Streamlit Analyzer for Electricity Readings

Introduction

Streamlit is a useful tool for analyzing large sets of data with visual charts. I chose Streamlit to analyze electricity data due to its ease of use, real-time interactivity, and interactive visualizations. In this article I will discuss how I created a Streamlit application for analyzing data from electricity meters at the microgrid in Sunday Morning Resort boutique hotel.

Use Case

The Sunday Morning Resort microgrid is equipped with 23 electricity meters to record sensor readings of current, voltage, frequency, and power. The purpose of my application is to find anomalies and interruptions in these readings and determine which meters contribute most to the total electricity cost.

The application reads a large parquet file containing all the electricity readings from the microgrid, as well as electricity prices from the Entso-e API. Users can visualize selected electricity data in line charts and heatmaps for any chosen time interval.

The parquet file

The parquet file for analysis consists of electrical data between 1.11.2023 and 12.5.2024 resulting in over 20 000 000 rows of data. Each meter represents a location, such as a building or a car charger. Each meter has an L1 phase type, L2 phase type, and an L3 phase type. These phase types each have sensors for current, voltage, active power, apparent power, power factor, frequency, total active energy, and total active returned energy. Each meter also has a Total phase type for the total readings. The total readings are total current, total active power, total apparent power, total active energy, and total active returned energy.

The application makes two dataframes from this parquet file: a dataframe for the L1, L2, and L3 readings and a dataframe for the total readings.

Making the application

The application is written with Python 3.11 and its source code is available in this GitHub repository.

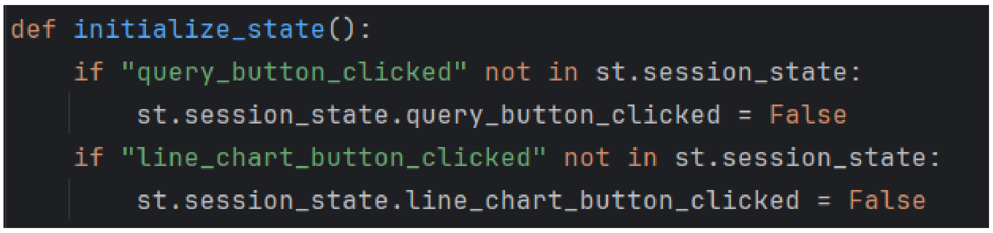
The application consists of two main files:

1. Main Logic File: Manages user interactions and navigation.
2. Feature Execution File: Contains the DataAnalyzer class, which has methods for listing columns, showing samples, describing, and making SQL queries to the chosen dataframe. It also includes methods for drawing line charts and heatmaps for selected sensors, and visualizing expenses for each electricity meter.

Session states

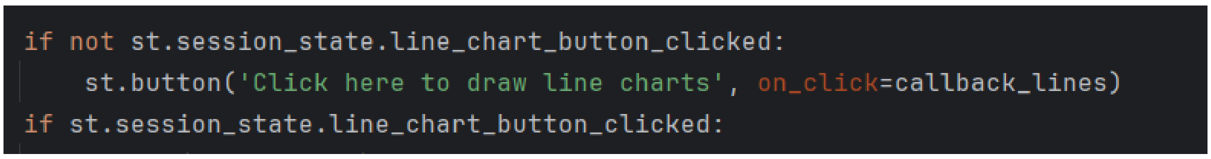
Streamlit applications run from top to bottom every time a change is made. Whenever the user presses a button, selects a radiobutton, or checks a checkbox, the application will re-execute in the same order in the app.py file.

To prevent variables from being reset to their original values, we need to use a session state. Every variable that determines what is displayed on the browser needs to be saved in a session state.



