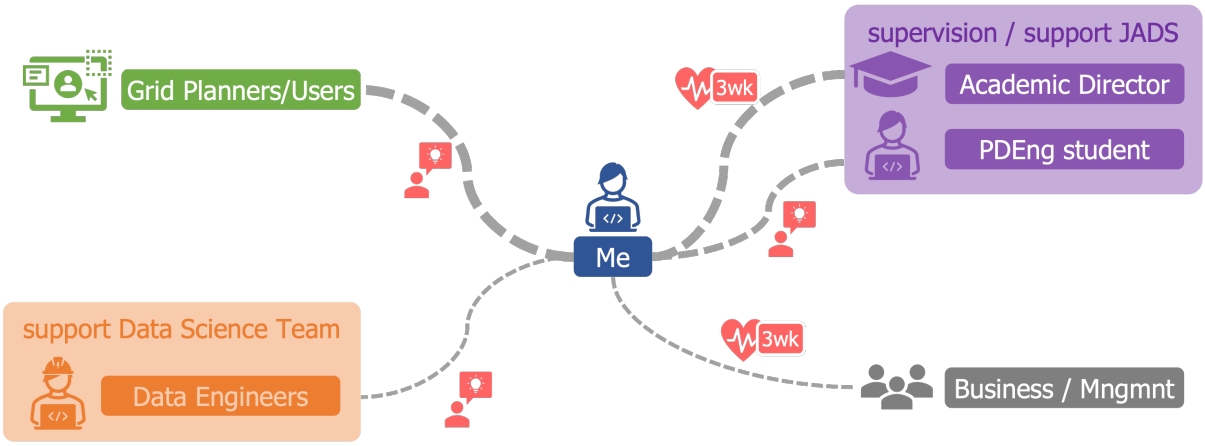
## 2.4 Project Plan

### 2.4.1 Organisation

The involved stakeholders are:

- Grid Planners: The end users of the tool. Ad hoc, they will be updated/asked for input/their expertise on the field of grid planning and their needs. They assess if the project is a success.
- Data Engineers: To get the tool into production the ICT guidelines and processes within Enexis have to be respected. Expertise of the Data engineers is crucial in the second half of the project to get things into production/deployment.
- Management: The direct manager enables Developer/Lead Bram to work two days a week on this project. Together with all other Enexis stakeholders they will be updated at the end of every sprint in the sprint review session.

- Academic supervision: Jeroen de Mast will be the academic director that monitors progress and the academic level. Every three weeks there will be an one-on-one meeting to discuss progress and issues.
- Academic support: PdEng candidate Akshaya Ravi of JADS is available for support on technical and academic issues. Together with a buddy from the Lead Track periodic meetings (every four weeks) will be planned, but also ad hoc issues will be discussed directly.




The Stakeholders with meeting frequencies (heartbeat / ad hoc).

### 2.4.2 Resources

The project will take place in June, July, September, October and November 2021.

- **Personnel**
  - Project lead / developer: Available 2 days per week.
  - Grid Planners: Available 2 hours per week.
  - Data engineers: Available 2 hours per week in October - November.
  - Supervision / support: Both available for a few hours every 3 weeks.
- **Data**
  - Data

### 2.4.3 Requirements

The tool should be implemented in such way that it is:

- Maintainable: The code has to have docstrings and unit tests.
- Scalable: The tool has to be able to process 35k transformers.
- Deployable: The tool has to be able to go in production according to the Enexis standards (Test-Acceptance-Production).

### 2.4.4 Assumptions

- DALI data is available without huge quality issues during the project
- Weekly extremes on transformer data is an acceptable aggregation level for capacity planning.
- Data Engineering has capacity for several hours a week for support between September and December.
- The computational burden for probabilistic models is no problem regarding the computational power available.

### 2.4.5 Risks and Contingencies

- **Lead / developer has just become a father (is technically up to September on parental leave) and bought a house that has to**
  - Mitigation: None.
- **Grid Planners are immensely occupied with the current challenges in the grid. Although not a lot of time is required, it mig**
  - Mitigation: Be clear and direct regarding expectations and communications and limit the effort and time for this project for Grid Planners without giving in on quality/input.
- **Data Engineers are also loaded with work and might not have time/resources available.**
  - Mitigation: It is essential to request capacity in the beginning of the project, although they will be involved only in the second half
- There is no use of sensitive data in this project regarding privacy (GDPR) or security. DALI data is allowed to be used. Credentials are not embedded in code and access to data sources is restricted by design.

### 2.4.6 Costs and Benefits

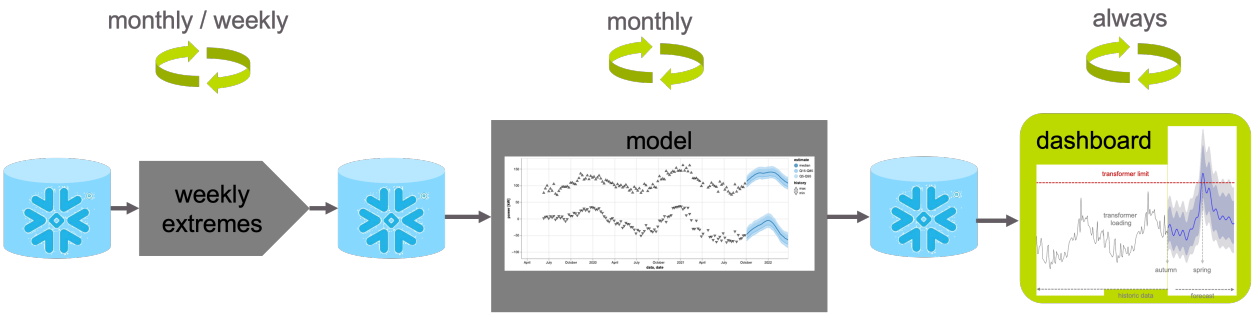
- Data collection: Data is available in an existing database. Only querying is needed, no active additional data collection.
- Implementation: Open source software is used besides already licensed applications. Only computation power will cost additionally. More details will be available after a first proof of concept.

### 2.4.7 Implementation concept

The stool is split into three steps to tackle the amount of data available, but still make the results manageable:

- Preprocessing: To condense 15 minute load averages into weekly extremes (minimum and maximum).
- Forecasting: To fit a model on the aggregated data and create forecasts.
- Dashboarding: To display (forecast) results to the end user.

In between steps results are stored in a Snowflake database.




The proposed steps for implementation of the tool.

### 2.4.8 Planned milestones

Table 1: Scheduled project milestones for 2021.

| week | CRISP-DM step | detail |
|---|---|---|
| 26 | Business Understanding | |
| 29 | Data understanding | Go / No Go |
| – | – | Summer break |
| 36 | Data Preparation | |
| 39 | Modeling | |
| 42 | Evaluation | Go / No Go |
| 45 | Deployment | |

CHAPTER

THREE

# DATA UNDERSTANDING

## 3.1 Initial Data Collection

The goal is to forecast accurately maximum and minimum loading of transformers. There is 15 minute average data available of 11k transformers. The maximum and minimum loading is not needed on this detailed level. According to the Grid Planners, weekly or monthly extreme details are more than enough.

For this first model to forecast transfer loading no external variables will be used as input for the model. Only historic data will be used (with metadata to extract de capacity of the transformer).

