BACHELOR

Energy Transition
Decreasing the dependency of households on the electricity grid

Wals, Veerle

Award date:
2023

Innovation Space Bachelor End Project

Energy Transition: Decreasing the dependency of households on the electricity grid

Quartile 3/4 - 2022-2023

Full Name	Student ID	Studies
Veerle Wals	1335669	BAM

Supervisor: Remco van der Hofstad

Eindhoven, July 10, 2023

Contents

1 Introduction

1.1 Description of situation

Worldwide, the need for renewable energy has grown immensely over the past years. It has become a standard for households to look into the option of investing in solar panels. However, this also brings some problems with it. A company in the Netherlands that focuses on creating electronics as smart mobility and semiconductor solutions regarding sustainability now also wishes to find a possible solution for a specific problem with solar panels; namely the problem focusing on the imbalance between supply and demand. They wish to be able to advise their clients on possible solutions to decrease this imbalance. This project focuses specifically on the difference in supply and demand regarding the solar energy, which causes a dependency on the electricity grid. The times at which solar energy supply is high differs from the times where demand for electricity is high. This means that households either have to get electricity from the grid in case of higher demand or put electricity back in the grid in case of a surplus. In case of a shortage, energy has to be taken from the grid which is not desired for energy usage but also for the cost the household pays for it. In case of a surplus, the energy has to be put back into the grid, which can cause an overload on the grid. This is not desired for the workings of the grid, but it might also come to the point where households would have to pay to put energy back into the grid. Thus, we want to reduce this dependency on the electricity grid.
This bachelor final project is done in collaboration with three other students, each focusing on their own expertise. The group consists of two mechanical engineers and one software scientist. There will be instances where the report will refer to insights from one of the others. The work can be combined, but can also be read all individually. The focus of this report is on creating a model for energy supply, which will then be used along with different demand scenarios. Finally, possible solutions for changing demand and supply are investigated and simulated to check their influence.

1.2 Assumptions and definitions

To simplify this problem, we will make some basic assumptions and make sure all definitions are clear. We will be looking at the electricity grid, since this takes up most of the energy in a household. We could also look at gas, especially for heating, but since in the future heating will most likely not be done with gas anymore, electricity is the most important energy usage in households of the future. Next to that, the type of renewable energy we will be focusing on is solar energy, supplied by solar panels:

Definition 1.1 In this project, the grid and the electricity grid are used interchangeably.

Definition 1.2 Dependency. It is desire to reduce the amount of electricity we take from the grid when there is a high demand, but also reduce the amount of surplus energy we put back into the grid. This means we will look at the total energy difference between supply and demand. For a year this equals: $\int_{x=0}^{365} |S_x - D_x| dx$, with S_x, D_x being the supply and demand respectively.

There are also some assumptions made to simplify the problem. These are defined below.

Assumption 1.1 This report will mostly focus on the possible solutions to decrease the dependency on the grid without looking at the economic impact of those solutions. We do not take the cost of solar panels into account. However, to be able to give a good advice to the company, in the end a few notes will be made about economic feasibility of possible solutions.

Assumption 1.2 Cable and conversion losses will be ignored.

Assumption 1.3 The assumption is made that electricity is the main part of the energy usage of a household, including heating. Thus, electricity and energy can also be used interchangeably.

Assumption 1.4 By a neighborhood, we assume approximately 30 households. The assumption is also made that these are wired to each other in such a way that both supply and demand can flow to all houses, such that there will be no issue in sharing energy.

1.3 Problem statement

What is a possible solution to reduce the electricity grid dependency of a neighborhood, a street of 30 houses, by 70%? In this report, the focus will be on reducing the dependency over an entire year. When investigating results, the company also desired to look into the timespan of a week. Since there is no average week, we take one week with very high supply and one with very low to have two examples.
This main question is divided into several sub-questions, corresponding to the sections in this report. They are formulated as follows.

Sub-question 1 on supply What model best describes and predicts the amount of solar radiation over time?
Sub-question 2 on supply How can solar radiation be used to compute the energy supply of a solar panel, used on a household?
Sub-question 3 on demand How can the electricity consumption of households be described?
Sub-question 4 on dependency What is the current state of the grid dependency of a household?
Sub-question 5 on dependency decrease What effect do possible solutions have on the grid dependency of a household?
Sub-question 6 on simulation In what way can we create the neighborhood of thirty households and investigate their total dependency?

In Section 1.4, we will state where in the report each sub-question will be answered.

1.4 Report structure

This report is organized as follows. Firstly, it will dive deeper into the supply of solar energy in Section 2, where sub-question 1 will be answered. To get to the supply, a mathematical model will be determined in Section 2.1.5 to describe the behaviour of solar radiation from which the supply of solar energy can be determined, answering the second sub-question. Next, we answer sub-question 3 by dividing the demand into three different scenarios in Section 3. Finally, in Section 4, a simulation will use the supply model and the different demand cases to determine the difference in supply and demand, i.e. the dependency on the grid. This answers sub-question 4. To answer sub-question 5, the simulation will also look into the effects of adding a storage solution and dynamic pricing. The focus will first be on household level, whereas after that it will be expanded to a neighborhood of thirty households in Section 4.5, answering the last sub-question. After this follow the discussion in Section 5 and conclusion in Section 6. Lastly, in the final Section 7 an advice will be given to the company on what possible next steps to take.

2 Solar energy supply

Focusing on the supply part of the problem, we aim to create a model that will return the solar energy supply a household obtains via its solar panels for 2023. For this, the solar panels use solar radiation that is then converted into energy. Section 2.1 will focus on obtaining a model for the solar radiation, while section 2.2 will then describe how this solar radiation is converted into energy by solar panels.

2.1 Modelling solar radiation

2.1.1 Data

When looking at the supply of energy, the focus is on the solar energy supplied to the households in our neighborhood. There is direct data about solar energy supply available from CBS. However, this data has two flaws when trying to work with it. One is that it is only available as the total energy supply in the Netherlands, which does not give a good overview of how much is supplied specifically to households. It is possible to divide this by the number of households in the Netherlands, but then other buildings using electricity, such as factories, will be disregarded. However, it is possible to divide the total by the number of households in the Netherlands to get some first insights; these should then always be higher than data focusing solely on households but should follow the same trends. The second and main problem however, is that this data is only measured in an average per month. This means there are very little data points to work with, so we cannot draw good conclusions from it. The data we will be working with is on solar radiation and temperature from KNMI ([11]). This data consists of hourly data points from KNMI in De Bilt on both solar radiation in Joules per cm^2 and temperature in 0.1 degrees Celsius.
The data from 2012 to 2022 on the solar radiation can be visualized as seen in Figure 1 below.

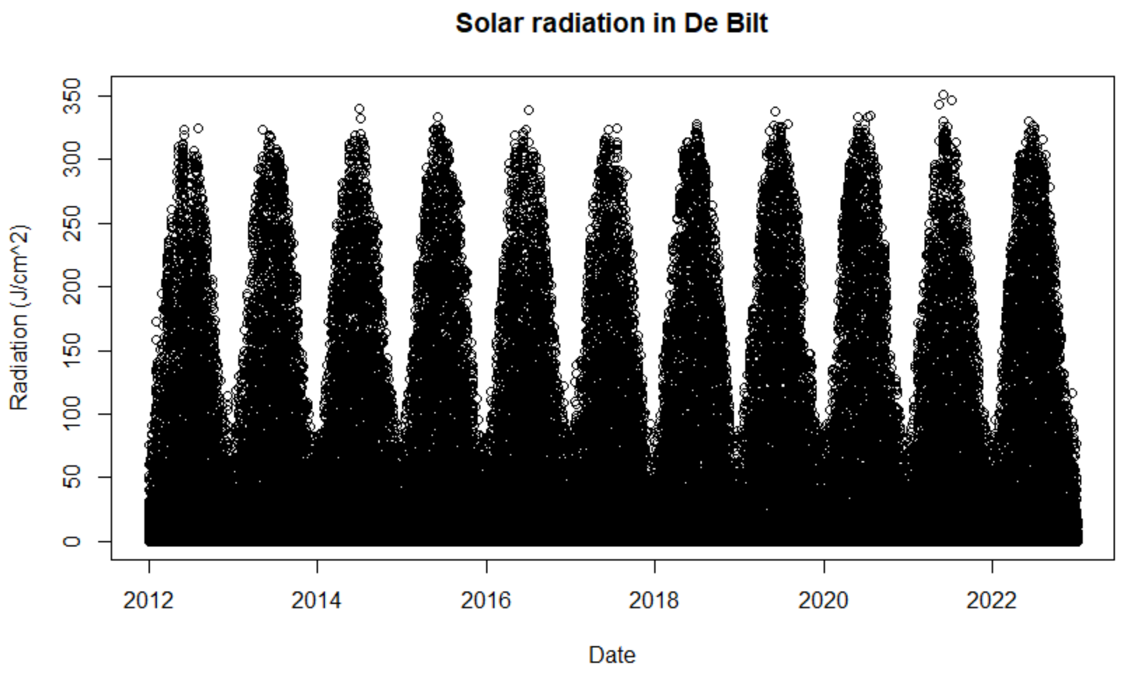

Figure 1: Solar radiation data from KNMI visualized in a scatter plot

For this solar radiation, we will create a regression model to describe past behavior, but also to predict the future. For the simulation in Section 4, we will focus on the entire year of 2023.

2.1.2 General information regression models

The aim is to create a regression model for solar radiation that can be used to obtain the energy supply from solar panels. This model is based on fluctuations over time, namely daily and yearly cycles, and temperate. Firstly, a model will be made only based on the time. After that, a regression model for the temperature will be created. This is done since temperature is used as an explanatory variable, but there is of course no temperature data available for the future. Therefore, we take two options; namely using this regression model for temperature to predict its values over 2023 or using the temperature data over 2022 as if it were the same for 2023. This extensive model for solar radiation based on both time and temperature will be described in section 2.1.5.
In the next subsections, we will use several variables and methods to create these regression models. The first basic model looks as follows:

$$Q_i = \beta_0 + \sum_{j=1}^{n} (\beta_j c_{i,j} + \gamma_j s_{i,j}) + \varepsilon_i, \tag{1}$$

where

$$\begin{aligned}
&Q = \text{solar radiation (Joule}/cm^2) \\
&c_{i,j} = \cos(\frac{2j\pi i}{T}) \\
&s_{i,j} = \sin(\frac{2j\pi i}{T}) \\
&T = \begin{cases} 3600 * 24 = 86.400, & \text{for periods of a day} \\ 3600 * 24 * 365 = 31.536.000, & \text{for periods of a year} \end{cases}
\end{aligned}$$

This model includes the use of both sine and cosine components. This is done because of the periodic behaviour of the solar radiation. However, it is also not perfectly periodic which requires the use, and combination, of more terms than simply one sine or cosine. For this mode, we do not include an assumption for the normality of the error terms. We would desire this assumption to hold, as both the p-value and the confidence intervals we will use are based on this assumption. Fortunately, in large sample sizes, violations of the normality assumptions on the error terms often do not have a noticeable influence on the results ([16]). Also, it is even stated that this is the least important assumption in linear regression ([10]). For this model, homoscedasticity and independence are of greater importance, which is why a residuals versus fitted plot will be investigated for the models.
We specifically want to be able to differentiate between the usage of periods of a year versus periods of a day. This is done by denoting the variables as cd_j and sd_j when using periods of a day and as cy_j and sy_j when using periods of a year. This results in the variables being defined as follows:

$$\begin{aligned}
cd_{i,j} &= cos(\frac{2j\pi i}{86400}), \\
sd_{i,j} &= sin(\frac{2j\pi i}{86400}), \\
cy_{i,j} &= cos(\frac{2j\pi i}{31536000}), \\
sy_{i,j} &= sin(\frac{2j\pi i}{31536000}).
\end{aligned}$$

The variable i is the time variable; a numerical version of the dates as found in the KNMI data. The dates consist of a day and time, with time steps of one hour. In the numerical version this results in time steps of 3600, namely the amount of seconds in an hour. This is also the reason for using the amount of seconds in the periods T. To incorporate more fluctuations, we add the sines and cosines with higher values of j. We use multiples of two times pi, since both one year and one day follow an entire cycle of two pi.
When looking at the performance of the model, but also on how many parameters of $c_{i,j}$ and $s_{i,j}$ to add, the adjusted R-squared, R^2_α, is taken into account.

$$R^2_\alpha = 1 - (1 - R^2)(\frac{n-1}{n-p}). \tag{2}$$

Here the variable n equals the total number of observations, while $n-p$ is the degrees of freedom for the residual term. The R-squared is calculated using the sum of squares decomposition

$$SS = SSR + SSE. \tag{3}$$

The total sum of squares is made up of the sum of squares for regression and the sum of squares for error. These are defined using observed data points, y_i, the mean of all data points, $\bar{y}$, and the data points calculated by the regression, $\hat{y_i}$. In case of an amount of n data points, these terms are defined as

$$\begin{aligned} SS &= \sum_{i=1}^{n}(y_i - \bar{y})^2, \\ SSR &= \sum_{i=1}^{n}(\hat{y_i} - \bar{y})^2, \\ SSE &= \sum_{i=1}^{n}(y_i - \hat{y_i})^2. \end{aligned}$$

Using these terms, the adjusted R-squared is calculated as

$$R^2 = 1 - \frac{SSR}{SS} \tag{4}$$

From this, the adjusted R-squared is calculated as denoted in equation 2. The adjusted R-squared is a number between 0 and 1, where generally speaking it is deemed better the closer it is to 1. Thus, for choosing the amount of variables c_i and s_i, the adjusted R-squared is an important factor. However, when adding a higher dimension, the R-squared generally speaking always gets higher. If it does not, it is easy to note that the variable should not be added. It is not desired to add too many parameters to complicate the model unnecessarily. So if it does not add significantly to the adjusted R-squared, one can choose not to add it.
Next to the adjusted R-squared not increasing, in some cases added variables simply aren't statistically significant. Using $\alpha = 0.05$, the significance can be concluded from the p-value; if the p-value is lower than 0.05 it is very unlikely that the estimator equals zero. When the estimator equals zero, it does not contribute to the model and is thus statistically not significant. In the case of a p-value below 0.05 we then state that the parameter is statistically significant. This can be used as a strategy to add onto the model as long as parameters stay statistically significant. Naturally, they should also add to the adjusted R-squared, otherwise the model gets unnecessarily complicated. Per regression model, the reasoning for adding or removing certain estimators will be described. In general, when removing variables, they are removed by following backwards elimination and running the code again after each removal. This means we remove variables in the order of p-values, starting with the variable with the largest p-value, as long as their p-values are above the threshold of $\alpha = 0.05$ ([1]). After the last removal, running the code again returns the final model with possibly adapted parameters for the remaining variables.
Regarding the statistical output of a regression model, the confidence interval will also be taken into account along with the estimation and statistical significance. Again using $\alpha = 0.05$, a confidence interval of 95% will be determined, which contains the estimate itself. The meaning of this interval is that there is a 95% chance that the value of this parameter is indeed in this interval. Ultimately, this confidence interval is desired to be as small as possible, such that the estimator used cannot deviate much from that value. The 95% confidence interval around a parameter β is calculated as

$$\hat{\beta} \pm t_{0.025,n-2}\sqrt{\frac{MSE}{\sum_{i=1}^{n}(y_i - \bar{y})}}, \tag{5}$$

with β being inside this interval. The MSE equals the mean squared error, which is defined as, with $n-p$ again being the degrees of freedom on the residuals,

$$MSE = \frac{SSE}{n-p}.$$

When zero is inside the confidence interval, we cannot state that the variable adds to the model with a parameter of something other than zero. Thus, from this it can also be concluded that the estimator is statistically not significant, which will also be noticeable by the p-value for statistical significance being above 0.05.