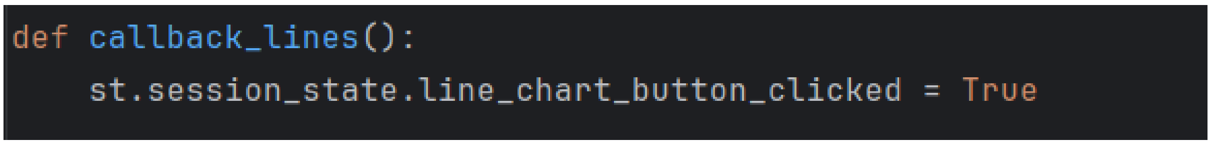
```
def initialize_state():
    if "query_button_clicked" not in st.session_state:
        st.session_state.query_button_clicked = False
    if "line_chart_button_clicked" not in st.session_state:
        st.session_state.line_chart_button_clicked = False
```

For example, here the Boolean variable line_chart_button_clicked determines whether the line chart logic will be rendered on the browser or not.



```
if not st.session_state.line_chart_button_clicked:
    st.button('Click here to draw line charts', on_click=callback_lines)
if st.session_state.line_chart_button_clicked:
```

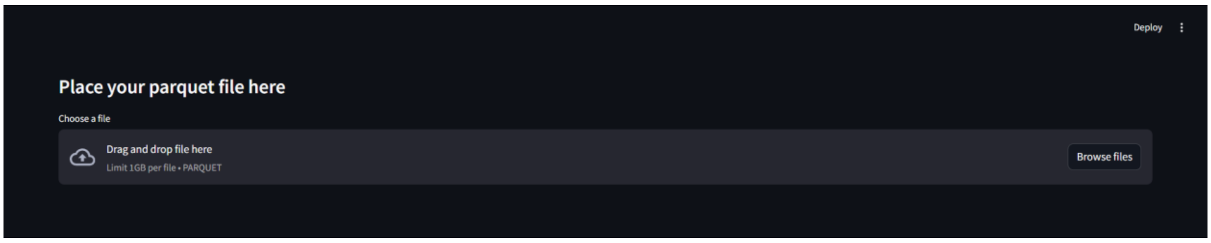
If the button is clicked, a callback function will be executed, resulting in a truthy value for line_chart_button_clicked and the rest of the code with the line chart logic will be executed.



```
def callback_lines():
    st.session_state.line_chart_button_clicked = True
```

Running the application

The application allows you to drag and drop a parquet file to upload.