Data is available in Snowflake. in a Jupyter Notebook pandas with a SQLAlchemy engine is used for the data collection from Snowflake. Credentials are available via Vault in the Enexis domain.

### 3.1.1 Measurement data source

In a Snowflake database the following data is available:

Table 1: Snowflake data source details.

| | |
|---|---|
| Database | DB_CDWH_P |
| Schema | CDWH_4_BDM |
| Table | BDM_DALI_METINGEN |

### 3.1.2 Meta data source

In a Snowflake database the following metadata is available:

Table 2: Snowflake data source details.

| | |
|---|---|
| Database | DB_CDWH_P |
| Schema | CDWH_5_INF_MEETDATA_E View: DIM_DALI_BOX_VW |
| Table | DIM_DALI_BOX_VW |

## 3.2 Data Description

### 3.2.1 Measurement data details

The measurements are available since Q2 2018. Since then more and more transformers were equiped with DALI boxes and at the moment of writing (Q3 2021) a total number of 10,992 boxes are measured
Measurements on open doors (from safety perspective important) and currents, voltages and powers accumulated up to 89,052,020,404 records in the Snowflake table.

For this project we focus first on the active power on a transformer om the medium voltage connection side. The power is available for all three power phases as well as the sum of them.
Below are the fields of interest given that are queried from the aforementioned table.

Table 3: DALI table fields of interest.

| Field o.i. | Type | Example |
|---|---|---|
| BOXID | VARCHAR | ESD.000240-2 |
| CHANNELID | VARCHAR | register://electricity/0/activepower/sumli?avg=15 |
| WAARDE | DOUBLE | -3.408062 |
| DATUMTIJD | TIMESTAMPTZ | 2021-05-12 07:45:00.000000000 |

### 3.2.2 Measurement data details

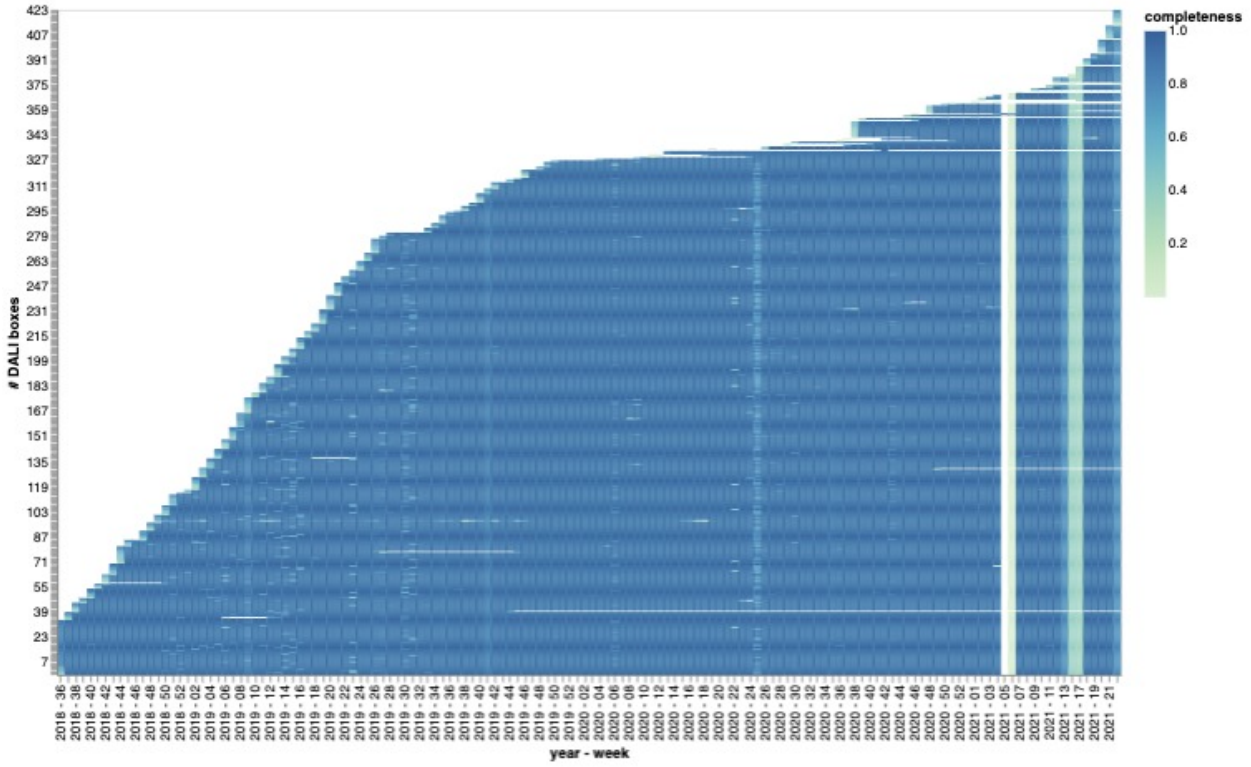
In the metadata table there are 15,058 records present.
Only 9,920 boxids are operational and have a nominal power correctly registered.

## 3.3 Data Exploration

The measurement data is up-to-date, but has missing values in two moments in time (both in 2021). This issue was discovered and is addressed. For the project it won't be a huge issue, since the data is aggregated to weekly extremes. This will cause extremes to be less extreme if (a lot of) data is missing. Only if data is missing for a whole week, there will also be missing data in the aggregated set. In both cases (outlier and missing data) the model can handle this.

For timeseries modelling it is advised to have at least two periods (years in this case) of measurement data. As the figure below shows, this might be a problem eventually. One of the challenges is therefore to explore if prior knowledge of the population can overcome this issue.

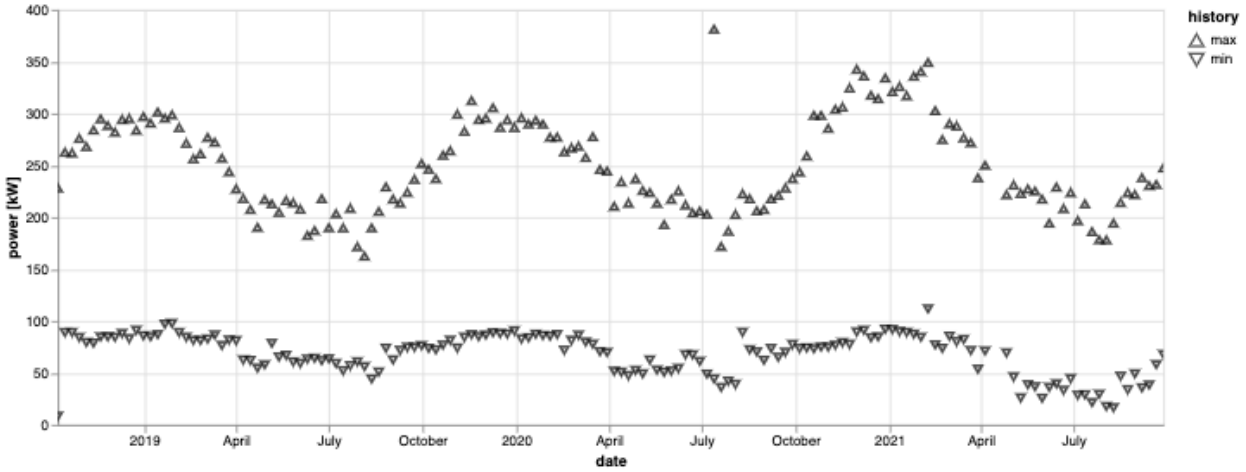
In the figure below the completeness of the data over time is given for the transformers in the area of Breda. On the vertical axis the transformers are shown ordered by the time they got operational. On the horizontal axis the time is given. In color the completeness is given per week: Darkblue (value 1) means that all data was present, towards white (value 0) means no data at all.