2.1.3 Regression model for solar radiation based solely on time

Firstly, a model for the solar radiation will be created that solely depends on the time variable. This is the simplest model to start off with; it would be highly desirable if the behaviour of solar radiation could be explained using the time variable only. We will investigate whether this is the case or if the extra variable on temperature is needed. Since solar radiation heavily depends on the season throughout the year, it could be expected to be periodic. In the picture of the visualized data above in Figure 1 this indeed seems to be the case. Thus, the usage of sines and cosines will play an important part. Specifically, we will be looking at linear regression using sine and cosine elements in our variables. The regression model will look as follows, with the value of n, or how many levels of sines and cosines get added, being chosen per model,

$$Q_i = \beta_0 + \sum_{j=1}^{n}(\beta_j c_{i,j} + \gamma_j s_{i,j}) + \varepsilon_i. \tag{6}$$

Two models will be created; one using periods of a day and one using periods of a year.
Using periods of a day, the model was created by adding on $c_{i,j}$ and $s_{i,j}$ until they were not statistically significant anymore. At the level j being equal to six, both the cosine and sine were not statistically significant. When taking the level from 1 to 5, this returns the estimated values in Table 1, with their corresponding confidence interval and statistical significance.

Model variable	Estimation of parameter	95% confidence interval	Statistical significance
(constant)	43.879	[43.55;44.21]	$< 2 * 10^{-16}$
cd_1	-58.967	[-59.43;-58.50]	$< 2 * 10^{-16}$
sd_1	27.607	[27.14;28.07]	$< 2 * 10^{-16}$
cd_2	15.648	[15.19;16.11]	$< 2 * 10^{-16}$
sd_2	-17.261	[-17.72;-16.80]	$< 2 * 10^{-16}$
cd_3	-1.270	[-1.73;-0.81]	7.2e-8
sd_3	0.111	[-0.35;0.57]	0.636
cd_4	0.435	[-0.03;0.90]	0.065
sd_4	1.321	[0.86;1.78]	$2.13 * 10^{-8}$
cd_5	0.542	[0.08;1.00]	0.022
sd_5	0.163	[-0.30;0.62]	0.490

Table 1: Statistical output for the model using periods of a day

Since the confidence intervals are quite small, the estimated values for the parameters can most likely not be too far off. After one by one removal of the statistically insignificant variables and running the code again to check whether any changes occur, the final regression formula equals

$$Q_i = 43.88 - 58.97cd_{i,1} + 27.61sd_{i,1} + 15.65cd_{i,2} - 17.26sd_{i,2} - 1.27cd_{i,3} + 1.32sd_{i,4} + 0.54cd_{i,5} + \varepsilon_i. \tag{7}$$

Regarding the model with periods of a year, there are only three statistically significant variables. For the sine and cosine levels j equal to 4 or 5, the parameters for both cosine and sine are not statistically significant, so these will not be taken into account at all. The statistical output for this model is given in Table 2

Model variable	Estimation of parameter	95% confidence interval	Statistical significance
(constant)	43.903	[43.49;44.32]	$< 2 * 10^{-16}$
cy_1	-38.611	[-39.20;-38.02]	$< 2 * 10^{-16}$
sy_1	-0.051	[-0.64;0.54]	0.864
cy_2	-0.144	[-0.73;0.44]	0.630
sy_2	-0.432	[-1.02;0.16]	0.149
cy_3	2.125	[1.54;2.71]	$1.3 * 10^-12$
sy_3	-0.188	[-0.78;0.40]	0.530

Table 2: Statistical output for the model using periods of a year

When using solely the three statistically significant variables, after backwards elimination of the statistically insignificant variables, running the code a final time results in

$$Q_i = 43.90 - 38.61cy_{i,1} + 2.12cy_{i,3} + \varepsilon_i. \tag{8}$$

Graphs for these two models compared to the actual solar radiation data can be seen in Figures 2 and 3 below.

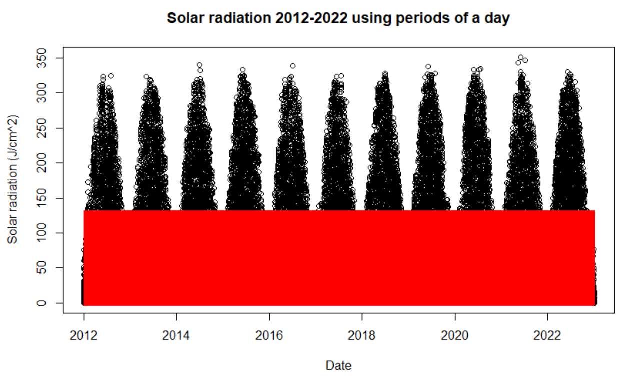

Figure 2: Visualization of model (7) versus data

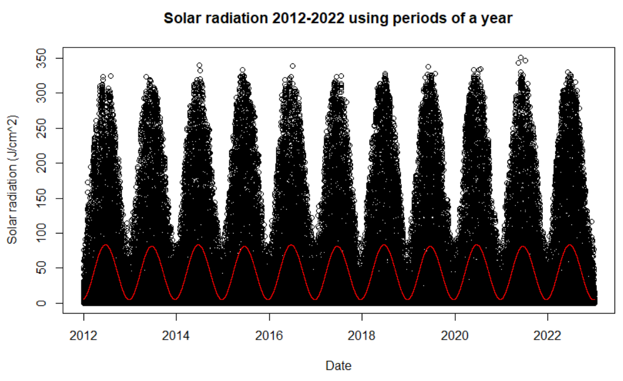

Figure 3: Visualization of model (8) versus data

Since the periods of a day cause the graph to look dense, we zoom in on 2022 to better visualize what is happening. This is shown in Figure 4.

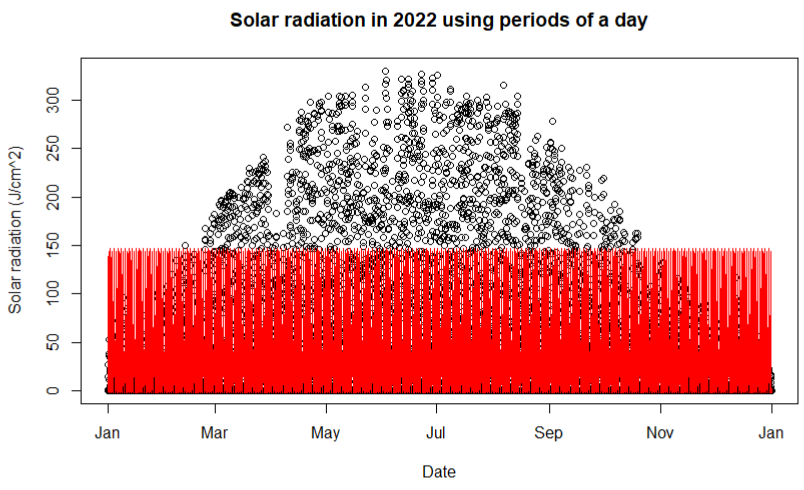

Figure 4: Visualization of model (7) over the year 2022 only

The adjusted R-squared of these regression models with periods of a day and periods of a year equal 0.4717 and 0.1474 respectively. This is in line with logical expectations, since the solar radiation fluctuates a lot every day, and being equal to 0 at night during the entire year. Thus, using periods of a day seems more reasonable. However, we note that now the peak does not fluctuate during the year, which is what happens in reality. This causes the need for a combination of both periods of a day and a year.

When combining both models, both cosine and sine variables are used for the two different periods. The general equation for this regression model is then formulated as

$$Q_i = \beta_0 + \sum_{j=1}^{n} (\beta_j cd_{i,j} + \gamma_j sd_{i,j} + \delta_j cy_{i,j} + \zeta_j sy_{i,j}) + \varepsilon_i. \tag{9}$$

In the model using periods of a day, the parameters were added starting at the level j equal to one and increasing it by one after every step. However, in this case, when j equals five, this results in three out of four variables not being statistically significant; only cd_5 is statistically significant. To make the decision on whether to add this, the adjusted R-squared is taken into account. It is concluded that the adjusted R-squared does not increase when adding this variable, so we choose not to add it into the model, as it unnecessarily complicates it. When a parameter for a variable is unequal to zero, it should always contribute to the (adjusted) R-squared to be added ([4]). For j equal to four, three out of the four variables are statistically significant. However, this only increases the adjusted R-squared by 0.0002, which does not change the adjusted R-squared when rounded to two decimals. Since it does increase the complexity of our model by adding three variables, we also choose not to add these. For j equal to three, both cosine variables are statistically significant and contribute to the adjusted R-squared, so these will stay. This generates the statistical output as denoted in Table 3.

Model variable	Estimation of parameter	95% confidence interval	Statistical significance
(constant)	43.903	[43.62;44.18]	$< 2 * 10^{-16}$
cd_1	-58.974	[-59.37;-58.58]	$< 2 * 10^{-16}$
sd_1	27.606	[27.21;28.00]	$< 2 * 10^{-16}$
cd_2	15.642	[15.25;16.03]	$< 2 * 10^{-16}$
sd_2	-17.263	[-17.66;-16.87]	$< 2 * 10^{-16}$
cd_3	-1.275	[-1.67;-0.88]	$1.9 * 10^{-10}$
sd_3	0.108	[-0.28;0.50]	0.589
cy_1	-38.618	[-39.01;-38.23]	$< 2 * 10^{-16}$
sy_1	-0.033	[-0.43;0.36]	0.869
cy_2	-0.151	[-0.54;0.24]	0.451
sy_2	-0.424	[-0.82;-0.03]	0.034
cy_3	2.138	[1.75;2.53]	$< 2 * 10^{-16}$
sy_3	-0.193	[-0.59;0.20]	0.335

Table 3: Statistical output for the combined model using periods of both a year and a day

There seems to be one variable, namely sy_2, for which the parameter is statistically significant, but with the value of 0.034. This is relatively high for a statistical significant variable with α being 0.05. After removing this variable, the adjusted R-squared does not decrease, so we decide not to unnecessarily add it in the model. After this decision, removing the statistically significant variables by backwards elimination and running the code again results in the formula of the model looking as

$$Q_i = 43.90 - 58.97cd_{i,1} + 27.61sd_{i,1} + 15.64cd_{i,2} - 17.26sd_{i,2} - 1.28cd_{i,3} - 38.62cy_{i,1} + 2.14cy_{i,3} + \varepsilon_i. \quad (10)$$

This combined model increases the adjusted R-squared to 0.619 and produces the plot as seen in Figure 5. Next to it, the residuals vs fitted plot is displayed in Figure 6.

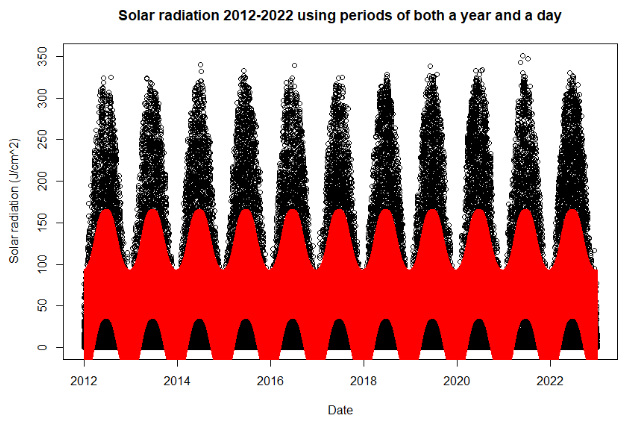

Figure 5: Visualization of model (10) versus data

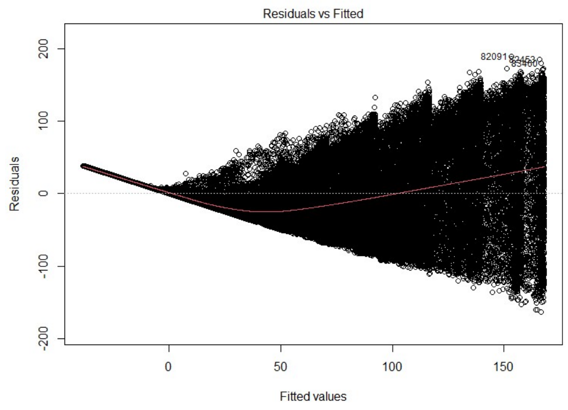

Figure 6: Residuals versus fitted graph for model (10)

Now there is indeed a combination of both yearly and daily fluctuation. However, it looks like the peaks should have a higher maximum, almost double of what they are now. Besides that, the lowest values now also go below zero which cannot happen in real life. They do not seem to go far below zero, but it is still not desired. When looking at the residuals versus fitted graph, it can also be seen that the variability increases as the fitted values increase. It would be desired for these points to be randomly scattered around the identity line. This heteroscedasticity indicates the need for a change in the variables or possibly an extra explanatory variable. A solution we will look into later is to add the temperature variable onto the current model.

2.1.4 Regression model for temperature

A regression model will now be created for the temperature. This will be used in predicting the solar radiation, by having a model for the predicted temperature values instead of data points, which are not available for the future. It may also be useful when computing the solar energy conversion, since the conversion rate decreases when the temperature is high ([15]).
The same hourly KNMI data as before will be used, but taking the temperature variable into account instead of solar radiation. Again, we will make use of a linear regression model with the same combined sine and cosine components. Since this regression is heavily influenced by all the hourly data points, while the temperature change every hour does not matter that much, we also take daily averages of this data. For this same reason, only periods of a year are used.