Place your parquet file here

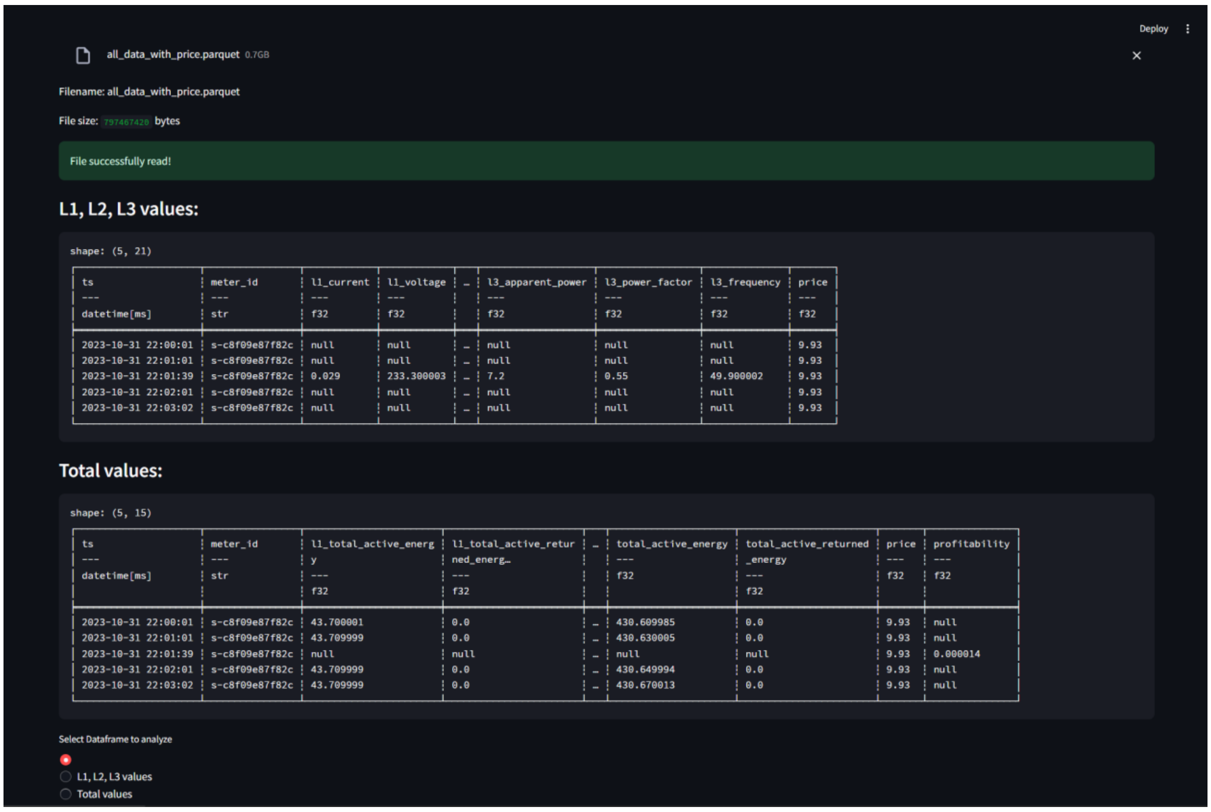
Choose a file

Drag and drop file here

Limit 1GB per file • PARQUET

Once the parquet file is uploaded, two Polars dataframes are created and displayed to the user. The first dataframe contains the readings from the L1, L2, and L3 sensors, while the second dataframe contains the readings from the total values. The second dataframe also has an additional expenses column, which is calculated by multiplying the total active power with the price.

$$Expenses = \left(\frac{€}{MWh}\right) * \left(\frac{1}{1000000}\right) * (W) = \left(\frac{€}{h}\right)$$



Deploy

all_data_with_price.parquet 0.7GB

Filename: all_data_with_price.parquet

File size: 797467420 bytes

File successfully read!

L1, L2, L3 values:

```
shape: (5, 21)
┌─────────────────────┬────────────────┬────────────┬────────────┬───┬────────────────────┬─────────────────┬──────────────┬───────┐
│ ts                  ┆ meter_id       ┆ l1_current ┆ l1_voltage ┆ … ┆ l3_apparent_power  ┆ l3_power_factor ┆ l3_frequency ┆ price │
│ ---                 ┆ ---            ┆ ---        ┆ ---        ┆   ┆ ---                ┆ ---             ┆ ---          ┆ ---   │
│ datetime[ms]        ┆ str            ┆ f32        ┆ f32        ┆   ┆ f32                ┆ f32             ┆ f32          ┆ f32   │
╞═════════════════════╪════════════════╪════════════╪════════════╪═══╪════════════════════╪═════════════════╪══════════════╪═══════╡
│ 2023-10-31 22:00:01 ┆ s-c8f09e87f82c ┆ null       ┆ null       ┆ … ┆ null               ┆ null            ┆ null         ┆ 9.93  │
│ 2023-10-31 22:01:01 ┆ s-c8f09e87f82c ┆ null       ┆ null       ┆ … ┆ null               ┆ null            ┆ null         ┆ 9.93  │
│ 2023-10-31 22:01:39 ┆ s-c8f09e87f82c ┆ 0.029      ┆ 233.300003 ┆ … ┆ 7.2                ┆ 0.55            ┆ 49.900002    ┆ 9.93  │
│ 2023-10-31 22:02:01 ┆ s-c8f09e87f82c ┆ null       ┆ null       ┆ … ┆ null               ┆ null            ┆ null         ┆ 9.93  │
│ 2023-10-31 22:03:02 ┆ s-c8f09e87f82c ┆ null       ┆ null       ┆ … ┆ null               ┆ null            ┆ null         ┆ 9.93  │
└─────────────────────┴────────────────┴────────────┴────────────┴───┴────────────────────┴─────────────────┴──────────────┴───────┘
```