Completeness for DALI data in the service area of Breda.

## 3.4 Data Quality

Beside the missing data described above, the data quality (of the 15 minute power averages) seems like expected. The reason is probably that the 15 minute averaging already smooths out the extreme (short circuit) values and measurement errors. Although sometimes outliers can still be seen in the data (which can propagate into the weekly extremes as shown on the figure below). Taking not only the extremes, but also the second highest/lowest value per week for robustness did not make a lot of difference (probably also due to the aforementioned smoothing).




Example of weekly extremes with an outlier for the maximum in July 2020.

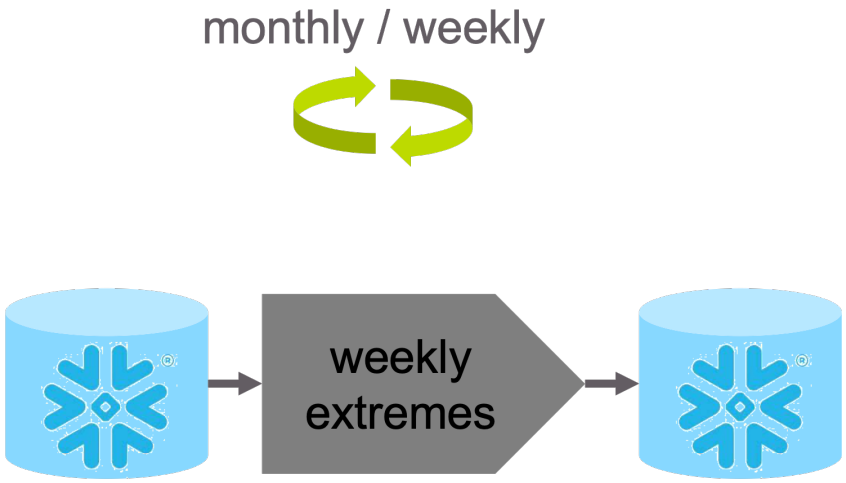
CHAPTER

FOUR

# DATA PREPARATION

The data preparation step focuses on converting raw measurement data with a frequency of 15 minutes into weekly extremes and preparing these extremes to be usable input for the forecasting model.

Initially the whole history is aggregated and stored in a Snowflake database. After that updates are done every month / week.




The data preparation step resulting in weekly aggregated data.

## 4.1 Data Selection

Columns described in Data Understanding are selected from the measurement () and metadata tables.

From the measurement data only data is selected from DALI boxes that have nominal power registered in the metadata table and that are in operation (`src.utils.snowflake.read_meta()`). Both data and metadata are needed to indicate future overloading and add value to Grid Planners.

As mentioned only the active power (P) on medium voltage side is used initially. Apparent power (S) is preferred (for a more fair comparison with the nominal power), but since this is 15 minute average data, this is hard to reconstruct.

In this step the separate power phases are selected and processed as well as the sum of the phases (`src.utils.snowflake.make_week_extremes_query()`).

Only the 15 minute average channels are selected for preprocessing.

## 4.2 Data Cleaning

No data cleaning is performed on the raw data before aggregation, but data is checked and cleaned on the following after reading it in (*src.preprocess.preprocess.load_data()*) and before being used by a model for forecasting:

- Data of extremes (minimum or maximum) having the value of zero in the beginning of the series are removed (*src.preprocess.preprocess.remove_leading_idling()*). This is for example the case if a DALI box is in operation, but its transformer is not.
- Only data is used that has a history of more than two years (*src.preprocess.preprocess.too_short()*). This will ensure in this stage that the seasonality (sub)model has enough data to tune on.
- Only data is used of transformers that have a measurement in their history with an absolute value higher than half the transformer capacity (*src.preprocess.preprocess.too_small()*).

Duplicate data (can only be created by updating the extreme table) is not an issue for the model and will not be eliminated.

Missing data is neither a problem for the model and is also not imputed.

## 4.3 Data Construction

From the raw 15 minute data the weekly minimum and maximum are determined. This is done per channel and boxid (*src.utils.snowflake.make_week_extremes_query()*). The week definition used is the ISO-week since this is always a full week.

A SQL query aggregates and writes the result asynchronously on the Snowflake database. This can be done in batch for all historic measurements (*src.utils.snowflake.create_week_extremes()*), but the created table can also be updated per week (*src.utils.snowflake.update_week_extremes()*).

Table 1: Snowflake table details for weekly extremes data.

| | |
|---|---|
| Database | DB_DATASCIENCE_P |
| Schema | DATASCIENCE_1_ETL |
| Table | DS_SPARK_DALI_WEEK_EXTREMES |

The fields of the table are listed below. The table is clustered by BOXID and L. The amount of rows is condensed from 89,052,020,404 to 3,457,856 records.

Table 2: Extremes table fields.

| Field | Type | Example |
|---|---|---|
| BOXID | VARCHAR | ESD.000240-2 |
| L | VARCHAR | sumli |
| YEAR | NUMBER | 2021 |
| WEEK | NUMBER | 53 |
| PROCESSED_ON | TIMESTAMPTZ | 2021-05-12 07:45:00.000000000 |
| MAX | DOUBLE | 678.90 |
| MIN | DOUBLE | 123.45 |

## 4.4 Data Integration

Since no additional data sources are used, no joins or merges are required.

## 4.5 Data Formatting

The model does not demand an order (e.g. by year and week) of the data. For the modelling stage the data is queried from the table in *Data Construction*

Consecutively, a date column is constructed from the ISO year and week format with day==1.

The extra columns period and model_var are assigned and filled with the values "history", "observed" respectively for measurement data. This is in preparation for long formatting and concatenating forecast results in a later stage (`src.preprocess.preprocess.format_data()`).