The model for this is then

$$T_i = \beta_0 + \sum_{j=1}^{n} (\beta_j cy_{i,j} + \gamma_j sy_{i,j}) + \varepsilon_i. \tag{11}$$

Firstly looking at the regression using all data points, the model gets quite extensive. All variables using j from one up to and including seven are statistically significant and contribute to the adjusted R-squared. When j equals eight, the adjusted R-squared does not increase so we do not add these $cy_{i,8}$ and $sy_{i,8}$ variables. In Figure 7, it can be seen what this model looks like compared to the data points. However, we will only investigate the results for the model using average daily temperatures, since that is most relevant for doing future predictions on solar radiation based on this model.
For this model using daily averages, at the level j equal to eight, both variables are not statistically significant. Even though the model gets a bit extensive, we will investigate the possibility of adding all statistically significant variables if they also increase the adjusted R-squared. Below an overview of the statistical output of this model can be found.

Model variable	Estimation of parameter	95% confidence interval	Statistical significance
(constant)	109.891	[108.92;110.87]	$< 2 * 10^{-16}$
cy_1	-61.563	[-62.94;-60.19]	$< 2 * 10^{-16}$
sy_1	-39.561	[-40.94;-38.18]	$< 2 * 10^{-16}$
cy_2	2.321	[0.94;3.70]	0.001
sy_2	4.570	[3.19;5.95]	$9.21 * 10^{-11}$
cy_3	1.340	[-0.04;2.72]	0.057
sy_3	-0.389	[-1.77;0.99]	0.580
cy_4	0.345	[-1.03;1.72]	0.624
sy_4	1.076	[-0.30;2.45]	0.126
cy_5	2.577	[1.20;3.96]	$2.52 * 10^{-4}$
sy_5	5.329	[3.95;6.71]	$4.29 * 10^{-14}$
cy_6	3.160	[1.78;4.54]	$7.26 * 10^{-6}$
sy_6	2.656	[1.28;4.03]	$1.6 * 10^{-4}$
cy_7	3.434	[2.05;4.81]	$1.1 * 10^{-6}$
sy_7	3.618	[2.24;5.00]	$2.77 * 10^{-7}$

Table 4: Statistical output for the model on temperature using daily averaged data

To simplify the model, we attempt to remove the statistically significant variable with the highest p-value after removing all the insignificant variables via backwards elimination. This, however, results in a decrease of the adjusted R-squared, thus it will still be included in the model. The regression formula using all statistically significant variables results in the equation

$$\begin{aligned} T_i =&109.89 - 61.56cy_{i,1} - 39.56sy_{i,1} + 2.32cy_{i,2} + 4.57sy_{i,2} + 2.58cy_{i,5} + 5.33sy_{i,5} + 3.16cy_{i,6} + 2.66sy_{i,6} \\ &+ 3.43cy_{i,7} + 3.62sy_{i,7} + \varepsilon_i. \end{aligned} \tag{12}$$

This regression model using daily averages is displayed in Figure 8. The first model returns an adjusted R-squared of 0.602; for model (12) using daily averages this increases to 0.733.

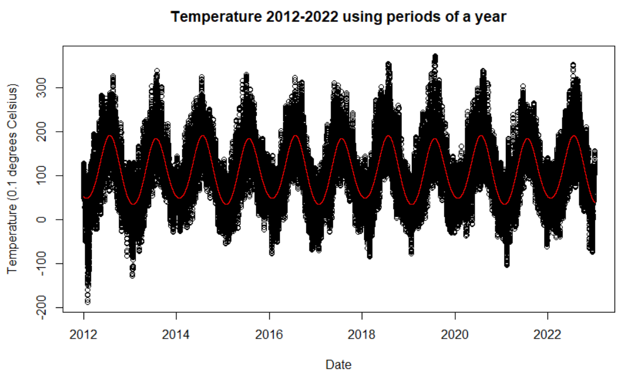

Figure 7: Visualization of linear regression model using all data points

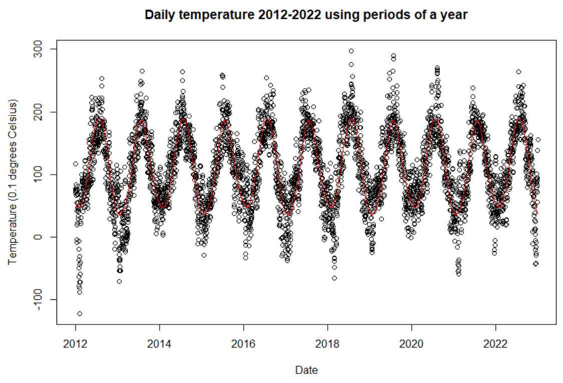

Figure 8: Visualization of model (12) versus averaged data points

When producing the residuals versus fitted plot for model (12), we note that it is scattered randomly around the horizontal line, which is as desired. We cannot find a pattern in the behaviour of the points. Thus, the model for temperature seems to be reliable.

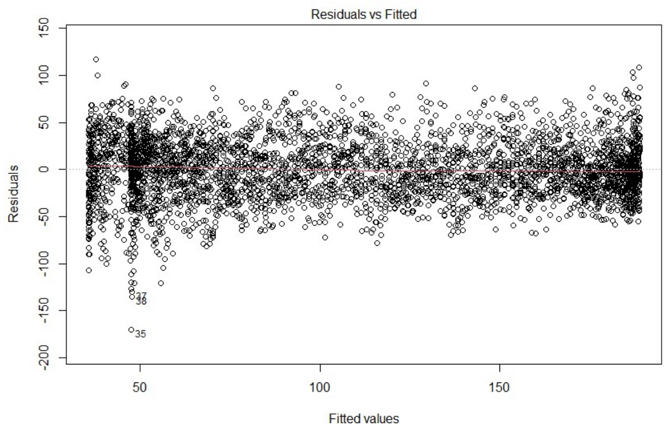

Figure 9: Residuals versus fitted plot for model (12)

2.1.5 Regression model for solar radiation based on temperature and time

It would be desirable to also use the temperature as an explanatory variable to better describe the behaviour of solar radiation. We hope that it will be an improvement on the model based solely on time. In this case, periods of both a year and a day are again used together with the added temperature variable. This temperature variable is added as a standalone value but also by multiplying it with all cosine and sine parameters. This is done since the temperature and time also influence each other, besides only influencing the solar radiation on their own. The general model for this will look as follows:

$$Q_i = \beta_0 + \psi T_i + \sum_{j=1}^{n}(\beta_j cd_{i,j} + \gamma_j sd_{i,j} + \delta_j cy_{i,j} + \zeta_j sy_{i,j} + \kappa_j T_i cd_{i,j} + \lambda_j T_i sd_{i,j} + \mu_j T_i cy_{i,j} + \nu_j T_i sy_{i,j}) + \varepsilon_i. \quad (13)$$

The temperature data previously used from KNMI will be used for this model as well. When deciding on the amount of variables in this model, the statistical significance and adjusted R-squared are again taken into account. After adding the variables corresponding to j equal to four, it can be noticed that this only increases the adjusted R-squared by 0.0004. This again does not influence the adjusted R-squared rounded to two decimals, so in order not to complicate the model by using eight extra variables, the level j will run from one to three. The statistical output can be found in Table 5.

Model variable	Estimation of parameter	95% confidence interval	Statistical significance
(constant)	0.157	[-0.59;0.90]	0.678
T	0.271	[0.27;0.28]	$< 2 * 10^{-16}$
cd_1	-12.583	[-13.23;-11.94]	$< 2 * 10^{-16}$
sd_1	-1.698	[2.29;-1.10]	$2.05 * 10^{-8}$
cd_2	9.078	[8.47;9.68]	$< 2 * 10^{-16}$
sd_2	-3.186	[-3.78;-2.59]	$< 2 * 10^{-16}$
cd_3	-3.890	[-4.48;-3.30]	$< 2 * 10^{-16}$
sd_3	4.254	[3.67;4.84]	$< 2 * 10^{-16}$
cy_1	21.895	[20.86;22.93]	$< 2 * 10^{-16}$
sy_1	2.384	[1.44;3.33]	$7.40 * 10^{-7}$
cy_2	-14.738	[-15.64;-13.84]	$< 2 * 10^{-16}$
sy_2	-5.281	[-6.18;-4.38]	$< 2 * 10^{-16}$
cy_3	2.035	[1.31;2.76]	$3.63 * 10^{-8}$
sy_3	3.344	[2.66;4.03]	$< 2 * 10^{-16}$
$T * cd_1$	-0.320	[-0.33;-0.31]	$< 2 * 10^{-16}$
$T * sd_1$	0.257	[0.25;0.26]	$< 2 * 10^{-16}$
$T * cd_2$	0.021	[0.02;0.03]	$< 2 * 10^{-16}$
$T * sd_2$	-0.103	[-0.11;-0.10]	$< 2 * 10^{-16}$
$T * cd_3$	0.026	[0.02;0.03]	$< 2 * 10^{-16}$
$T * sd_3$	-0.049	[-0.05;-0.04]	$< 2 * 10^{-16}$
$T * cy_1$	-0.367	[-0.38;-0.36]	$< 2 * 10^{-16}$
$T * sy_1$	0.085	[0.08;0.09]	$< 2 * 10^{-16}$
$T * cy_2$	0.018	[0.01;0.02]	$2.21 * 10^{-6}$
$T * sy_2$	-0.017	[-0.02;-0.01]	$1.22 * 10^{-6}$
$T * cy_3$	0.008	[0.002;0.01]	$6.14 * 10^{-3}$
$T * sy_3$	-0.028	[-0.03;-0.02]	$< 2 * 10^{-16}$

Table 5: Statistical output for the model for solar radiation based on time and temperature

Note that the estimate for the constant β_0 is not statistically significant. This is not strange, since it is expected to be around zero. This results in the model

$$\begin{aligned}Q_i = {}& 0.27T_i - 12.58cd_{i,1} - 1.70sd_{i,1} + 9.08cd_{i,2} - 3.19sd_{i,2} - 3.89cd_{i,3} + 4.25sd_{i,3} + 21.89cy_{i,1} + 2.38sy_{i,1} \\ & - 14.74cy_{i,2} - 5.28sy_{i,2} + 2.03cy_{i,3} + 3.34sy_{i,3} - 0.32T_icd_{i,1} + 0.26T_isd_{i,1} + 0.02T_icd_{i,2} - 0.10T_isd_{i,2} \\ & + 0.03T_icd_{i,3} - 0.05T_isd_{i,3} - 0.37T_icy_{i,1} + 0.08T_isy_{i,1} + 0.02T_icy_{i,2} - 0.02T_isy_{i,2} + 0.01T_icy_{i,3} - 0.03T_isy_{i,3} + \varepsilon_i\end{aligned} \tag{14}$$

This model returns an adjusted R-squared of 0.774. Since this formula looks quite extensive, it is checked whether all variables contribute to the fit of this model by noticing the changes in the adjusted R-squared when removing them. The most obvious idea is that all variables multiplied by the temperature variable may seem to have less of an influence given the smaller values of parameters. However, the adjusted R-squared decreases to 0.657 in the case that they are all removed.

Below, a visualization of this model compared to the actual data is displayed in Figure 10. Note that most of the error seems to lie in the peaks. In Figure 11, we display the residuals versus fitted plot. There seems to be a linearly decreasing border, which can be explained by the fact that solar radiation should always contain non-negative values only. There is less variability in higher fitted values, which is a bit strange but acceptable. There is no clear increasing pattern anymore as there was in model (10), where temperature was not included in the model.

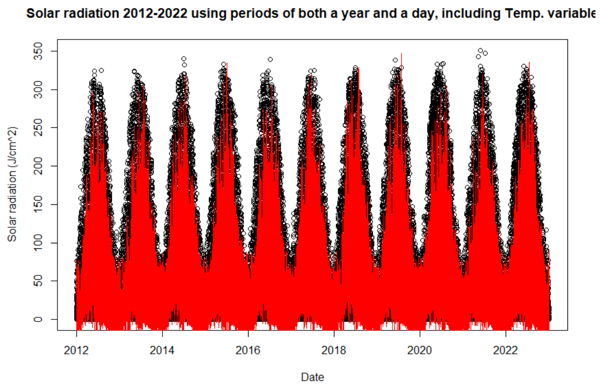

Figure 10: Visualization of model 14 versus data

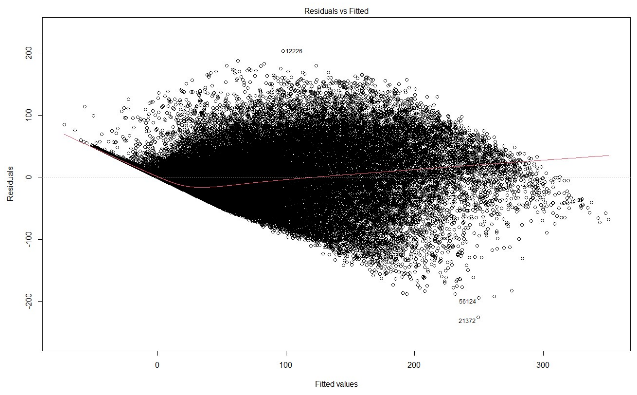

Figure 11: Residuals versus fitted graph for model 14

This model seems to describe the behaviour of solar radiation relatively well. However, to predict the solar radiation in the future, this model requires knowledge on the temperature data. Since it is likely to not change that much, one option to look into will be simply using the temperature data from 2022 for the year 2023. Next to that, we also want to try and use our previous regression model on temperature to predict the temperature and use that in the model for solar radiation. When doing this, the formula follows the same structure as in equation (13), whilst now using another equation for the temperature variable, namely the equation from model (12), instead of data. Only half of the variables for j being equal to four are statistically significant. When removing these, the adjusted R-squared again drops by only 0.0004. Notice that now the constant β_0 is again statistically significant, which is a bit strange. Removing all four sine and cosine variables using j equal to three and periods of a year does not cause the adjusted R-squared to decrease, so we remove those. The overview of statistical output can be found in Table 6.