Total values:

```
shape: (5, 15)
┌─────────────────────┬────────────────┬──────────────────────┬──────────────────────┬───┬─────────────────────┬───────────────────────┬───────┬───────────────┐
│ ts                  ┆ meter_id       ┆ l1_total_active_energ ┆ l1_total_active_retur ┆ … ┆ total_active_energy ┆ total_active_returned ┆ price ┆ profitability │
│ ---                 ┆ ---            ┆ y                    ┆ ned_energ…           ┆   ┆ ---                 ┆ _energy               ┆ ---   ┆ ---           │
│ datetime[ms]        ┆ str            ┆ ---                  ┆ ---                  ┆   ┆ f32                 ┆ ---                   ┆ f32   ┆ f32           │
│                     ┆                ┆ f32                  ┆ f32                  ┆   ┆                     ┆ f32                   ┆       ┆               │
╞═════════════════════╪════════════════╪══════════════════════╪══════════════════════╪═══╪═════════════════════╪═══════════════════════╪═══════╪═══════════════╡
│ 2023-10-31 22:00:01 ┆ s-c8f09e87f82c ┆ 43.700001            ┆ 0.0                  ┆ … ┆ 430.609985          ┆ 0.0                   ┆ 9.93  ┆ null          │
│ 2023-10-31 22:01:01 ┆ s-c8f09e87f82c ┆ 43.709999            ┆ 0.0                  ┆ … ┆ 430.630005          ┆ 0.0                   ┆ 9.93  ┆ null          │
│ 2023-10-31 22:01:39 ┆ s-c8f09e87f82c ┆ null                 ┆ null                 ┆ … ┆ null                ┆ null                  ┆ 9.93  ┆ 0.000014      │
│ 2023-10-31 22:02:01 ┆ s-c8f09e87f82c ┆ 43.709999            ┆ 0.0                  ┆ … ┆ 430.649994          ┆ 0.0                   ┆ 9.93  ┆ null          │
│ 2023-10-31 22:03:02 ┆ s-c8f09e87f82c ┆ 43.709999            ┆ 0.0                  ┆ … ┆ 430.670013          ┆ 0.0                   ┆ 9.93  ┆ null          │
└─────────────────────┴────────────────┴──────────────────────┴──────────────────────┴───┴─────────────────────┴───────────────────────┴───────┴───────────────┘
```

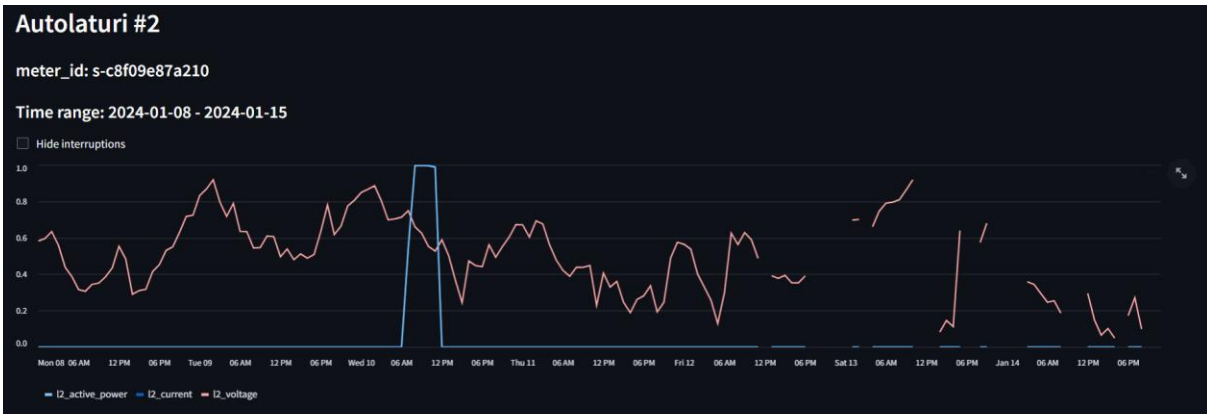
Select Dataframe to analyze

●
○ L1, L2, L3 values
○ Total values

The user can then choose which dataframe to analyze.

Line Charts for selected sensors

Streamlit is a good tool for finding relationships between different electricity readings because it can draw line charts with multiple lines over a time series. Here is an example of a car charger's L2 current, voltage, and active power values. The anomalies in the readings are easily spotted from peak values and interruptions. I also used MinMaxScaler to make it easier to spot any possible relationships between the different sensors.