An example of the loaded extreme data is shown below

```
df_data
```

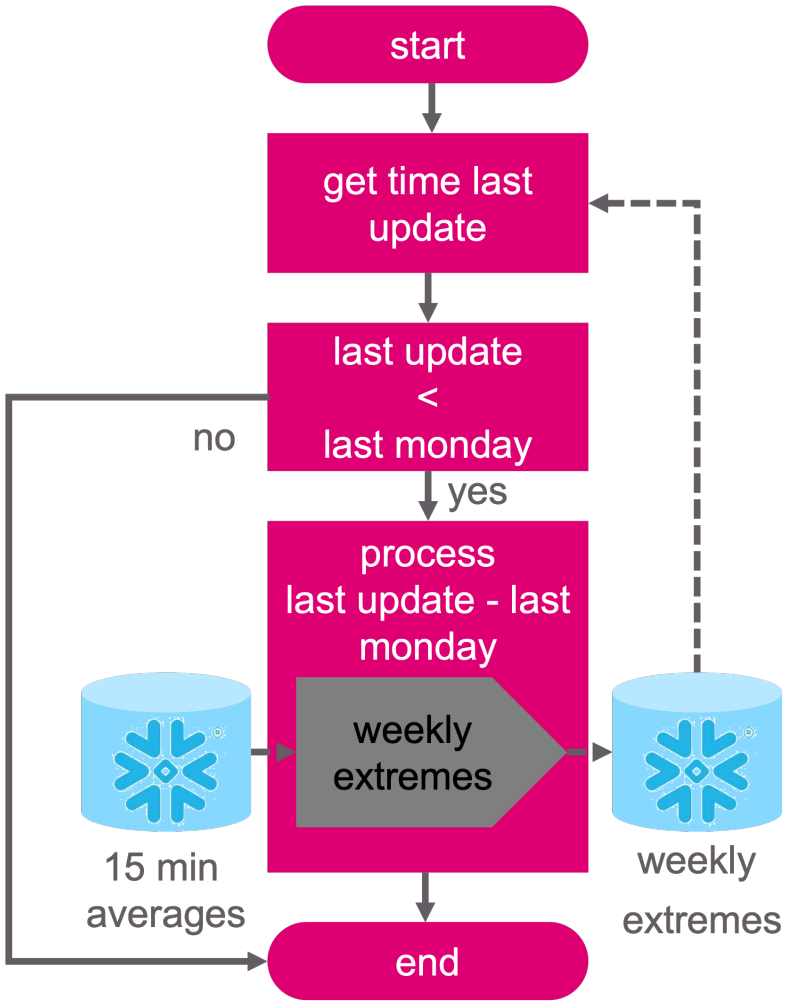
| | boxid | date | l | processed_on | week | year | extreme | value | period | model_var |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | VRY.CHOPS-1 | 2018-09-03 | sumli | 2021-10-07 07:03:50.602000+00:00 | 36 | 2018 | max | 145.486206 | history | observed |
| **1** | VRY.CHOPS-1 | 2018-09-10 | sumli | 2021-10-07 07:03:50.602000+00:00 | 37 | 2018 | max | 147.918304 | history | observed |
| **2** | VRY.CHOPS-1 | 2018-09-17 | sumli | 2021-10-07 07:03:50.602000+00:00 | 38 | 2018 | max | 154.419098 | history | observed |
| **3** | VRY.CHOPS-1 | 2018-09-24 | sumli | 2021-10-07 07:03:50.602000+00:00 | 39 | 2018 | max | 155.272598 | history | observed |
| **4** | VRY.CHOPS-1 | 2018-10-01 | sumli | 2021-10-07 07:03:50.602000+00:00 | 40 | 2018 | max | 169.614395 | history | observed |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **313** | VRY.CHOPS-1 | 2021-08-30 | sumli | 2021-10-07 07:03:50.602000+00:00 | 35 | 2021 | min | -223.345993 | history | observed |
| **314** | VRY.CHOPS-1 | 2021-09-06 | sumli | 2021-10-07 07:03:50.602000+00:00 | 36 | 2021 | min | -211.977295 | history | observed |
| **315** | VRY.CHOPS-1 | 2021-09-13 | sumli | 2021-10-07 07:03:50.602000+00:00 | 37 | 2021 | min | -209.037598 | history | observed |
| **316** | VRY.CHOPS-1 | 2021-09-20 | sumli | 2021-10-07 07:03:50.602000+00:00 | 38 | 2021 | min | -223.494995 | history | observed |
| **317** | VRY.CHOPS-1 | 2021-09-27 | sumli | 2021-10-07 07:03:50.602000+00:00 | 39 | 2021 | min | -173.239502 | history | observed |

The format of the loaded extremes data.

## 4.6 Data Updating Process

The weekly extremes can be updated every week (or longer).

By running `src.preprocess.update_extremes()` the function `src.utils.snowflake.update_week_extremes()` is called. This will will trigger the following steps which update the weekly extremes Snowflake table:




The detailed process to create and assess load forecasts.

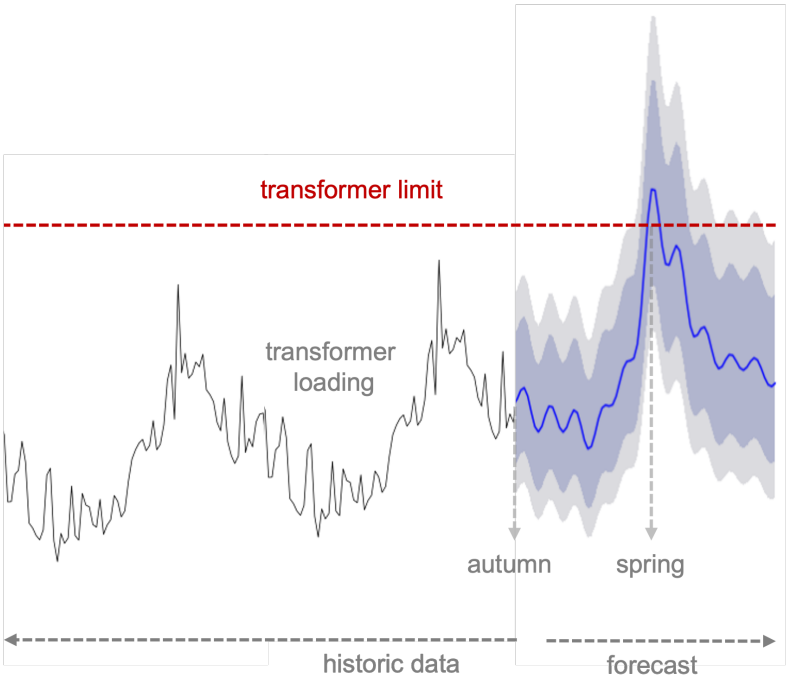
CHAPTER

FIVE

# MODELING

## 5.1 Modeling Technique

The Grid Planners are looking to data driven forecast with a uncertainty indication as mentioned before. This is visualized as a concept in the figure below.




A possible output of the model visualized.

### 5.1.1 Input Features

The Grid Planners want to have a result dat is based on real observed data. Or, to make it more explicit: Data that has been measured and not data from possible scenarios with a wide range of uncertainty.

Another suggestion was made to use (historic) weather data as an exogenous feature / input for the model. However, this is not data we have available with a reasonable confidence for the forecast horizon (in the future). And seasonality can also be captured from the observed data itself for different timeseries models.

Therefore, the model only uses historic DALI measurement data.

### 5.1.2 Fit-Predict Process

A model will be fitted / tuned for every DALI measurement extremes since every transformer has it's own unique signal (`src.forecast.forecast.determine_estimates()`).

A model is trained on observed data, a forecast is made and stored and finally the trained model is discarded (`src.forecast.forecast.forecast()`).


The weekly extremes are loaded for tuning the model. The model forecasts are afterwards stored in a Snowflake database.

The next time a forecast is required, new data is available and a new model will be fitted and used for forecasting.

### 5.1.3 Model Environment

A probabilistic approach has been implemented to fulfill the wish to forecast and display uncertainty rather than a point estimate.

From the main probabilistic toolboxes (STAN, EDWARD2, Pyro, TensorFlow2 Probability) PyMC3 was used for it's extensive documentation. A future step could be to transfer the model into TensorFlow2 Probability since PyMC3 is not the most recent toolbox anymore.