Model variable	Estimation of parameter	95% confidence interval	Statistical significance
(constant)	35.780	[28.59;42.97]	$< 2*10^{-16}$
T	0.064	[0.01;0.12]	0.02
cd_1	-5.390	[-6.17;-4.61]	$< 2*10^{-16}$
sd_1	-7.755	[-8.54;-6.97]	$< 2*10^{-16}$
cd_2	16.510	[15.73;17.29]	$< 2*10^{-16}$
sd_2	-3.691	[-4.48;-2.91]	$< 2*10^{-16}$
cd_3	-7.592	[-8.38;-6.81]	$< 2*10^{-16}$
sd_3	8.595	[7.81;9.38]	$< 2*10^{-16}$
cy_1	-15.087	[-22.33;-7.84]	$4.53*10^{-5}$
sy_1	-24.081	[-37.97;-10.19]	$6.80*10^{-4}$
cy_2	-21.274	[-31.27;-11.28]	$3.04*10^{-5}$
sy_2	11.558	[2.87;20.25]	$9.16*10^{-3}$
cy_3	6.210	[4.05;8.37]	$1.74*10^{-8}$
sy_3	1.465	[-1.03;3.97]	0.25
$T*cd_1$	-0.488	[-0.49;-0.48]	$< 2*10^{-16}$
$T*sd_1$	0.322	[0.32;0.33]	$< 2*10^{-16}$
$T*cd_2$	-0.008	[-0.01;-0.001]	0.02
$T*sd_2$	-0.124	[-0.13;-0.12]	$< 2*10^{-16}$
$T*cd_3$	0.057	[0.05;0.06]	$< 2*10^{-16}$
$T*sd_3$	-0.077	[-0.08;-0.07]	$< 2*10^{-16}$
$T*cy_1$	-0.167	[-0.22;-0.12]	$1.34*10^{-10}$
$T*sy_1$	0.202	[0.10;0.30]	$1.08*10-4$
$T*cy_2$	0.109	[0.03;0.19]	$6.87*10^{-3}$
$T*sy_2$	-0.096	[-0.16;-0.03]	$3.79*10^{-3}$
$T*cy_3$	0.018	[0.002;0.03]	$2.62*10^{-3}$
$T*sy_3$	-0.026	[-0.04;-0.01]	$2.78*10^{-6}$

Table 6: Statistical output for the model for solar radiation based on time and predicted temperature using model (12)

Using statistically significant variables that also contribute to the adjusted R-squared, this results in the equation

$$\begin{aligned} Q_i = {} & 29.09 + 0.10T_i - 5.39cd_{i,1} - 7.75sd_{i,1} + 15.64cd_{i,2} - 3.69sd_{i,2} - 7.59cd_{i,3} + 8.60sd_{i,3} - 21.31cy_{i,1} \\ & + 6.18sy_{i,1} + 2.01cy_{i,2} - 5.90sy_{i,2} - 0.49T_icd_{i,1} + 0.32T_isd_{i,1} - 0.12T_isd_{i,2} + 0.06T_icd_{i,3} - 0.08T_isd_{i,3} \\ & - 0.11T_icy_{i,1} - 0.05T_icy_{i,2} + 0.03T_isy_{i,2} + \varepsilon_i, \end{aligned} \tag{15}$$

where T_i equals (12). For this model, the adjusted R-squared equals 0.718. Figure 12 shows a visualization of this model and Figure 13 shows its corresponding residuals vs fitted plot.

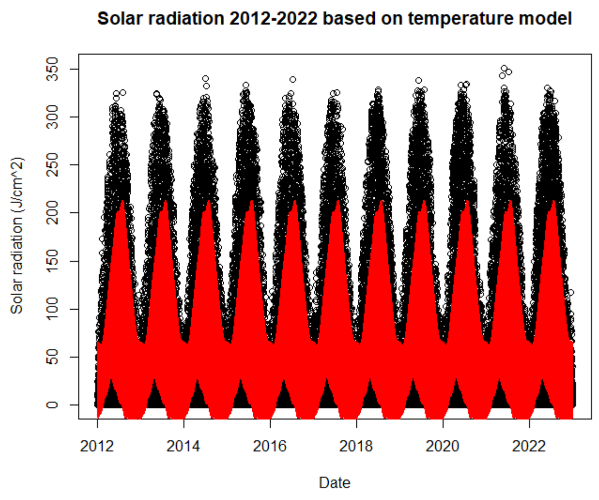

Figure 12: Visualization of model (15) versus data

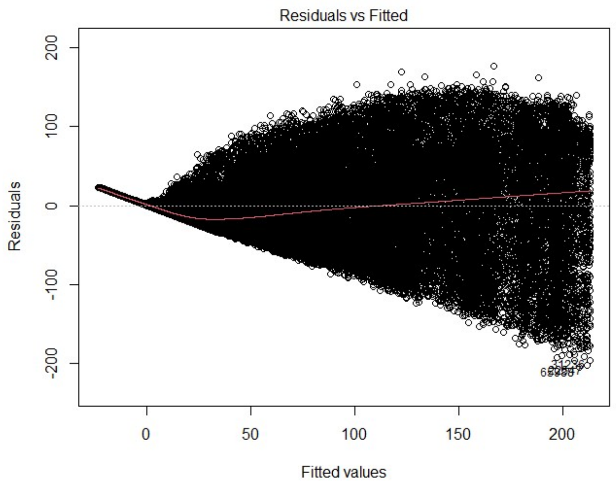

Figure 13: Residuals versus fitted graph for model (15)

It is immediately noticeable that the fit of the model has become worse mostly in the peaks. Also around zero, there seems to be more of a periodic movement now, which is not happening in the actual data. As can be seen, in the residuals versus fitted plot the variability becomes larger again for larger fitted values. This can be explained by the fact that this model again depends only on one variable, namely the dates, on which the temperature model is also based. This is less desirable, but in the case of predictions for the future the only deterministic variable available is time.

In the following sections, both combined models will be used to look into the solar radiation over 2023. We will use model (15) based on predicted temperature values via regression and model (14) using the temperature data from 2022 as explanatory variable. If possible, we will choose to use model (14) as it is more reliable.

2.2 Energy supply from solar panels

To obtain the amount of solar energy that is supplied, we can use the solar radiation to convert this into energy via solar panels. The average size of a solar panel used for households equals 16.500 cm^2 and households have an average amount of 10 solar panels per household ([20]). Next to that, we use a conversion rate of about 20% for solar radiation turned into energy ([17]). Note that this conversion rate decreases when temperatures get above 25 degrees Celsius ([9]). This does happen in the Netherlands, with an average of 24 to 40 days a year ([12]). However, this also includes days where the temperature was that high for only very few hours. In the regression model created for temperature, it even does not happen at all that temperatures rise above 25 degrees Celsius. Thus, we decide to disregard this.
To get the supply of solar energy per household, we use

$$Energy\_supply\_household = solar\_radiation * 2.778 * 10^{-7} * 0.2 * size\_solar\_panel * number\_of\_solar\_panels. \tag{16}$$

Here, $2.778 * 10^{-7}$ is the conversion from Joules to kWh and 0.2 equals the efficiency rate of 20% of the solar panel. Using the size of 16.500 cm^2 and amount of 10 solar panels, the total energy supply for one average household is obtained. For solar radiation, the model obtained by regression is used. We use the input of temperature data from KNMI, or in case of model (15) we use the regression equation for temperature. For the dates, we create a list with numerical values corresponding to the correct dates. To be able to look into the future at a later point, we will let our values for the dates run one year further to obtain the solar radiation over 2023. Using all this information, for all time steps in the list we created, we run the equation of the solar radiation model to obtain the value for that time point. This is then added to a list, creating a list with values for the amount of solar radiation per time step. This list is what is used as $solar\_radiation$ in equation 2.2. When visualizing the solar energy supply, it simply follows the regression model created but over different values, as it equals the regression model multiplied by several variables. Using the equation of model (14) for solar radiation in equation (16) with the temperature data over 2022, this returns the solar energy supply for a household over 2022 as displayed in Figure 14.

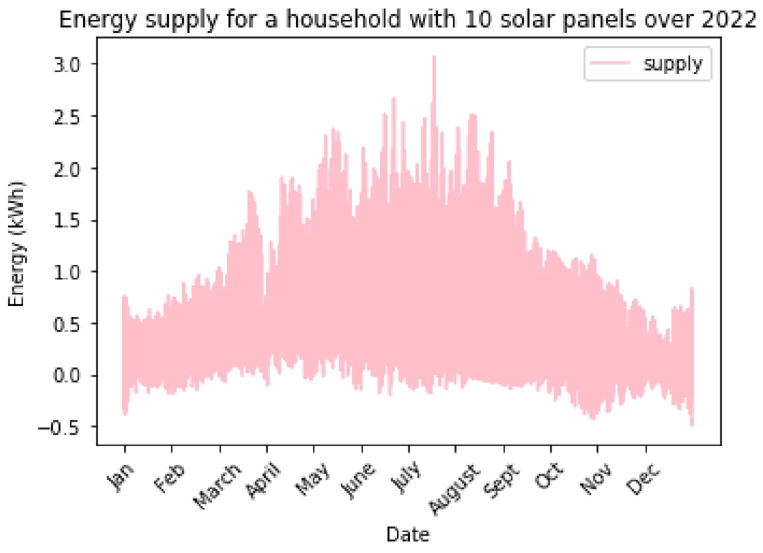

Figure 14: Energy supply for one household over 2022 using model (14)

3 Electricity demand

It is quite hard to obtain accurate data on electricity demand. In the Netherlands, there is no recent data available on actual household electricity usage over time. Our first simplistic scenario will therefore be based on an average usage, including peak and off-peak hours during a day. This is very simplistic, but may be more applicable for households where people work from home. The second scenario is a more realistic one, using earlier findings from 2007. Here, a seasonal difference is taken into account and we first let this total demand over time be equal to the current average demand of electricity for a household. For these first two scenarios, on top of the current electricity demand as it is set, we will also take heat pumps into account. In Section 1.2, we stated that gas will not be taken into account as modern houses use electricity also for heating. The current electricity demand does not take heating into account, which is the reason we will add this. In the last scenario, the demand is significantly higher; a future scenario is taken where both electric vehicles and heat pumps add to the electricity demand of households.

3.1 Scenario 1

In this scenario, the simplest form of demand will be used. We will use the average demand to create a constant demand during night and a constant demand during the day time. It is known that in the Netherlands, the average usage of electricity equals 2800 kWh per year ([14]). However, one average does not give a realistic overview of the electricity demand over time. For this, the day is divided into peak hours and off-peak hours, which are from 07:00 to 23:00 and 23:00 to 07:00 respectively ([18]). During these hours, a household uses approximately 0.05-0.15 kWh per hour during the night and 0.3-0.5 kWh per hour during the day ([7]). In this scenario, two values of 0.11 kWh and 0.42 kWh will be used for the off-peak and peak hours respectively, resulting in the yearly average of just under 2800 kWh. On top of that, we add a heat pump, which has an electricity usage of 3200 kWh per year ([22]). This results in a total electricity usage of just under 6000 kWh per year. Hourly, the heat pump increases the electricity usage 0.37 kWh, assuming a constant usage of electricity. Over a week, this demand scenario looks as in Figure 15.

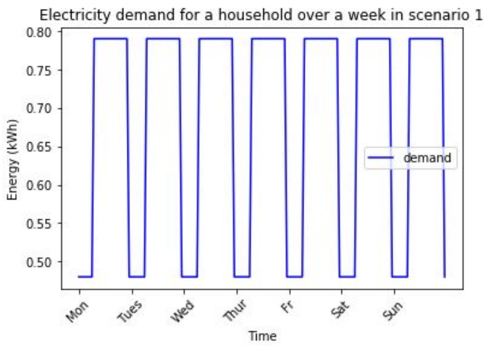

Figure 15: Demand scenario 1 visualized over a week

This scenario is very simplistic; however, recently it might become more applicable for people working from home. For them, the peak usage is spread more throughout the entire day instead of at certain hours only.

3.2 Scenario 2

In this scenario, a less simplistic demand profile will be formed, using older studies that gathered data throughout 2007. Since it is from 2007, we assume the average household did not work from home, which explains the peak usage shifting more towards the evening. In Figure 16, the output from the REMODECE project on residential monitoring of energy usage can be seen ([2]). This study takes into account the average European household. It may be that the average Dutch household differs a bit from that, especially when looking at air conditioning. However, it gives a good overview of someone not working from home. Next to this visualization, in Figure 17, an overview of the electricity usage of all households in the Netherlands over 2007 can be found. Here, we notice the difference between seasons.

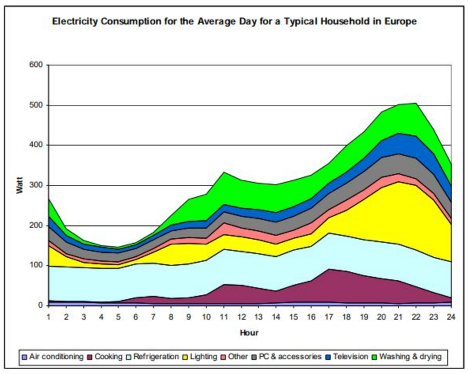

Figure 16: Electricity demand by REMODECE of an average European household based on data from 2007 ([2])

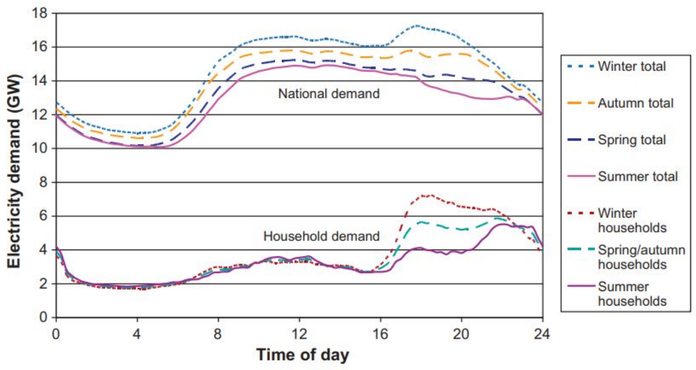

Figure 17: Total household electricity demand in the Netherlands in 2007 on average days of different seasons ([19])

Firstly, note that from midnight until around four in the afternoon, there are barely any differences between seasons. Then during the peak in the evening the differences become clear. To create the desired scenario, we follow the differences only in the peak as seen in Figure 17, while taking the amount throughout the day visualized in Figure 16 as the middle spring/fall case. This results in a total energy usage of 2823 kWh, so now just over 2800 kWh, for a household in this scenario. On top of this, again the heat pump usage is added of 0.37 kWh per hour. In total, the household then has an electricity usage of 6064 kWh over a year, just above the first scenario. The demand over one day in different seasons in this scenario is then given in Figure 18.

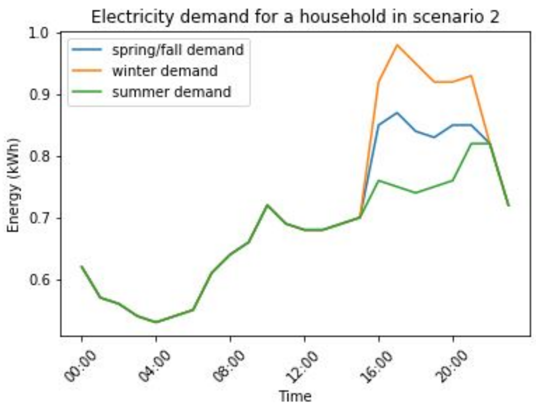

Figure 18: Demand scenario 2 visualized over a day for different seasons

In using these seasonal differences throughout the year, we divide them as each season being three months. We state that winter includes December, January and February; spring March, April and May; summer June, July and August; and fall September, October and November. Following this timeline, the three seasonal differences will be used.

3.3 Scenario 3

In the future, next to the heat pumps previously mentioned, electric vehicles are likely to become increasingly prevalent in households. This will cause households to have an increased electricity usage. There is a specific case to look at focusing on the year 2040, where electric vehicles make up 75% of all vehicles and electric heat pumps are installed in 300.000 existing households while 80% of the houses being built include a heat pump ([21]). This scenario is denoted as scenario B in Figure 19.