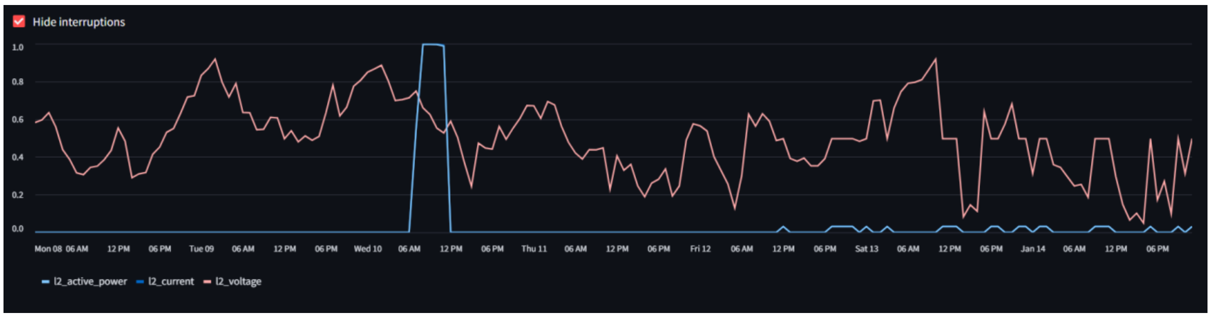
Autolaturi #2

meter_id: s-c8f09e87a210

Time range: 2024-01-08 - 2024-01-15

☐ Hide interruptions

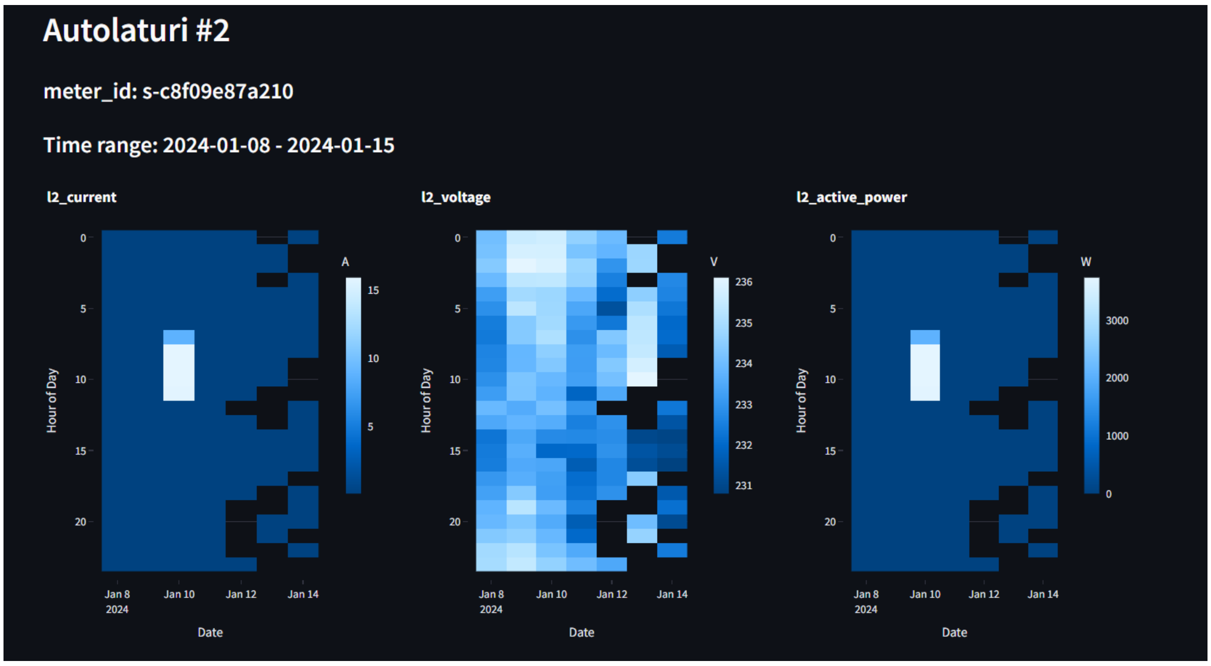
There is also an option to hide the interruptions in the readings to get a more consistent view of the chart. In this example, the missing values are replaced by the mean values of the readings.



Streamlit charts are updated dynamically, so changing the selected sensors or dates will automatically draw the line chart for the new sensors and dates.

Heatmaps for selected sensors

Similarly, heatmaps can be drawn for the selected sensors. What makes Streamlit user friendly, is that you don't need to select your electricity meter, date, or sensors again to draw the heatmaps either. Streamlit will automatically draw the heatmaps for the same sensors that you had already selected when drawing the line charts.