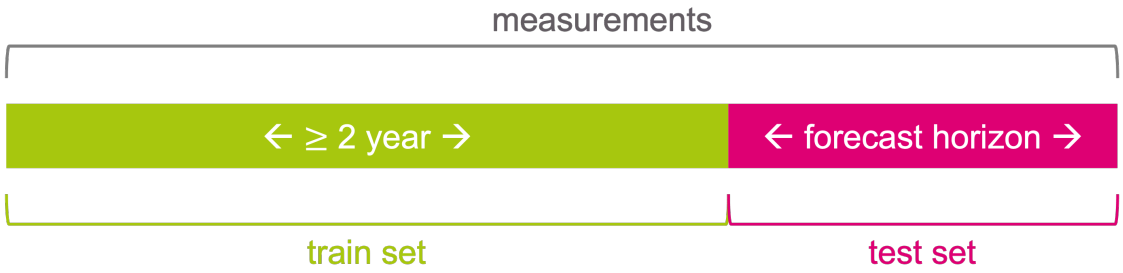
### 5.1.4 Generalized Additive Model Concept

To address the explainability of the model, a similar approach as Facebook Prophet is used. The model is a Generalized Additive Model (GAM) that consists of a trend / drift and seasonality component (and an error component). The translation to PyMC3 of Richie Vink was used as a basis.

The GAM approach makes it easy to decompose the different components of the timeseries and show the Grid Planners the effect of the drift and seasons separately.

## 5.2 Test Design

To evaluate the model the observed data is split into a train and test set based in the forecast horizon (`src.preprocess.preprocess.split_last()`). After splitting the train set is tested again for not being too short (at least two years: `src.preprocess.preprocess.too_short()`).




The split of measurement data into train and test data.

The test set is only used to validate forecasting results. The train set is used to train / fit / tune the model.

## 5.3 Model

### 5.3.1 Generalized Additive Model

The GAM model used is (`src.model.model.create_model()`):

$$\sigma_\epsilon \sim Uniform(lower = 0,\ upper = 1)$$

$$\Sigma \mid drift, yearly, \sigma_\epsilon = Normal(\mu = drift + yearly,\ sd = \sigma_\epsilon)$$

The additive naming is explicit in this notation.

The error component has a has a bandwidth of :math: *sigma_epsilon*.

#### Drift Component

According to the Grid Planners a increasing growth is more and more common due to the energy transition. Therefore, a stable drift model (`src.model.model.drift_model()`) is used that can mimic that. An exponential function resulted in divergence during the model tuning, but a second order taylor series makes the model convergent and stable.

The drift component model with a taylor series with the order of $n$ is described as:

$$X_{drift}(t) = [t^0, ..., t^n]$$

$$\beta_{drift} \sim Normal(\mu = 0,\ sd = 0.5)$$

$$drift \mid \beta_{drift} = X_{drift}(t)\ \beta_{drift}$$

For modelling a drift that has the described growth, a polynomial with order $n = 2$ is used.

### Yearly Component

Since the data has been aggregated into weekly extremes, the only seasonality to model is the yearly pattern. The yearly seasonality is modeled with $n$ order fourier series (`src.model.model.seasonality_model()`). This based on the work of Richie Vink.

The yearly seasonality model is described as:

$$X_{yearly}(t) = [cos(\frac{2\pi 1t}{T}), ..., sin(\frac{2\pi nt}{T})]$$

$$\beta_{yearly} \sim Normal(\mu = 0,\ sd = 1)$$

$$drift \mid \beta_{yearly} = X_{yearly}(t)\ \beta_{yearly}$$

Here the $T$ is the period of the seasonality in unit of time of the data. The is unit of time in case is a week for the data and a year in weeks is $T = 52.1775$. The order taken for the fourier series is $n = 5$.

### Enabling Forecasts

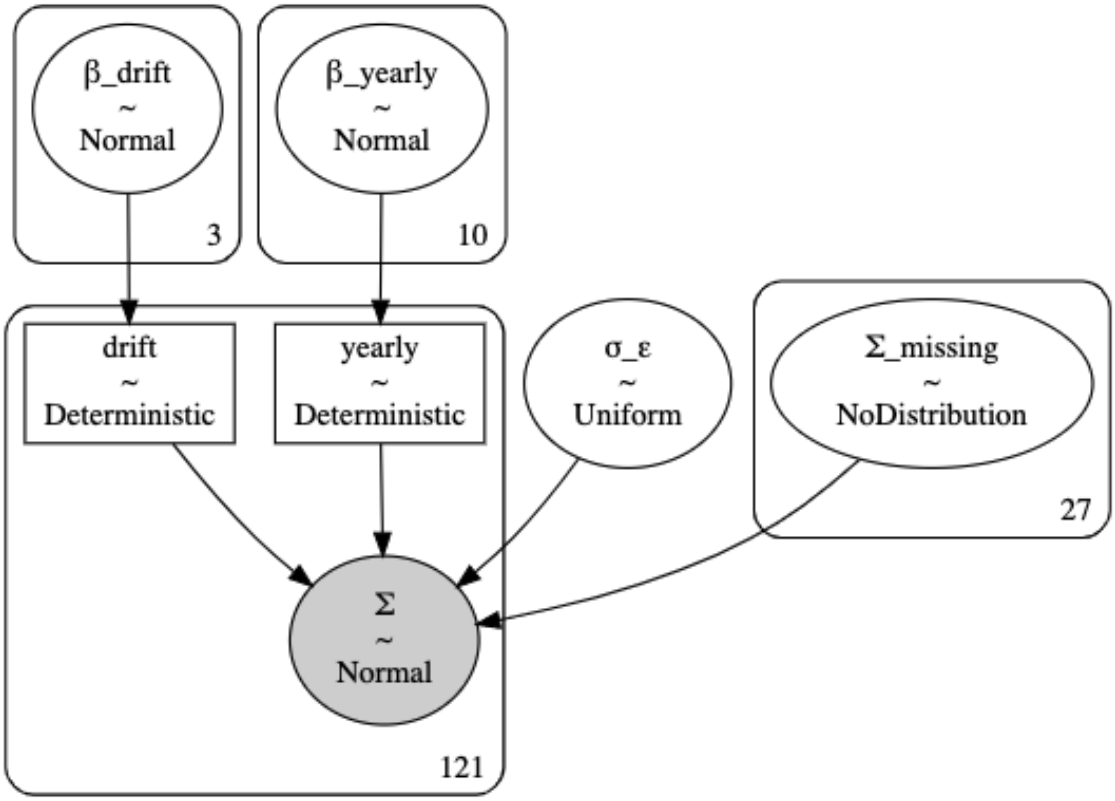
The model parameters ($\beta$)'s can now be tuned to produce the model is most likely to produce the observed (measurement) data.

To forecasting, the model also needs to produce beyond the timestamps it has been tuned on. The PyMC3 model can cope with this by feeding it with timestamps that are extrapolated for the forecasting horizon (`src.preprocess.preprocess.extrapolate_timestamps()`).

The matching observations (measurements) can be intentionally filled with NaN's. in the model PyMC3 will name them $\Sigma_m issing$. (This characteristic makes the model also robust against missing data).

By sampling the posterior predictive after tuning, also samples are generated for the extrapolated forecast timestamps (`src.forecast.forecast.determine_estimates()`).

### Total model Σ




The total model visualized.

Two separate GAM models Σ (*src.model.model.create_model()*) are used for the weekly minimum and maximum.