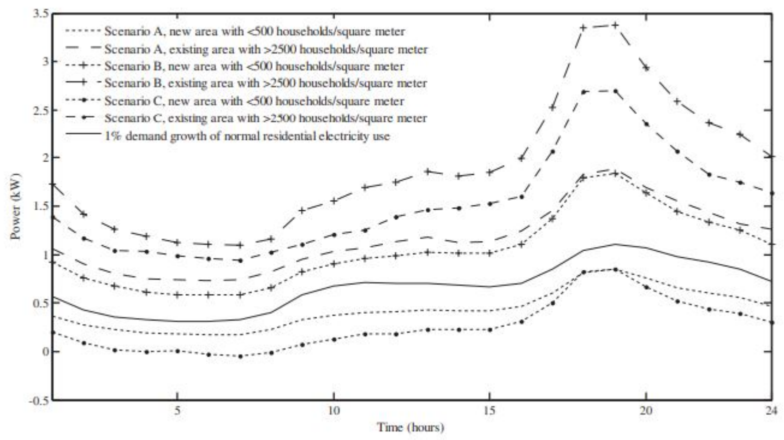

Figure 19: Electricity demand profiles in 2040 for three different scenarios [21]

In this study, the starting point of electricity demand is set on 3400 kWh per year, which is already higher than the current average electricity usage of 2800 kWh. This can be explained by the fact that this study was done in 2012, while the average electricity usage of a household has decreased in the last ten years ([6]). Because of this, our focus will be on the lower option of scenario B, with less than 500 households per squared kilometer. This is a possible scenario for 2040, but we will investigate what happens for households currently following such a demand profile. The fluctuation throughout the year will still be taken into account, using the same three seasonal variations as in the previous scenario. The scenario B as depicted in Figure 19 will be taken as the middle spring/fall case, with the seasonal fluctuations again being taken from Figure 17. In Figure 20, this scenario is visualized.

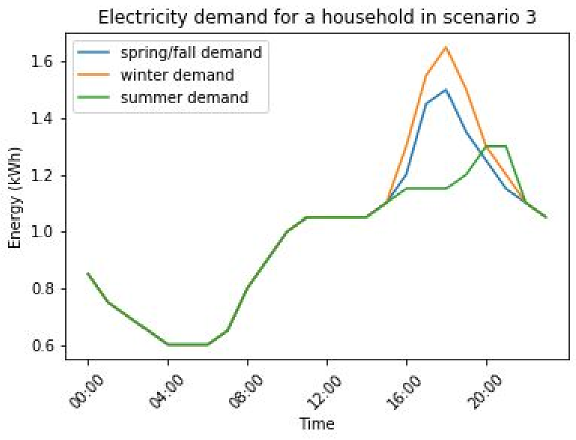

Figure 20: Demand scenario 3 visualized over a day for different seasons

This scenario results in a total electricity usage of 8550 kWh per year. This is indeed an extreme case, since currently a large household consisting of five people or more has an average electricity demand of 4350 kWh per year ([14]). However, after including the heat pumps in both our previous scenarios, the difference with this scenario is not that big and mostly comes from the addition of electric vehicles.

We have now described three different demand profiles a household can follow. The first one being a simplistic one, possibly representing a household with people working from home. The second one is a more traditional household following the times most households currently use their electricity. The third one continues the traditional peak usage in the evening, but also adds on an electric car, giving us an insight in how more households may use their energy in the future. Overall, a neighborhood is likely to consist of households using similar amounts of energy, as houses are of similar size. For the first two scenarios this holds. The third

scenario has a higher energy usage; however, it is not strange for only a few households in a neighborhood to have electric vehicles. Thus, we will be combining these three demand scenarios into one neighborhood.
When buying solar panels for a household, it is advised to include as many solar panels on a house to cover the demand of a year, which is why the average household currently was advised to have 10 solar panels ([8]). However, our demand scenarios use a different amount of electricity over the year. For this reason, households following the first two demand scenarios will have 16 solar panels on their house, while for the third scenario 23 solar panels are needed.

4 Simulation

After investigating the supply and demand separately, it is time to look into the difference between supply and demand, i.e. the dependency. We will use our findings in both previous sections and look at the dependency on the grid for the three demand scenarios stated in Section 3. After investigating the current situation, two solutions are implemented to find a way to decrease the dependency. For this, we develop code in Python; before describing the results, we explain the most important parts of the code in Subsection 4.1.

4.1 Code for simulation

There are some important things the code has to be able to do. Firstly, the main point of interest is the dependency, so we need a way to compute that. Then, two solutions are implemented: one to change the supply and one to change the demand. Lastly, a neighborhood will be created and used in a simulation.

4.1.1 Obtaining the dependency

The dependency consists of two parts: energy shortage and energy surplus. Thus the code is pretty simple; we calculate the shortage when the demand is higher than supply and the surplus when supply is higher than demand. Adding those two results in the total dependency.

Algorithm 1 Obtaining the difference in supply and demand, i.e. the grid dependency

```
for $i$ in range(time) do
    if demand higher than supply then
        $shortage = demand[i] - supply[i]$
        $total\_shortage \leftarrow total\_shortage + shortage$
    else if demand lower than supply then
        $surplus = supply[i] - demand[i]$
        $total\_surplus \leftarrow total\_surplus + surplus$
$total\_dependency \leftarrow total\_shortage + total\_surplus$
```

In the end, we will also use this in the simulation when multiple runs are done; then the average over these runs is taken for the shortage, surplus and dependency.

4.1.2 Creating a storage solution

The first solution is one to change the supply part. The idea is to add surplus energy to a storage system, which can then be put back into the house when there is a shortage. This is done with and without a maximum capacity; in Algorithm 2 pseudo-code for the one with a maximum capacity is shown. This means that the storage solution can only be filled until maximum capacity is met, and the rest simply stays as a surplus at that point in time.

Algorithm 2 Storage solution with maximum capacity

```
$battery\_storage = 0$
$maximum\_capacity = 130$
for $i$ in range(time) do
    if supply is higher than demand then
        $surplus = supply[i] - demand[i]$
        $battery\_storage \leftarrow battery\_storage + surplus$
        if battery storage exceeds maximum capacity then
            $supply[i] \leftarrow supply[i]$ - amount stored
            $battery\_storage \leftarrow max\_storage$
        else
            $supply[i] \leftarrow supply[i] - surplus$
    else if supply is lower than demand then
        if $battery\_storage <= demand[i] - supply[i]$ then
            $supply[i] \leftarrow supply[i] + battery\_storage$
            $battery\_storage = 0$
        else
            $battery\_storage \leftarrow battery\_storage - (demand[i] - supply[i])$
            $supply[i] = demand[i]$
```

In case of unlimited capacity, an empty list is created where the storage at every time point is added. This way, the maximum storage at a time can be checked, and also whether that happens more often.

4.1.3 Changing demand behaviour

The second solution focuses on changing the demand side. The idea of this is that when there is more demand than supply, (part of) the surplus demand gets redirected to the next time step. The other way around, if there is more supply than demand, (part of) the demand from the next time step gets shifted to the current time step. This is done by dynamic pricing, which causes people to shift their demand in case of higher or lower prices than desired. This means they will stall when prices are high and use more when prices are low. In Section 4.3.2, a realistic demand response will be set. The process of shifting this demand can be seen in Algorithm 3.

Algorithm 3 Changing demand behaviour

```
for $i$ in range(time) do
    if demand is higher than supply then
        $demand[i+1] \leftarrow demand[i+1] + response\_rate * (demand[i] - supply[i])$
        $demand[i] \leftarrow demand[i] - response\_rate * (demand[i] - supply[i])$
    else if demand is lower than supply then
        if $response\_rate * demand[i+1] <= supply[i] - demand[i]$ then
            $demand[i] \leftarrow demand[i] + response\_rate * demand[i+1]$
            $demand[i+1] \leftarrow demand[i+1] - response\_rate * demand[i+1]$
        else
            $demand[i+1] \leftarrow demand[i+1] - (supply[i] - demand[i])$
            $demand[i] = supply[i]$
```

The response rate is a number between 0 and 1, indicating the fraction of demand that gets shifted. Note that for this demand shift, we currently only focus on the immediate demand response for the current time, influencing only the demand at that moment and one hour in advance. However, realistically it is likely to have an effect on more than just those times. To incorporate a long-term influence of dynamic pricing, we will also look into shifting the entire peak of the energy usage of a household.

4.1.4 Simulating for a neighborhood

Finally, in Section 4.5, a neighborhood will be simulated. The making of the neighborhood is done simply by choosing 30 houses and picking a random number of times scenario 1, 2 or 3 happens. It is then counted how many times each scenario is picked, which results in the number of houses for those scenarios respectively. This can be seen as creating a neighborhood where a certain number of people work from home and a certain number of people have an electric car. It is expected that in a neighborhood the households will all follow a different demand profile depending on their lives. A function is made to generate the total neighborhood demand and supply. Below a short overview of how this function is structured is given in Algorithm 4. Here, the solutions are also incorporated; when simulating the current scenario of a neighborhood, those solutions are simply removed.

Algorithm 4 Generating the total neighborhood demand and supply

```
for $i$ in range(amount of houses) do
    pick random number of houses for each scenario
for all demand scenarios do
    set demand
    set amount of solar panels and corresponding supply
    sum demand and supply over all houses with that scenario
sum total demand and supply over all houses
implement demand change as done in Algorithm 3
implement storage solution as done in Algorithm 2
return total demand, total supply
```

This function is then set to run a certain number of times, from which each time a corresponding demand and supply for the neighborhood is generated.

4.2 Results of current situation of a household

First, a simulation of the current situation without any solutions added is needed. This way, our starting point of current dependency can be determined and we will know what needs to be decreased. This starting point consists of a certain amount of kWh that is the dependency on the electricity grid, consisting of a shortage and a surplus. We will look into the entire year of 2023, but also zoom into two specific weeks. These weeks are the first week of February and the last week of July; a week with very low solar radiation and one with very high solar radiation. They will later be denoted as a winter week and summer week respectively.
We will first look into both model (15) using the regression model to predict temperature values over 2023, and model (14) using temperature data from 2022 as if they were the same for 2023. We will do this to determine whether they give different results in terms of dependency.

4.2.1 Scenario 1

When looking at 2023, both models as described in Section 2.1.5 are implemented as supply of a household with the first demand scenario.

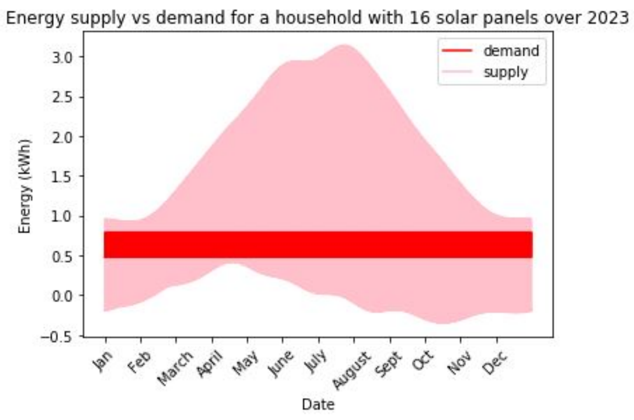

Figure 21: Supply versus demand over 2023 for scenario 1 using model (15)

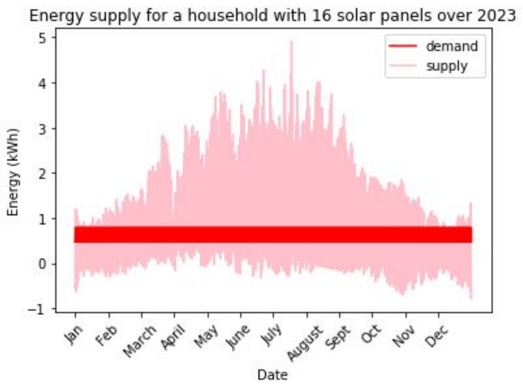

Figure 22: Supply versus demand over 2023 for scenario 1 using model (14) with 2022 temperature data

For model (15), the total dependency amounts to 5903 kWh over the whole year. For the model on the right in Figure 22 using model (14), the total dependency equals 6240 kWh. We notice that both give quite similar results, even though visually the peak in Figure 22 seems to be higher. In Table 7 below, the total dependency is set out into the energy shortage and surplus.
Now, we take the two specific weeks into account; the winter week and summer week. They are both visualized using both models for solar radiation.

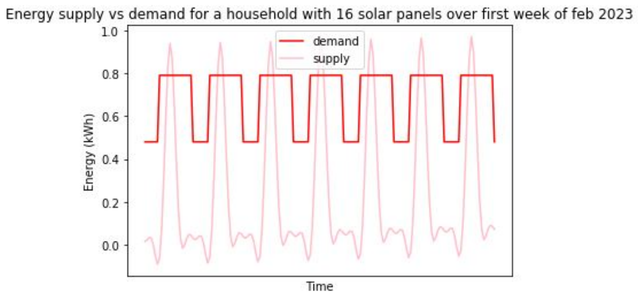

Figure 23: Supply versus demand winter week for scenario 1 using model (15)

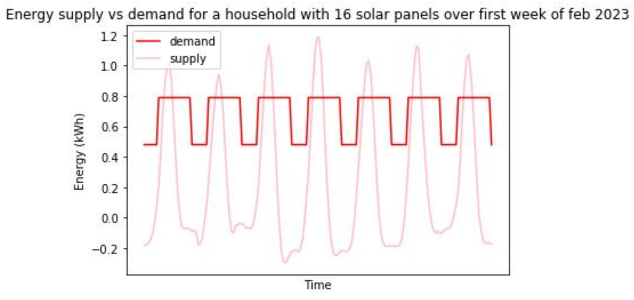

Figure 24: Supply versus demand winter week for scenario 1 using model (14) with 2022 temperature data

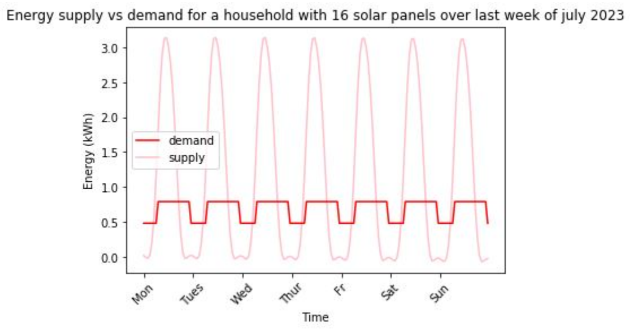

Figure 25: Supply versus demand summer week for scenario 1 using model (15)

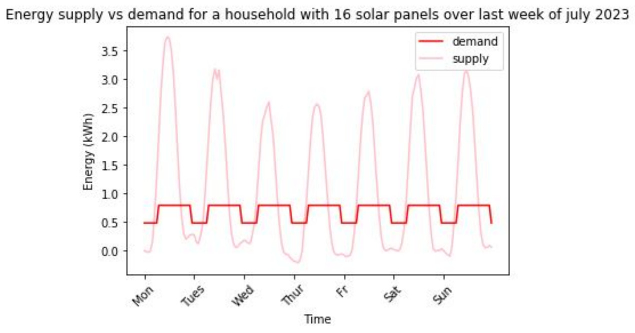

Figure 26: Supply versus demand summer week for scenario 1 using model (14) with 2022 temperature data

In Table 7, the dependency of these weeks can also be found.