Autolaturi #2

meter_id: s-c8f09e87a210

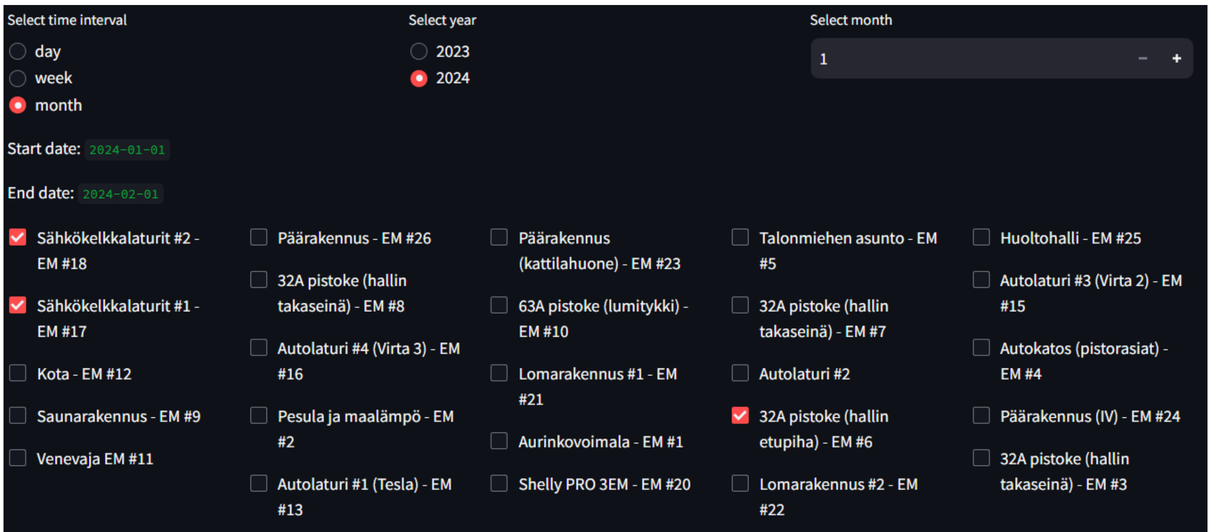
Time range: 2024-01-08 - 2024-01-15

Line Charts in power usage

Dataframes can be modified to switch between rows and columns of data. To view the expenses of different locations (meter_id), we need to pivot the original dataframe to make a new dataframe, where the meter_id values are the columns, and the expenses values are the new meter_id columns' values.

```
if not expenses_df.empty:
    # Pivot the dataframe to have a column for each location's expenses
    expenses_pivot_df = expenses_df.pivot_table(index='ts', columns='meter_id',
                                                values='expenses')
```

From the pivoted dataframe for expenses, we can view the line charts for each selected location in the microgrid. The total expenses and the total cost from the selected locations are automatically calculated as well.


Select time interval: day [✓ month] week

Select year: 2023 [✓ 2024]