The visual above shows the total GAM model with a polynomial drift order $n = 2$ (the bias of order 0 explains $N+1 = 3$) and a fourier order of $n = 5$ (the sine and cosine parts explain $N * 2 = 10$).

The number of observations (weeks of measurements for this case) is 121 and the forecasting horizon is just more than six months (27 weeks).

### Formatting Results

From the model posterior predictive samples are drawn for all timestamps (also measurement timestamps, 1000 samples per timestamp).

From the posterior samples, the quantile bands are determined (`src.forecast.format.make_quantile_bands()`). This reduces the data storage.

The upper and lower limits of the quantile bands are then stored in the same format as the input (`src.forecast.format.format_model_estimates()`). The input of the model and the output are then concatenated together. This eases the visualization later.

```
df_total
```

| | boxid | date | l | processed_on | week | year | extreme | value | period | model_var | band | boundary |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | HVT.111153-1 | 2019-06-10 | sumli | 2021-10-07 07:03:50.602000+00:00 | 24 | 2019 | max | 78.797707 | history | observed | NaN | NaN |
| **1** | HVT.111153-1 | 2019-06-17 | sumli | 2021-10-07 07:03:50.602000+00:00 | 25 | 2019 | max | 91.941017 | history | observed | NaN | NaN |
| **2** | HVT.111153-1 | 2019-06-24 | sumli | 2021-10-07 07:03:50.602000+00:00 | 26 | 2019 | max | 100.725197 | history | observed | NaN | NaN |
| **3** | HVT.111153-1 | 2019-07-01 | sumli | 2021-10-07 07:03:50.602000+00:00 | 27 | 2019 | max | 83.487396 | history | observed | NaN | NaN |
| **4** | HVT.111153-1 | 2019-07-08 | sumli | 2021-10-07 07:03:50.602000+00:00 | 28 | 2019 | max | 91.487610 | history | observed | NaN | NaN |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **721** | HVT.111153-1 | 2021-08-30 | sumli | NaT | 35 | 2021 | max | 129.740717 | future | Σ | median | lower |
| **722** | HVT.111153-1 | 2021-09-06 | sumli | NaT | 36 | 2021 | max | 133.958580 | future | Σ | median | lower |
| **723** | HVT.111153-1 | 2021-09-13 | sumli | NaT | 37 | 2021 | max | 137.037553 | future | Σ | median | lower |
| **724** | HVT.111153-1 | 2021-09-20 | sumli | NaT | 38 | 2021 | max | 140.406872 | future | Σ | median | lower |
| **725** | HVT.111153-1 | 2021-09-27 | sumli | NaT | 39 | 2021 | max | 145.338946 | future | Σ | median | lower |

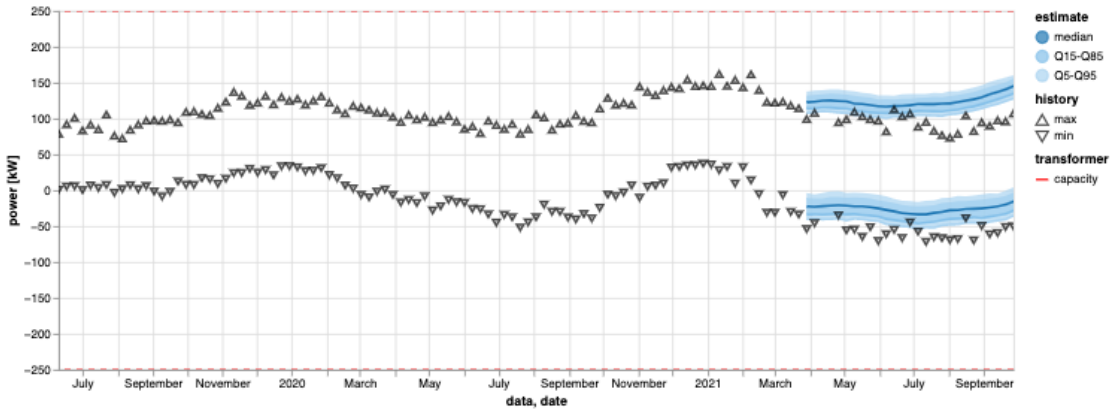
The concatenated input and result.

## 5.4 Model Assessment

The following model findings are most salient:

- **The model converges during tuning and gives feasible results.**
  - Exponential drift function tuning will not converge.
- **The computational burden on a CPU to tune and forecast both extremes is 1:24.**
  - CPU: 2 GHz Quad-Core Intel Core i5
  - RAM: 16 GB
- The model is fairly insensitive to outliers and missing data.
- The splitting of observations into train and test set works.
- The extrapolation with the forecasting horizon works.
- **An pure additive model may not be sufficient.**
  - Growth also increases the yearly component (see visualization below).
  - A pure multiplicative diverges.
  - A hybrid model (addition of a small fraction of a multiplicative model) might be an option.

A visualization of the results is shown in the figure below which shows most of the aforementioned points:




An visualization of the measurements (history) and forecast (estimates). Measurements from the train and test set are plotted.

### 5.4.1 Improvement suggestions

The following ideas could result in a better model:

- A hybrid additive-multiplicative model for dealing with the growing seasonality.
- **Adding a extra component to detect temporarily bypass switching of loads of other transformers.**
  - This could be implemented by estimating parameters of a rectangular function.
- **Making more recent observations more relevant for slowly changing loading patterns**
  - Possibilities are to mimic weights with pm.Potential or pm<distribution>(tau=weights).
- Use the population seasonality as a prior in case of a short history of observations.
- Use by-pass dummy model for outlier robustness.

CHAPTER

SIX

# EVALUATION

## 6.1 Results Evaluation

TO DO

### 6.1.1 Process Evaluation

TO DO

### 6.1.2 Next Steps

TO DO

CHAPTER

SEVEN

# API REFERENCE

This page contains auto-generated API reference documentation[1].

## 7.1 src

### 7.1.1 Subpackages

#### src.forecast

##### Submodules

#### src.forecast.forecast

##### Module Contents

src.forecast.forecast.**logger**

src.forecast.forecast.**config**

src.forecast.forecast.**determine_estimates**(*df_observed*)
Determine model parameters and forecast for a extreme (max or min).

**Parameters**

**df_observed** [pd.DataFrame] observed values of DALI data

**Returns**

**pd.DataFrame** estimates for extreme

src.forecast.forecast.**determine_estimates_minmax**(*df_observed*)
Determine model parameters and forecast for a extreme (max or min).

**Parameters**

**df_observed** [pd.DataFrame] observed values of DALI data

**Returns**

**pd.DataFrame** estimates for min and max

src.forecast.forecast.**forecast**(*boxid=None*)
Forecast for one or more DALI boxes.

[1] Created with sphinx-autoapi

**Parameters**

**boxid** [list] string or list with boxids to forecast for.

**Returns**

**pd.DataFrame** DataFrame with observations and forecast on long format

### src.forecast.make_forecasts

#### Module Contents

src.forecast.make_forecasts.**logger**

src.forecast.make_forecasts.**make_forecasts**(*clear=False*)
Make forecasts for all boxes.