Time	Supply model used	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year	model (15)	3141.4	2761.9	5903.4
Entire year	model (14)	3167.8	3072.7	6240.5
Winter week	model (15)	80.0	2.2	82.2
Winter week	model (14)	83.8	5.9	89.8
Summer week	model (15)	45.7	123.7	169.5
Summer week	model (14)	44.5	113.6	158.2

Table 7: Dependency amount in scenario 1 using model (14) and (15)

It can be noticed that, as expected, in the summer week the household has a larger surplus while in the winter week the dependency consists mostly of the shortage. However, it can also be seen that now the total problem of dependency seems to be larger in the summer than during winter, because of this large surplus. Over the entire year, the dependency seems to be split approximately equal over the shortage and surplus.

4.2.2 Scenario 2

Using the second demand scenario, as described in Section 3.2, the demand will have different peaks throughout the year. Looking over the entire year, a visualization of this can be found in Figures 27 and 28.

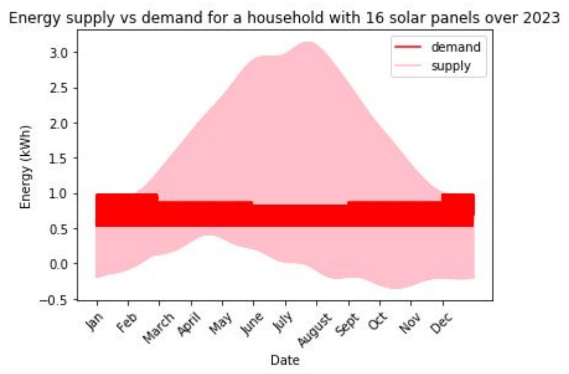

Figure 27: Supply versus demand over 2023 for scenario 2 using model (15)

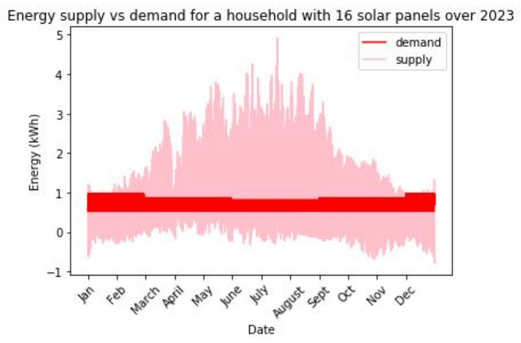

Figure 28: Supply versus demand over 2023 for scenario 2 using model (14) with 2022 temperature data

Before looking into the separate weeks to notice the daily behaviour of the demand, it can be seen that now the peak of demand is lower during summer, while the supply is at its highest. This may cause for a greater imbalance than in our first scenario. In Figures 29-32, both the summer and winter week are visualized.

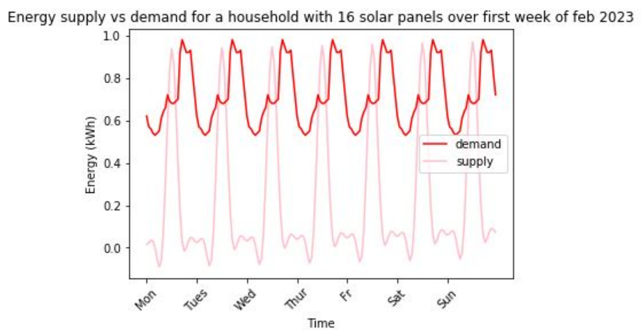

Figure 29: Supply versus demand winter week for scenario 2 using model (15)

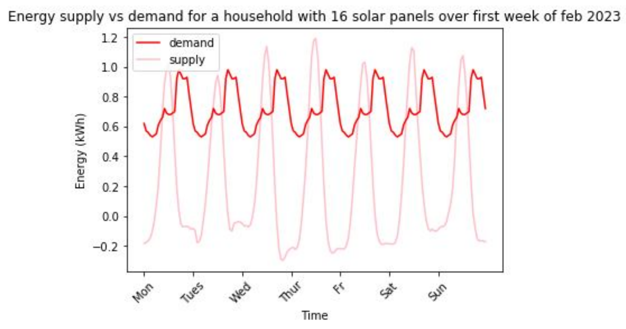

Figure 30: Supply versus demand winter week for scenario 2 using model (14) with 2022 temperature data

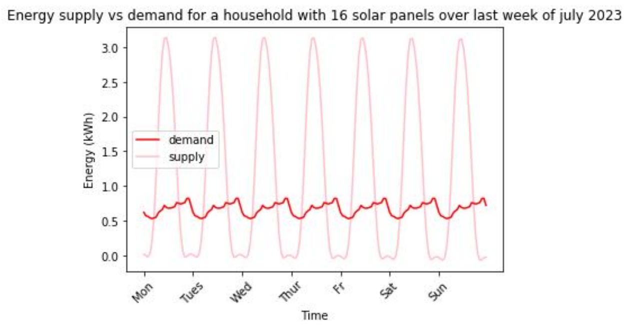

Figure 31: Supply versus demand summer week for scenario 2 using model (15)

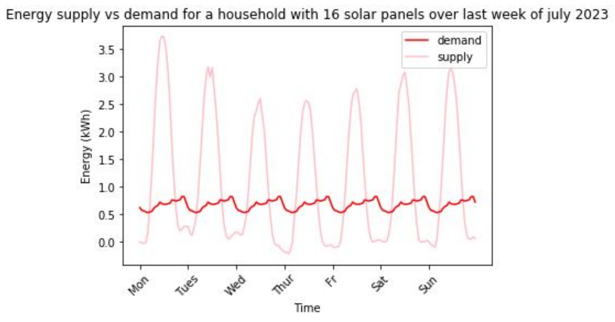

Figure 32: Supply versus demand summer week for scenario 2 using model (14) with 2022 temperature data

Notice that the demand of electricity has a higher peak in the winter week than in the summer week. However, since this scenario focuses on a more traditional household where energy usage is at its peak in the evening, the peak of demand and supply lie just besides each other. Especially during the winter week, it looks like the amount that is needed during the peak could be accommodated if only the timing was different.
In Table 8, the dependency of the entire year and separate weeks can be found for both models.

Time	Supply model used	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year	model (15)	3458.5	3030.7	6489.2
Entire year	model (14)	3488.0	3344.6	6832.6
Winter week	model (15)	86.8	4.5	91.3
Winter week	model (14)	91.9	9.3	101.2
Summer week	model (15)	49.7	130.3	180.0
Summer week	model (14)	48.8	120.6	169.4

Table 8: Dependency amount in scenario 2 using model (14) and (15)

4.2.3 Scenario 3

For this scenario, a future possibility of a house using a larger amount of electricity was taken. However, this house then also needs more solar panels to account for that. The expectation is then also that the dependency in absolute numbers is a lot higher, but that it follows the same pattern as the previous two scenarios. In Figures 33 and 34 there is again a visualization of the entire year 2023 for the two different models using this demand scenario.

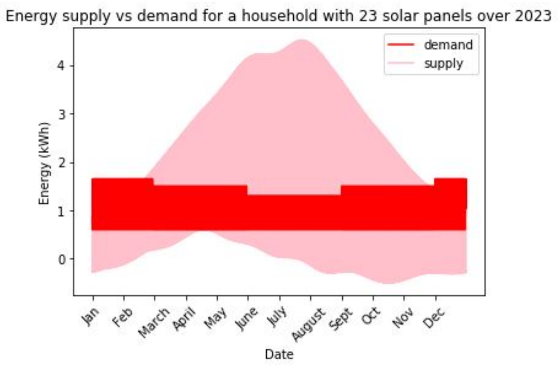

Figure 33: Supply versus demand over 2023 for scenario 3 using model (15)

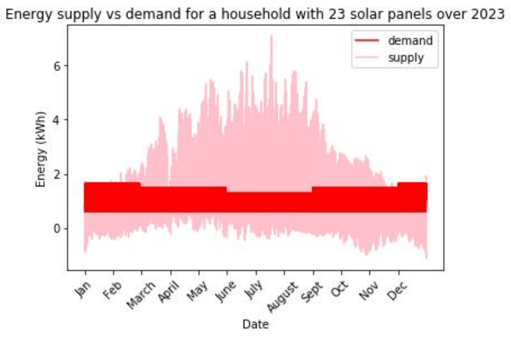

Figure 34: Supply versus demand over 2023 for scenario 3 using model (14) with 2022 temperature data

Notice that indeed the demand is significantly higher than in the previous scenarios, but so is the supply. In figures 35-38 we show a visualization of the focus on the winter and summer week.

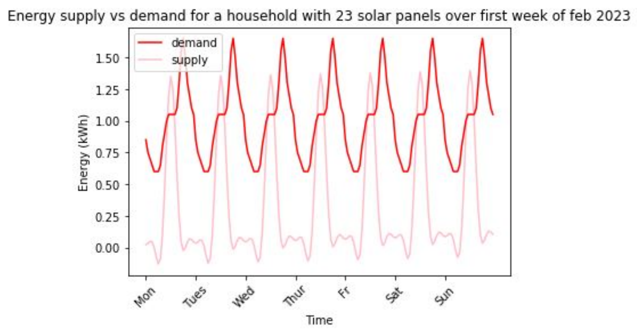

Figure 35: Supply versus demand winter week for scenario 3 using model (15)

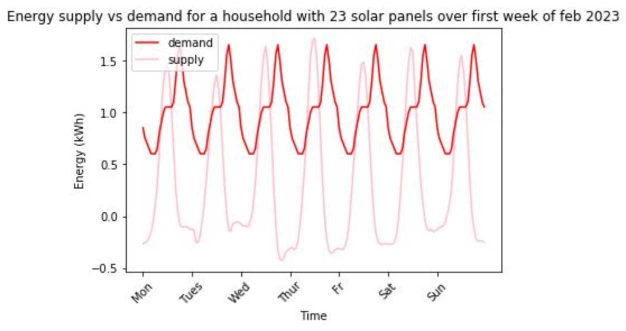

Figure 36: Supply versus demand winter week for scenario 3 using model (14) with 2022 temperature data

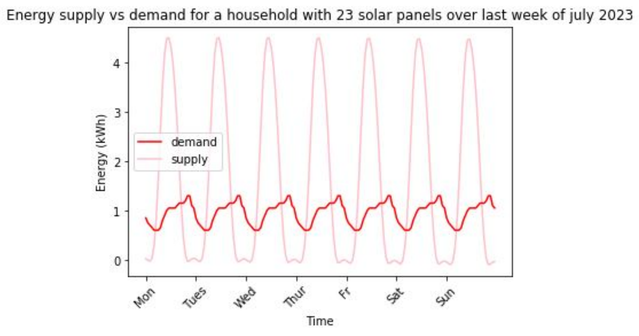

Figure 37: Supply versus demand summer week for scenario 3 using model (15)

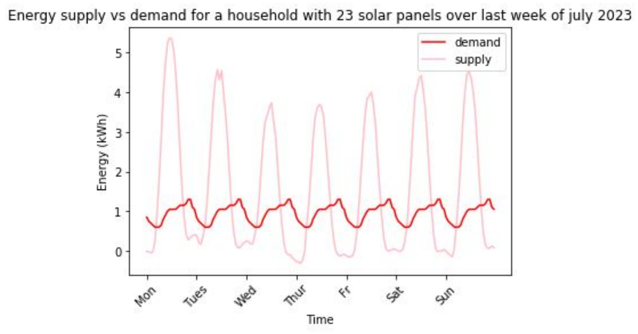

Figure 38: Supply versus demand summer week for scenario 3 using model (14) with 2022 temperature data

As expected, it looks quite similar to the previous scenario but with larger numbers for both supply and demand. In Table 9 the dependency of the entire year and separate weeks can be found for both models.

Time	Supply model used	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year	model (15)	4849.9	4397.8	9247.7
Entire year	model (14)	4891.1	4847.7	9738.8
Winter week	model (15)	119.3	5.0	124.3
Winter week	model (14)	126.9	12.3	139.2
Summer week	model (15)	71.7	190.0	261.7
Summer week	model (14)	69.7	175.3	245.1

Table 9: Dependency amount in scenario 3 using model (14) and (15)

Notice that indeed the total dependency is higher compared to the other two scenarios. However, with the larger amount of energy used taken into account, it is relatively similar to the other two scenarios. As in the other two scenarios, the shortage and surplus of energy are quite similar, but the shortage is always just a bit larger.

In this section, we have noticed that the difference between the two models used is usually not large. In all scenarios, taking the sum over the entire year, the usage of model (14) returns outcomes deviating at most 5.4% from the outcomes using model (15). While creating the models, it was stated that model (14) was more reliable, as it was based on temperature data rather than a regression model for temperature. Considering this, model (14), including temperature data from 2022, will be used in the upcoming sections.

4.3 Results of possible solutions

Two possible solutions to better match supply and demand will be implemented; one for changing the supply and one for changing the demand. We will first implement them on household level, to see the influence of this solution on one household. Again, all three demand scenarios will be taken into account.

4.3.1 Storage solution

Firstly, a storage solution is implemented to change the supply. This storage solution stores the surplus energy and returns it when there is a shortage of energy. Assumptions are made to simplify this solution. We state that energy can be charged into the storage compartment and discharged back into the house without losing energy or taking a significant amount of time for this. The energy can also be stored indefinitely while the 100% efficiency remains. Besides this, Assumption 1.2 stating that cable and conversion losses will be ignored also still holds.
One storage solution will have no maximum capacity; from this it will be known what is the optimum that can be achieved and what capacity would be needed for this. Then, a second storage solution will be implemented with a maximum capacity. For this case, a mechanical engineer in the group working on this project has advised a flywheel storage solution. This solution has a maximum capacity of around 130 kWh ([3]), which is the maximum capacity that we will use to check a more realistic case. The results of the unlimited storage solution will only be used as a guideline, whereas the second one is a realistic case to investigate more.