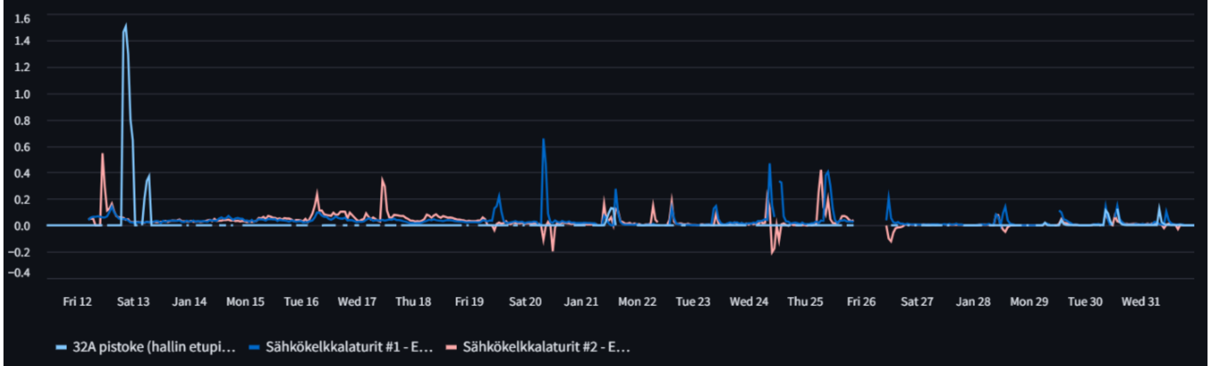
Select month: 1

Start date: `2024-01-01`

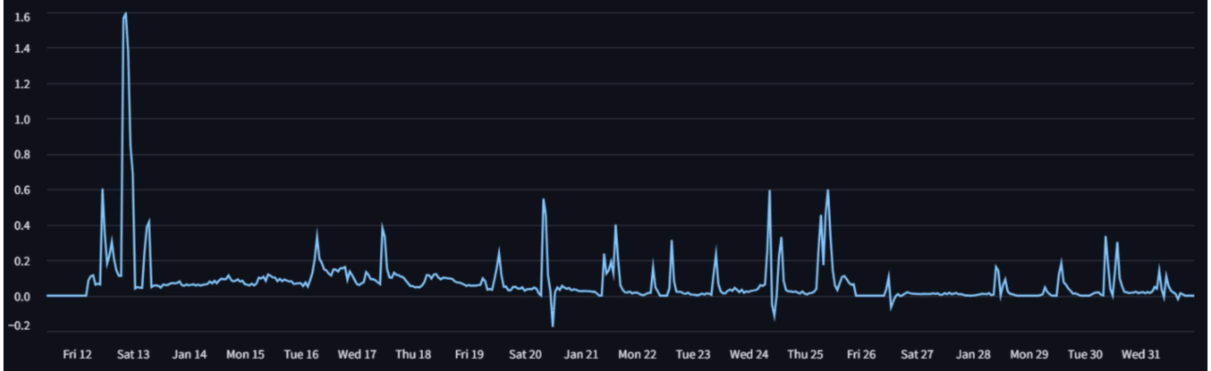
End date: `2024-02-01`

- [✓ Sähkökelkkalaturit #2 - EM #18]
- [✓ Sähkökelkkalaturit #1 - EM #17]
- Kota - EM #12
- Saunarakennus - EM #9
- Venevaja EM #11
- Päärakennus - EM #26
- 32A pistoke (hallin takaseinä) - EM #8
- Autolaturi #4 (Virta 3) - EM #16
- Pesula ja maalämpö - EM #2
- Autolaturi #1 (Tesla) - EM #13
- Päärakennus (kattilahuone) - EM #23
- 63A pistoke (lumitykki) - EM #10
- Lomarakennus #1 - EM #21
- Aurinkovoimala - EM #1
- Shelly PRO 3EM - EM #20
- Talonmiehen asunto - EM #5
- 32A pistoke (hallin takaseinä) - EM #7
- Autolaturi #2
- [✓ 32A pistoke (hallin etupiha) - EM #6]
- Lomarakennus #2 - EM #22
- Huoltohalli - EM #25
- Autolaturi #3 (Virta 2) - EM #15
- Autokatos (pistorasiat) - EM #4
- Päärakennus (IV) - EM #24
- 32A pistoke (hallin takaseinä) - EM #3

Net expenses (€/h)



Total net expenses (€/h)