**Parameters**

**clear: bool** Clear the Snowflake table

**Returns**

**None**

### src.model

#### Submodules

### src.model.format

#### Module Contents

src.model.format.**logger**

src.model.format.**make_quantile_bands**(*df_base*, *samples*, *quantiles=5, 15, 50, 85, 95*)
Translate samples to bands with edges defined by the given quantiles and merge them to a base dataFrame

**Parameters**

**df_base** [pd.DataFrame] base DataFrame with timestamps/length in accordance of the samples shape

**samples** [np.array] the posterior predictive samples

**quantiles: list** an iterable with the edges of the bands (0,1), ordered increasingly.

**Returns**

**pd.DataFrame** long format version with the quantile bands of the samples

src.model.format.**format_model_estimates**(*df_base*, *pp*, *quantiles=5, 15, 50, 85, 95*)
Format the samples into quantile bands for every model variable.

**Parameters**

**df_base** [pd.DataFrame] base DataFrame with timestamps/length in accordance of the samples shape

**pp** [dict] per model variable (key) the posterior predictive samples

**quantiles: list** an iterable with the edges of the bands (0,1), ordered increasingly.

**Returns**

**pd.DataFrame** long format version with the quantile bands of the samples for every model variable

## src.model.model

### Module Contents

src.model.model.**logger**

src.model.model.**det_dot**(*a*, *b*)
Dot product for Theano.

The theano dot product and NUTS sampler don't work with large matrices. source: https://www.ritchievink.com/blog/2018/10/09/ … build-facebooks-prophet-in-pymc3-bayesian-time-series-analyis-with-generalized-additive-models/

**Parameters**

**a** [np.array]

**b** [tt.vector]

**Returns**

**np.array** dot product of the two.

src.model.model.**fourier_series**(*t*, *p=52.1775*, *n=5*)
Calculate fourier representation of t for a period and order.

Based on source: https://www.ritchievink.com/blog/2018/10/09/ … build-facebooks-prophet-in-pymc3-bayesian-time-series-analyis

**Parameters**

**t** [range] range to be used as input variable

**p** [float] period to use for the fourier orders

**n** [int] order of fourier series

**Returns**

**np.array** matrix fourier representation of t

src.model.model.**seasonality_model**(*t*, *p=52.1775*, *n=5*, *seasonality_prior_scale=1*)
Create seasonality model with fourier series.

Based on source: https://www.ritchievink.com/blog/2018/10/09/ … build-facebooks-prophet-in-pymc3-bayesian-time-series-analyis

**Parameters**

**t** [range] range to be used as input variable

**p** [float] period to use for the fourier orders

**n** [int] order of fourier series

**seasonality_prior_scale: float**

**Returns**

**pm.var** PYMC3 variable

src.model.model.**polynomial**(*t*, *n=4*)
Calculate polynomial representation of t for an order.

**Parameters**

**t** [range] range to be used as input variable

**n** [int] order of polynomial

**Returns**

**np.array** matrix polynomial representation of t

src.model.model.**drift_model**(*t*, *n=4*)
Polynomal drift/trend function for additive model.

**Parameters**

**t** [range] range to be used as input variable

**n** [int] order of polynomal.

**Returns**

**pm.var** PYMC3 variable

src.model.model.**create_model**(*t*, *y*, *p_fourier*, *n_fourier=5*, *n_polynomial=2*)
Create a PYMC3 GAM model with a trend/drift and a seasonal/yearly component.

**Parameters**

**t** [timestamps] input series of scaled timestamps

**y** [float] observed values

**p_fourier: float** scaled period of the timestamps to take for the fourier component (a year)

**n_fourier** [int] order of the fourier component

**n_polynomial: inbt** order of the polynomial component

**Returns**

**PYMC3 model context**

## src.plot

### Submodules

## src.plot.altair

### Module Contents

src.plot.altair.**plot_estimate**(*df*, *legend=True*)
Plot the load forecast transformer.

**Parameters**

**df** [pd.DataFrame] DataFrame with the columns date, lower, upper, forecast(Q10-Q190, median), extreme

**Returns**

**Altair chart**

src.plot.altair.**plot_observed**(*df*)
Plot the historic/observed load of a transformer.

**Parameters**

**df** [pd.DataFrame] DataFrame with like:

**Returns**

**Altair chart**

src.plot.altair.**plot_limits**(*df*)
Plot the capacity limits of a transformer as a ruler of as a line if metadata changes over time.

**Parameters**

**df** [pd.DataFrame] DataFrame with the limits to be plotted

**Returns**

**Altair chart**

src.plot.altair.**lightness_scale**(*factor*, *limits=['#bedef4', '#1f77b4']*)

src.plot.altair.**plot_base**(*df_data=None*, *df_meta=None*)
Plot one or more of the following: history, forecast, transformer limits.

**Parameters**

**df_data** [pd.DataFrame] DataFrame with the timeseries

**df_meta** [pd.DataFrame] DataFrame with the metadata

**Returns**

**Altair layer chart**

src.plot.altair.**plot_decompose**(*df*)
Plot decomposition of PYMC3 GAM model components by model variable.

**Parameters**

**df** [pd.DataFrame] data of estimates in long format

**Returns**

**alt.Chart** Altair plot

src.plot.altair.**plot_all**(*df_data*, *df_meta=None*)
Plot estimates, observed and limits

**Parameters**

**df_data** [pd.DataFrame] DataFrame with the timeseries

**df_meta** [pd.DataFrame] DataFrame with the metadata

**Returns**

**Altair layer chart**

### src.plot.format

#### Module Contents

src.plot.format.**format_limits**(*df_meta*, *df_data=None*)
Format the transformer limits for plotting.

**Parameters**

**df_meta: pd.DataFrame** DataFrame with the trasnformer limits.

**df_data: pd.DataFrame** Optionally, the historic preprocess to determine the time range for the limits.

**Returns**

**A DataFrame with the limits to plot.**

src.plot.format.**format_history**(*df_data*)
Format the historic for plotting.

**Parameters**

**df_data: pd.DataFrame** The historic preprocess to be plotted.

**Returns**

**A DataFrame with the historic preprocess to plot.**

src.plot.format.**format_forecast**(*df_forecast*)
Format the forecasted load.

**Parameters**

**df: pd.DataFrame**

**Returns**

**pass through input**

src.plot.format.**dummy_forecast**(*df*)
Create a dummy forecast with median and Q10-Q90 based on historic preprocess to test plotting function.

**Parameters**

**df** [pd.DataFrame] Historic preprocess to use as a basis

**Returns**

**pd.DataFrame** Forecast preprocess that can be used to plot with plot_forecast()

### src.preprocess

#### Submodules

### src.preprocess.preprocess

#### Module Contents

src.preprocess.preprocess.**logger**

src.preprocess.preprocess.**too_short**(*df_data*, *threshold=52*)
Check if number of entries is long enough.

**Parameters**

**df_data: pd.DataFrame** Data to check.

**threshold: int** Minimum lenght.

**Returns**

**bool** True if preprocess is long enough.

src.preprocess.preprocess.**too_small**(*df_data*, *capacity*, *threshold=0.25*)
Check if values of preprocess is greater than threshold x capacity of transformer.

**Parameters**

**df_data: pd.DataFrame** Data to check.

**capacity: int** Capacity of the transformer.

**threshold: int** Fraction of the capacity that should be reached.

**Returns**

**bool** True if preprocess exists that is greater than threshold x capacity.

src.preprocess.preprocess.**remove_leading_idling**(*df_data*, *capacity*, *threshold=0.01*)
Remove preprocess that was generated when DALI box was active but electrical connection was not.