Scenario 1 When looking into the first demand scenario, an unlimited storage solution results in a maximum capacity needed of 1651 kWh at a time. This is not one outlier, but such a large amount of storage at a time happens more often. This does decrease the total dependency a lot; from the original 6240.5 kWh to 680.1 kWh. This equals 11% of the original dependency, so we state that a decrease of 89% would be possible when the storage solution has a capacity of 1651 kWh. Naturally, this is not a realistic amount to store for one household. When implementing the flywheel storage system, one household would have one flywheel resulting in a maximum capacity of 130 kWh. For the first demand case, we show the visualization of using a storage solution in Figures 39-41.

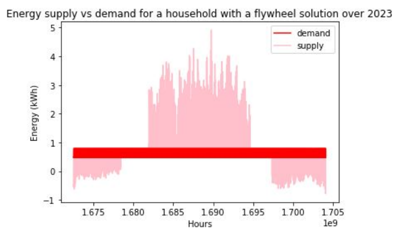

Figure 39: Result using a storage solution over 2023 in scenario 1

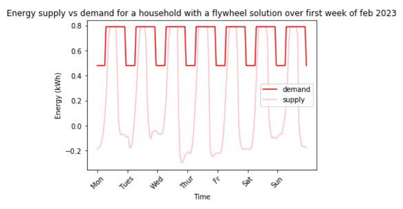

Figure 40: Result using a storage solution over winter week in scenario 1

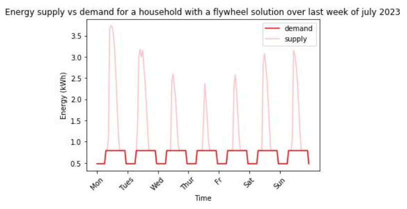

Figure 41: Result using a storage solution over summer week in scenario 1

It can be seen that during winter, the demand will simply be met by the supply while the rest is stored, while during summer there is still a surplus since the storage system has a maximum capacity. However, in that case there is no shortage since the system does store enough to cover that. The dependency, again divided into shortage and surplus, for using this solution can be found in Table 10 below.

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	3167.8	3072.9	6240.5
Entire year	1616.3	1521.2	3137.5
Winter week	77.9	0	77.9
Summer week	0	68.9	68.9

Table 10: Dependency amount in scenario 1 using a flywheel storage solution

Considering the entire year of 2023, the realistic storage solution in this scenario provides a dependency decrease of 50%.

Scenario 2 In this scenario, the maximum amount of energy that would need to be stored at a time with unlimited capacity equals 1675 kWh. This would result in a total dependency over 2023 of 720.6 kWh; again a similar decrease of 89% compared to the original dependency of 6832.6 kWh. When taking the flywheel storage solution with a maximum capacity of 130 kWh, the demand and supply graphs can be seen in Figures 42-44.

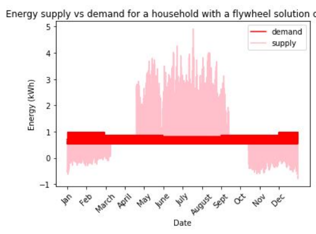

Figure 42: Result using a storage solution over 2023 in scenario 2

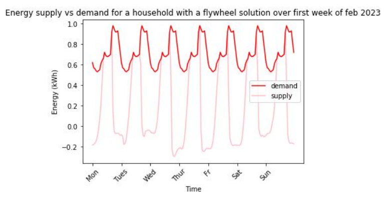

Figure 43: Result using a storage solution over winter week in scenario 2

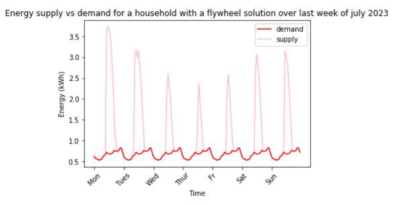

Figure 44: Result using a storage solution over summer week in scenario 2

We notice the same behaviour as in the first scenario; at the beginning and end of the year there is still a shortage but no surplus while in the middle of the year there is a surplus but no shortage. This is naturally because of the maximum storage capacity of the flywheel; the bigger this capacity becomes, the better it will be able to decrease this difference. In Table 11, an overview of the dependency values can be found.

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	3488.0	3344.6	6832.6
Entire year	1688.0	1544.7	3232.7
Winter week	82.6	0	82.6
Summer week	0	71.6	71.6

Table 11: Dependency amount in scenario 2 using a flywheel storage solution

Notice that using this storage solution, dependencies are very similar for the first and second demand scenario. This can be explained by the fact that the peak has shifted but overall the average usage of a day, and thus a week, is equal in both scenarios. Taking the entire year of 2023, this flywheel storage solution results in a dependency decrease of 53%.

Scenario 3 This scenario takes households into account with a high energy usage. This results in the need for an even larger storage system. The maximum storage at a time for a storage solution with infinite capacity would be 2479 kWh. This results in a dependency of 996.2 kWh, which is a decrease of 90%. This is again similar to what could best be achieved in the other two scenarios, but now a larger storage system is required. When using the flywheel solution with the maximum capacity of 130 kWh, we thus expect this to have less impact in this scenario than in the previous two. In Figures 45-47 we show a visualization over the year 2023 and the two specific weeks.

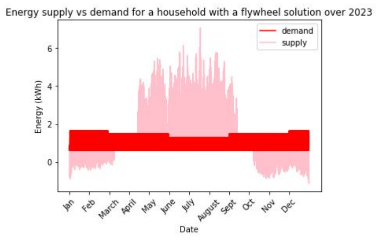

Figure 45: Result using a storage solution over 2023 in scenario 3

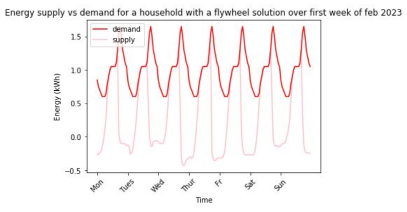

Figure 46: Result using a storage solution over winter week in scenario 3

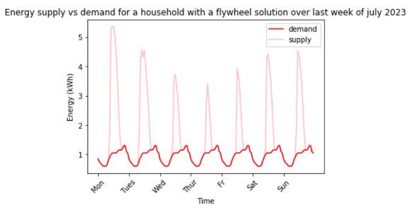

Figure 47: Result using a storage solution over summer week in scenario 3

This again looks similar to the previous scenario, but with larger values. In Table 12, we show the values for the dependency.

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	4891.1	4847.7	9738.8
Entire year	2392.8	2349.4	4742.2
Winter week	114.5	0	114.5
Summer week	0	105.3	105.3

Table 12: Dependency amount in scenario 3 using a flywheel storage solution

Using this flywheel storage solution returns a decrease in the dependency of 51%. This is similar to the previous two scenarios, which is better than we expected. We expected to need a larger storage solution, but it turns out that the same maximum capacity results in approximately the same decrease in dependency, even though the total energy usage is higher.

Overall, the storage solution seems to have a relatively large impact on the dependency in all demand scenarios, all around 50%. When the capacity of the storage solution increases, the possibilities of how much the dependency could be decreased are very large, approximately 90%.

4.3.2 Dynamic pricing

In this section, a solution for changing the demand is implemented. This is done by dynamic pricing, to influence the behaviour of people. It was found that dynamic pricing in combination with enabling technologies, such as two-way programmable communicating thermostats, was most efficient. This amounted in a demand response of 27% to 44% ([5]). Given that this study is from 2010 and the current prices of electricity are a hot topic, the assumption is made that the response rate in 2023 lies at the top of this range. Therefore, we implement a change in demand such that 44% of electricity used when supply is low will be stalled to a later time. When supply is high and demand low, also 44% will be taken from demand one time step into the future and added onto the current time point as long as this doesn't exceed the current supply. In real life this can be seen as wanting to do your laundry in an hour, but seeing that the price is low at this moment so deciding to do it now instead. For this, it is assumed that we know the supply one time step into the future, such that we can determine the amount to switch to the current time.
Again, we will compare this realistic scenario to a case where the demand response is 100%. This way we check what is the maximum possible result we can obtain if people were to switch their usage entirely as desired.

This solution focuses solely on the short-term change of demand. In the long term, people are likely to switch their entire behaviour next to the short term reactions of current prices. For this, we state that the entire peak usage in scenario 2 and 3 get switched from 16:00-23:00 to 08:00-15:00. In a household we choose that they either switch this demand pattern or that they do not switch. In the creation of the neighborhood, we will set the case that half of the houses switch their demand pattern and half do not, corresponding to the demand response of 44%. As we are currently focusing on a singular household, we will implement the demand change that does switch their entire demand pattern in the long term. In the tables, the dependency values for a non-switching household solely using the short-term hourly change are shown as well.

Scenario 1 For the first scenario, we focus solely on the short-term demand change, as there is no peak during the day to shift. In case of a demand response of 100%, the total dependency over 2023 equals 2035.8, which is a 67% decrease from the current situation. However, it is worth to note that this would result in an extreme peak at the end of the year, as the demand gets increased every time step without having sufficient supply to cover it. This is another reason why this total demand change is not a realistic scenario; if the year would continue into 2024, this would result in people stalling all their energy usage until there is enough supply again, which could take several months. When using the more realistic demand shift, the result can be seen in Figures 48-50 and Table 13.

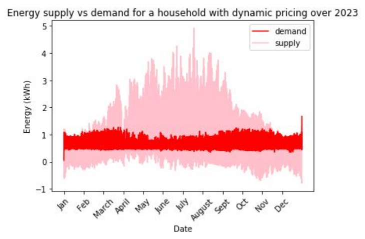

Figure 48: Result using dynamic pricing over 2023 in scenario 1

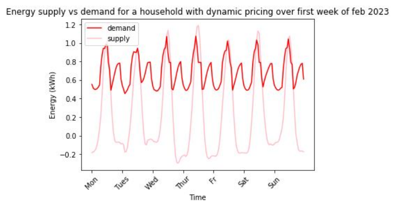

Figure 49: Result using dynamic pricing over winter week in scenario 1

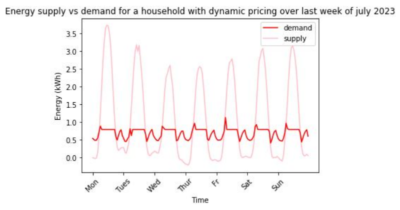

Figure 50: Result using dynamic pricing over summer week in scenario 1

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	3167.8	3072.9	6240.5
Entire year	3003.7	2908.6	5912.3
Winter week	80.6	2.7	83.3
Summer week	40.9	110.0	150.9

Table 13: Dependency amount in scenario 1 using dynamic pricing

This solution in this scenario results in a lower decrease of the dependency than the storage solution. Overall, the entire dependency decreases with 5% over 2023.

Scenario 2 In case of total demand response, the same as before happens. At the end of the year a peak occurs with all piled up demand amounts. Over the entire year, this results in a dependency of 2048 kWh with the peak shift, which is a decrease of 70%. Without the peak shift, using the original demand scenario, the short-term solution alone also results in a maximum decrease of 70%. When using the realistic scenario of demand response, the situation looks as in Figures 51-53.

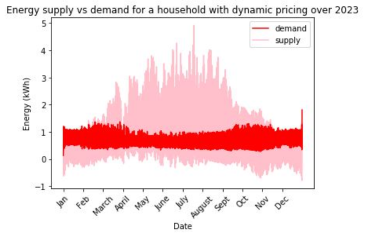

Figure 51: Result using dynamic pricing over 2023 in scenario 2

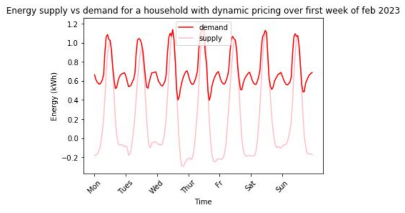

Figure 52: Result using dynamic pricing over winter week in scenario 2

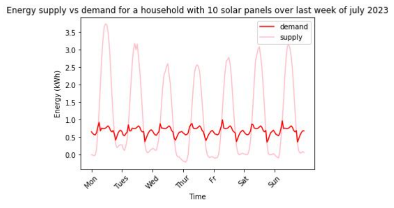

Figure 53: Result using dynamic pricing over summer week in scenario 2

In this demand scenario, the switched peak from the evening to the daytime seems to fall together well with the peak of the supply. We notice that in the summer it gets gets smoothed out under the supply curve. Even though it looks quite alright, the dependency is still relatively high. In Table 14 the dependency is shown.

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	3488.0	3344.6	6832.6
Entire year	3059.3	2916.0	5975.3
Winter week	82.9	0.3	83.2
Summer week	41.7	113.4	155.1
Entire year short-term only	3314.2	3170.8	6485.0
Winter week short-term only	88.9	6.3	95.2
Summer week short-term only	45.1	116.8	161.9

Table 14: Dependency amount in scenario 2 using dynamic pricing

For a household switching their peak demand long term, this results in a dependency decrease of 13%. When a household solely uses the short-term demand switch for the current time, the dependency decrease equals 5%.

Scenario 3 Since this scenario uses a higher demand profile, the peak at the end of the year when shifting the entire demand is also even higher. Overall, a demand response of 100% would result in an entire dependency of 3004.3; a decrease of 69%. Again, the same holds for solely using the short-term demand change with a 100% response rate. When visualizing the realistic scenario, it looks as shown in Figures 54-56.

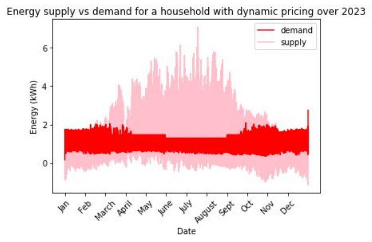

Figure 54: Result using dynamic pricing over 2023 in scenario 3

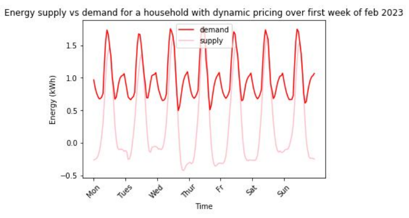

Figure 55: Result using dynamic pricing over winter week in scenario 3

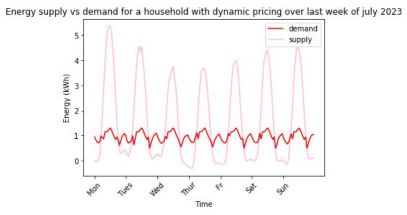

Figure 56: Result using dynamic pricing over summer week in scenario 3

Just as in last scenario, the switched demand peaks seem to match the supply. However, there is still a large dependency on the grid. This can be explained by the fact that during the night there is no supply of energy but there will always be demand, since that simply cannot be switched off during the night. The dependency can be found in Table 15.

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	4891.1	4847.7	9738.8
Entire year	4231.9	4188.6	8420.5
Winter week	115.0	0.4	115.5
Summer week	58.3	163.8	222.1
Entire year short-term only	4643.6	4600.2	9243.8
Winter week short-term only	122.6	8.1	130.7
Summer week short-term only	64.5	170.0	234.5

Table 15: Dependency amount in scenario 3 using dynamic pricing

The total dependency using long-term shift of peak demand decreases with 14%. When solely using short-term change, the dependency decrease equals 5%. This is similar to the previous scenario.