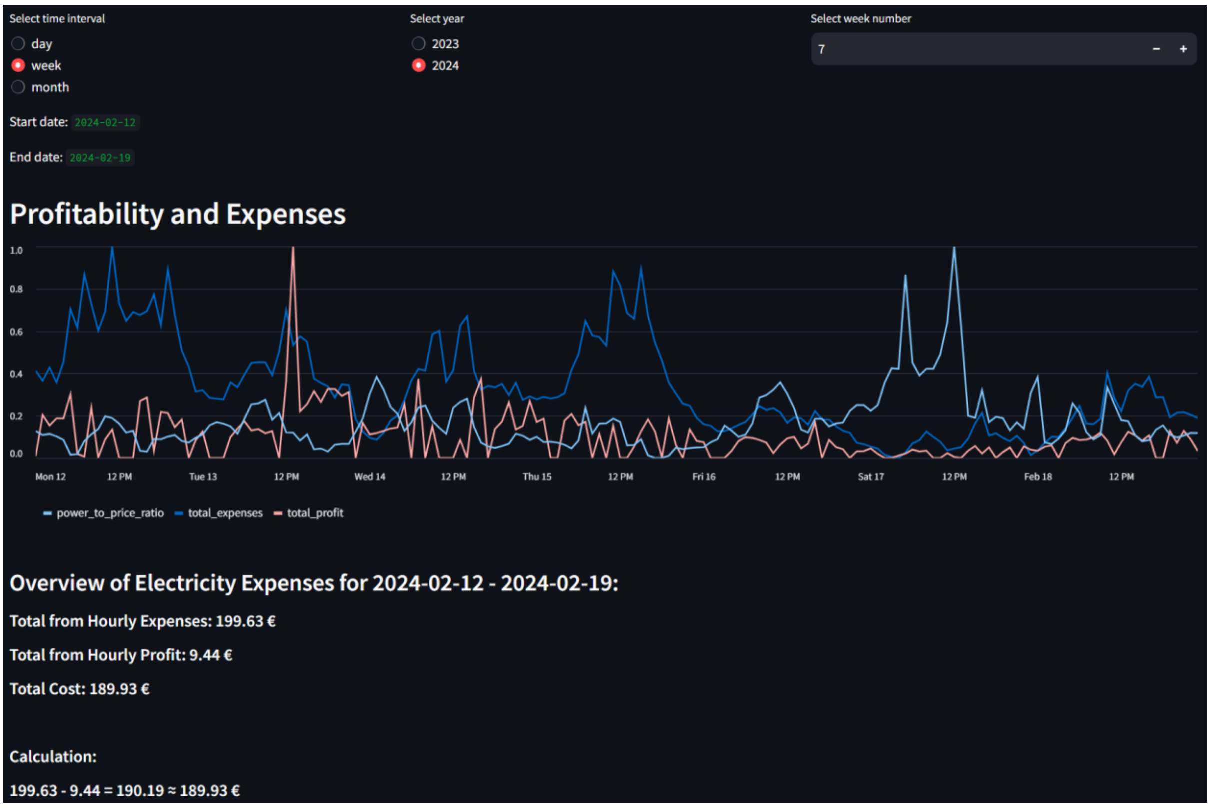


Total cost of electricity during 2024-01-01 - 2024-02-01:

38.08 €

Note that the expenses here are Net Expenses, meaning that negative values are also included. Negative values come from electricity production, such as the solar panel or charging the charger.

The Cost-effectiveness section draws the total expenses from all the electricity meters and the profitability from electricity production on the same line chart. There is also a third line that represents the cost-effectiveness of the power usage. It is the power / price ratio. The purpose of cost-effectiveness is to find when it is the ideal time to be using the most power. When the power to price ration is high, it is the ideal time to be using the most power.


Select time interval

day [✓ week] month

Select year

2023 [✓ 2024]

Select week number

7

Start date: 2024-02-12

End date: 2024-02-19

Profitability and Expenses

Overview of Electricity Expenses for 2024-02-12 - 2024-02-19:

Total from Hourly Expenses: 199.63 €

Total from Hourly Profit: 9.44 €

Total Cost: 189.93 €

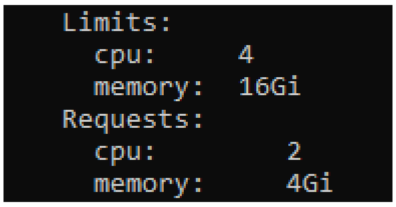
Calculation:

199.63 - 9.44 = 190.19 ≈ 189.93 €

Here the total cost is calculated from the actual values, not hourly values, and is therefore slightly different from the hourly values calculation.

Running the application in Rahti

Because of the dataframe’s large size, the application’s memory usage peaks at around 12 GB when loading the two dataframes for L1, L2, L3 values and Total values. The application is deployed at Rahti 2 and it uses one pod with 4 cpu’s and 16Gi of memory.


```
    Limits:
      cpu:      4
      memory:   16Gi
    Requests:
      cpu:         2
      memory:      4Gi
```

The application is deployed here and it can be tested with this parquet file. If the application does not respond after reading the file, it is likely that Rahti has run out of memory. If you refresh the page and read the file again, it should work because the memory is reset after every out of memory error.