**Parameters**

**df_data** [ps.DataFrame] Data to be cleaned.

**capacity: int** Capacity of the transformer.

**threshold: int** Fraction of the capacity that should be reached.

**Returns**

**pd.DataFrame** DataFrame without the idling preprocess points.

src.preprocess.preprocess.**load_data**(*boxid*)
Load preprocess and metadata for boxid and do preprocessing.

**Parameters**

**boxid: str** ID of DALI box.

**Returns**

**None | (df_data, df_meta)** If checks are OK, return DataFrames with historic and meta preprocess.

src.preprocess.preprocess.**format_data**(*df*)

src.preprocess.preprocess.**split_last**(*df_data*, *period=dt.timedelta(weeks=26)*)
Split the historic dataset into a train and test set a certain period from the end.

**Parameters**

**df_data: pd.DataFrame** Dataset to split.

**period: datetime** Period from the end to split from

**Returns**

——- df_train, df_test

src.preprocess.preprocess.**extrapolate_timestamps**(*df*, *horizon=dt.timedelta(weeks=26)*)
Extrapolate the data with timestamps for the future.

**Parameters**

**df** [pd.DataFrame] data to extrapolate.

**horizon** [dt.timedelta] the amount of time to extrapolate further.

**Returns**

**pd.DataFrame** Only the extrapolated/added part.

### src.preprocess.update_extremes

### src.utils

#### Submodules

### src.utils.logger

#### Module Contents

src.utils.logger.**init_logging**()
Initialize logger.

**Returns**

**logger** object

### src.utils.parser

#### Module Contents

src.utils.parser.**parse_config**(*abs_path*) → EnvYAML
Parse the configuration in EnvYAML format.

src.utils.parser.**parse_args**() → dict
Parse the command-line arguments.

### src.utils.preprocess

#### Module Contents

src.utils.preprocess.**downcast**(*s*, *try_numeric=True*, *category=False*)
Downcast a series to the lowest possible memory type.

**Parameters**

**s** [pd.Series] Series to downcast.

**try_numeric: bool** If True it will try to read strings as numeric values.

**category: bool** If True (string) objects will be cast as a category.

**Returns**

**Downcasted series.**

src.utils.preprocess.**map_labels**(*series*, *kind='categorical'*, *labels=None*, *backwards=False*, ***arg*)

Map a Series values by the labels given.

**Parameters**

**series: pd.Series** Series to map on.

**kind: str** Indicator for kind of preprocess in series. With kind of {"categorical", "ordinal"} the mapping is applied, otherwise not.

**labels: dict** Defines with the mapping {key_0: value_0, etc.}.

**arg:** Additional arguments.

**Returns**

**pd.Series** Series with mapped values.

**class** src.utils.preprocess.**MinMaxScaler**(*upper=1*, *lower=- 1*)

MinMax Scaler like in sklearn, prevents total library import/dependency.

Initialize scaler with upper and lower boundary.

**Parameters**

**upper** [float] upper boundary to scale to

**lower** [float] lower boundary to scale to

**fit**(*self*, *X*, *y=None*)

Get fit parameters.

**Parameters**

**X** [np.array] preprocess to fit on

**y** [None] solely for consistency

**Returns**

**self** instance with self.min, self.max defined.

**transform**(*self*, *X*)

Scales preprocess according to fitted parameters.

**Parameters**

**X** [np.array] preprocess to scale

**Returns**

**np.array** scaled preprocess

**fit_transform**(*self*, *X*, *y=None*)

Execute consecutively self.fit and self.transform.

**Parameters**

**X** [np.array] preprocess to scale

**y** [None] solely for consistency

**Returns**

**np.array** scaled preprocess

**inverse_transform**(*self*, *X*, *y=None*)
Scale back to original domain.

**Parameters**

**X** [np.array] preprocess to scale

**y** [None] solely for consistency

**Returns**

**np.array** scaled preprocess

## src.utils.snowflake

### Module Contents

src.utils.snowflake.**logger**

src.utils.snowflake.**config**

src.utils.snowflake.**channel_like = register://electricity/0/activepower/%?avg=15**

src.utils.snowflake.**column_details =**

(BOXID VARCHAR(50), L VARCHAR(5), YEAR NUMBER(4), WEEK NUMBER(2), PROCESSED_ON TIMESTAMP_TZ, MAX DOUBLE, MIN DOUBLE)

src.utils.snowflake.**format_connection**(*name*)
Get (connection) details in the right format for different tables.

**Parameters**

**name** [name of the different sources]

**Returns**

**Connection**

src.utils.snowflake.**read_meta**(*boxid=None*)
Read meta preprocess of DALI box.

**Parameters**

**boxid** [list] ID of DALI box to read. None results in all available preprocess of DALI boxes that have nominal power specified.

**Returns**

**pd.DataFrame with metadata.**

src.utils.snowflake.**make_week_extremes_query**(*boxid=None*, *last_processed=dt.datetime(2001, 1, 1)*)
Build the query to request week extremes.

**Parameters**

**boxid** [list] If not None: select specific boxes.

**last_processed: datetime** Date to determine week extremes from. Cuts are always made on the last monday.

**Returns**

**str** Query string.

src.utils.snowflake.**create_table_query**(*query*)
Make query to create or replace table and insert preprocess of select query.

**Parameters**

**query** [str] Select query to be used.

**Returns**

**str** New query

src.utils.snowflake.**insert_table_query**(*query*)
Make query to insert preprocess of select query in an existing table.

**Parameters**

**query** [str] Select query to be used.

**Returns**

**str** New query

src.utils.snowflake.**create_week_extremes**()
Execute query to create whole new table of week extremes asynchroniously.

src.utils.snowflake.**get_last_processed_time**()
Retrieve last time table update has been done.

**Returns**

**datetime** Last processing time.

src.utils.snowflake.**update_week_extremes**()
Update week extremes from last processing time upt o last monday.

src.utils.snowflake.**read_week_extremes**(*boxid=None*, *L=None*)
Read week extremes for a DALI box and phase.

**Parameters**

**boxid: list** If not None: A list with DALI box IDs to read.

**L: str** The phases to retrieve (sumli, L1, L2, L3)

**Returns**

**pd.DataFrame** Week extremes.

src.utils.snowflake.**clear_forecasts**()

src.utils.snowflake.**write_forecasts**(*df*)

src.utils.snowflake.**read_forecasts**(*boxid=None*)

### src.utils.vault

#### Module Contents

src.utils.vault.**get_secrets**(*connection*)
Get secrets from Vault.

**Parameters**

**connection: str** The application/connection to get secrets for.

**Returns**

**requested** **secrets**

src.utils.vault.**get_vault_secret**(*url*, *token*, *path*, *mount_point*, ***kwargs*)
Get vault secrets to certain path in a dictionary.

**Parameters**

**url** [str] Base URL of Vault server

**token** [str] Token to get access to Vault

**path** [str] Relative path to location of secret

**mount_point** [str] Mount point of address

**Returns**

**dict** Dictionary with keys and value of a secret

### 7.1.2 Package Contents

src.**init_logging**()
Initialize logger.

**Returns**

**logger** **object**

# PYTHON MODULE INDEX

## S

# INDEX

## P



## R



## S



## T



## U



## W