Overall, dynamic pricing seems to have less of an impact on the dependency. Long-term changing of the peak hours does have a larger influence, but still small compared to the storage solution. There is a possibility of increasing this as demand response increases; however, at one point electricity usage is required and one cannot keep procrastinating it. We have now only looked into very short-term influence of the price and a long-term total switch. One thing that would be relevant would be to look at the influence on for example the next 12 hours.

4.3.3 Combination of storage solution and dynamic pricing

As both solutions decrease the total dependency of a household, it is also relevant to investigate whether this would improve if both solutions are used at the same time. In this case, we assume that first the dynamic pricing influences the demand behaviour; after this the flywheel storage solution is implemented using the new demand of the household.

Scenario 1 For the first demand scenario, implementing both dynamic pricing and a flywheel storage system returns the results shown in Figures 57-59 and Table 16.

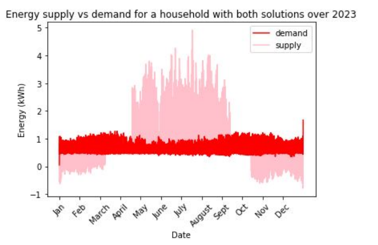

Figure 57: Result using both solutions over 2023 in scenario 1

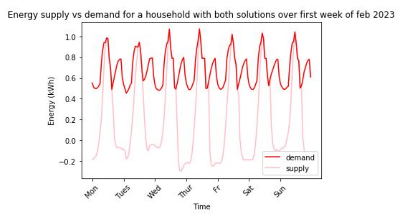

Figure 58: Result using both solutions over winter week in scenario 1

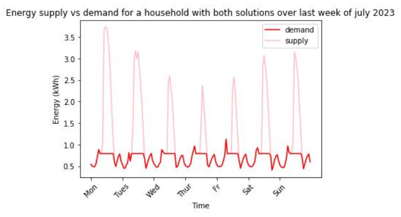

Figure 59: Result using both solutions over summer week in scenario 1

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	3167.8	3072.9	6240.5
Entire year	1615.8	1520.7	3136.5
Winter week	78.0	0	78.0
Summer week	0	68.9	68.9

Table 16: Dependency amount in scenario 1 using dynamic pricing and a flywheel storage solution

Using both solutions, the total dependency over 2023 decreases with 50%, compared to also 50% using only a storage solution and 5% using only dynamic pricing. Combining these two solutions does not significantly improve the result compared to solely using a storage solution; the dependency decreased with only 1 kWh.

Scenario 2 When implementing both solutions using the second demand scenario, the results can be seen in Figures 60-62 and Table 17.

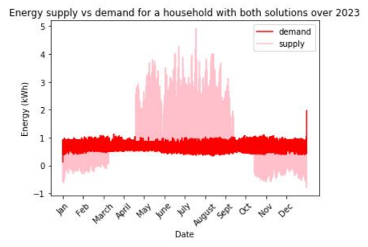

Figure 60: Result using both solutions over 2023 in scenario 2

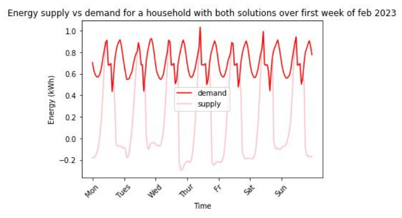

Figure 61: Result using both solutions over winter week in scenario 2

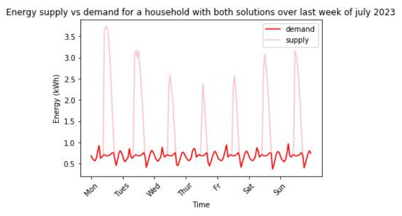

Figure 62: Result using both solutions over summer week in scenario 2

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	3488.0	3344.6	6832.6
Entire year	1686.5	1543.2	3229.7
Winter week	82.6	0	82.6
Summer week	0	71.5	71.5
Entire year short-term only	1687.5	1544.1	3231.6
Winter week short-term only	82.6	0	82.6
Summer week short-term only	0	71.5	71.5

Table 17: Dependency amount in scenario 2 using dynamic pricing and a flywheel storage solution

The total dependency over 2023 decreases with 53%, while using solely the storage solution returns a decrease of 53% and using dynamic pricing returns a 5% to 13% decrease. Again, using solely the storage solution gives the same impact on the dependency decrease.

Scenario 3 For the last demand scenario, the results of implementing both solutions can be found in Figures 63-65 and Table 18.

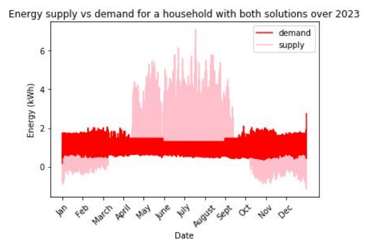

Figure 63: Result using both solutions over 2023 in scenario 3

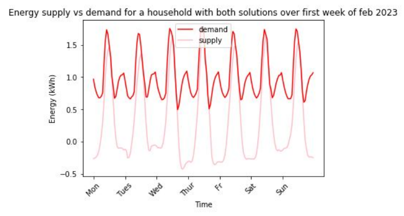

Figure 64: Result using both solutions over winter week in scenario 3

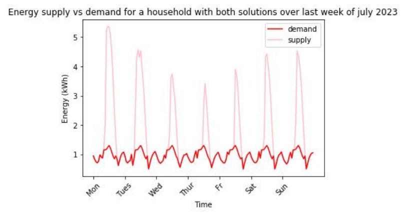

Figure 65: Result using both solutions over summer week in scenario 3

Time	Energy shortage (kWh)	Energy surplus (kWh)	Total dependency (kWh)
Entire year: current situation	4891.1	4847.7	9738.8
Entire year	2390.4	2347.0	4737.4
Winter week	114.6	0	114.6
Summer week	0	105.2	105.2
Entire year short-term only	2392.1	2348.7	4740.8
Winter week short-term only	114.6	0	114.6
Summer week short-term only	0	105.3	105.3

Table 18: Dependency amount in scenario 3 using dynamic pricing and a flywheel storage solution

Using both solutions in this scenario decreases the dependency with 51%. Using solely a storage solution also gives a dependency decrease of 51%, while using only dynamic pricing decreases the dependency by 5% to 14%. Again, adding dynamic pricing does not seem to have an influence.

In no scenario, the dynamic pricing contributes to the dependency decrease. The decrease obtained by using a storage solution is, in percentages, the same for all three scenarios.

4.4 Summary of possible solutions for one household

Overall, a storage solution seems to be a good solution with high impact on the dependency. The dynamic pricing also influences the dependency, but significantly less. In Table 19 an overview of the dependency decrease per solution is given.

Possible solution	Demand scenario	Dependency decrease
Storage solution	Scenario 1	50%
Storage solution	Scenario 2	53%
Storage solution	Scenario 3	51%
Dynamic pricing short-term only	Scenario 1	5%
Dynamic pricing short-term only	Scenario 2	5%
Dynamic pricing short-term only	Scenario 3	5%
Dynamic pricing including long-term	Scenario 2	13%
Dynamic pricing including long-term	Scenario 3	14%
Both solutions	Scenario 1	50%
Both solutions	Scenario 2	53%
Both solutions	Scenario 3	51%

Table 19: Dependency decrease in percentage per possible solution for one household

As denoted before, dynamic pricing does not seem to have any impact when combined with a storage solution. However, we will look at what happens when including it in the entire neighborhood and possibly increasing the response rate. When looking at implementing solely one solution, the storage solution has a higher impact on the dependency decrease.

4.5 Simulation of neighborhood with combined demand scenarios

When looking at a neighborhood of houses, we stated this as being thirty households in Assumption 1.4. We assume they operate the same as one household; in the sense that the electricity grid is connected everywhere and does not operate in a different way. This means that our assumption is that we simply add the supply and demand of all houses and then investigate the difference between that, and thus the total dependency of that neighborhood on the bigger grid that the entire neighborhood is connected to.
In this neighborhood, a random amount of households are assigned to follow demand profiles corresponding the the three different demand scenarios. The sum of the demand of these different households then equals the total demand for the neighborhood. Each house is assumed to have the number of solar panels corresponding with their energy usage, i.e. 16 solar panels for houses following demand scenario 1 or 2 and 23 solar panels for houses following demand scenario 3. The sum of this supply per household is then assumed to be the total supply for the neighborhood.

4.5.1 Current situation

Firstly, the current situation will be discussed. As the number of houses for each demand profile is randomly generated, we run this 5000 times and then take the averages to compute the average shortage, surplus and thus average total dependency. It is chosen to run this simulation 5000 times such that an adequate amount of runs is made for the thirty random households, but it also takes a suitable amount of time to run it. For the plots, it is visually more desirable to see less runs, so only the first ten runs are displayed in the plot.

When using all demand scenarios in the neighborhood, and assigning the households a demand profile at random, the results can then be seen in Figures 66-68 and Table 20.

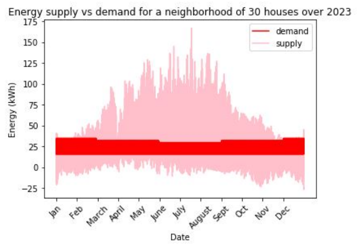

Figure 66: Result of current situation for a neighborhood of thirty houses over 2023

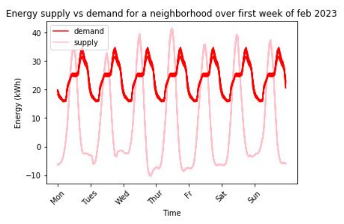

Figure 67: Result of current situation for a neighborhood of thirty houses over winter week in 2023

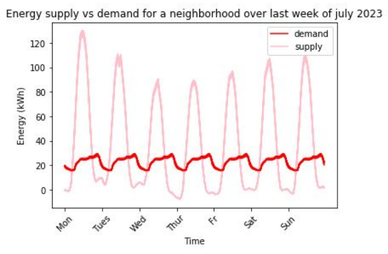

Figure 68: Result of current situation for a neighborhood of thirty houses over summer week in 2023

Time	Shortage (kWh)	Surplus (kWh)	Dependency (kWh)	Dependency per household (kWh)
Entire year	115345.8	112521.2	227866.9	7595.6
Winter week	3022.9	272.1	3295.0	109.8
Summer week	1629.6	4094.1	5723.7	190.8

Table 20: Average dependency amount over 5000 runs for a neighborhood with 30 houses of all demand scenarios

Notice that the average dependency of a household after 5000 runs is now 7595.6 kWh. If we simply take the average of the three scenarios, this would result in an average dependency of 7604.0 kWh. This indeed looks similar, which is what we expect after many runs taking random numbers of houses with the different demand scenarios.

4.5.2 Using possible solutions

Both solutions we investigated on household level, dynamic pricing and a storage solution, will also be incorporated in the entire neighborhood. For dynamic pricing, we will let half of the houses with scenario 2 and 3 have the adapted peak usage during the day instead of at night. After that, the total short-term shift is implemented for the total demand of the entire neighborhood. For the storage solution, we use one large system that takes the entire demand and supply of the neighborhood into account. Since we assumed the maximum capacity of a flywheel system was 130 kWh, we will now assume that for the entire neighborhood our maximum capacity is thirty times this amount. We assume that households are not willing to pay for more than one system. However, again all energy can be shared with all houses, so one flywheel is not attached to one specific house only.

Neighborhood with storage solution and dynamic pricing Previously, we stated that for all households, adding dynamic pricing on top of the storage solution did not contribute in the decrease of the dependency. However, we will also check whether this is the case when simulating the entire neighborhood. The results of using both solutions are shown in Figures 69-71 and Table 21.

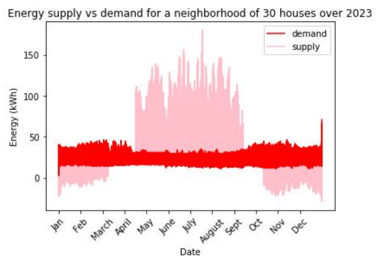

Figure 69: Result of using both solutions for a neighborhood of thirty houses over 2023

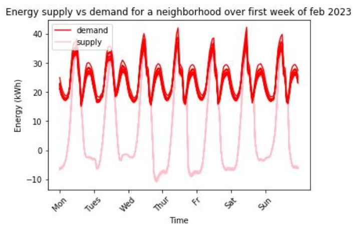

Figure 70: Result of using both solutions for a neighborhood of thirty houses over winter week in 2023

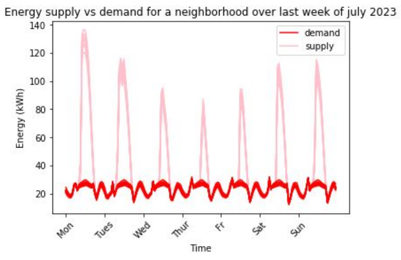

Figure 71: Result using both solutions for a neighborhood of thirty houses over summer week in 2023

Time	Shortage (kWh)	Surplus (kWh)	Dependency (kWh)	Dependency per household (kWh)
Entire year	59107.7	56266.1	115373.8	3845.8
Winter week	2723.0	20.7	2743.7	91.5
Summer week	47.1	2503.8	2550.9	85.0

Table 21: Average dependency amount over 5000 runs for a neighborhood with 30 houses of all demand scenarios using dynamic pricing and a flywheel storage solution

Using these solutions on a neighborhood with all three demand scenarios decreases the dependency on the grid of the entire neighborhood with 49% over 2023. Notice that during the winter week, the dependency decrease is only 17%, while for the summer week the decrease equals 55%. Over the entire year it can be seen that the surplus and shortage are approximately equal. The shortage for a neighborhood over the year 2023 is decreased with 49%, while the surplus decreases with 50%. Currently for a household it is more desirable to reduce the shortage, since getting electricity from the grid costs more than putting it back into the grid, for which now people even get money. Even when the point comes where people will have to start paying to return their energy to the grid, this will most likely still be less than getting energy from the grid. So for the costs of the household reducing the shortage is more important than reducing the surplus; however, for the overall congestion of the grid this will have a more negative influence, so it being equally decreased now is an overall desirable result.

We also want to look into what happens when using solely the storage solution or the dynamic pricing solution. In Table 22, the dependency over 2023 using only one solution is displayed.

Solution	Shortage (kWh)	Surplus (kWh)	Dependency (kWh)	Dependency decrease
Storage solution	59441.4	46643.9	116085.3	49%
Dynamic pricing	106410.7	103114.6	209525.3	8%

Table 22: Average dependency amount over 5000 runs for a neighborhood with 30 houses of all demand scenarios for both solutions separately

Notice that again the storage solution alone also gives a 49% decrease in dependency. The dynamic pricing does decrease the dependency a little bit, from 116.085 kWh to 115.374 kWh, but not enough to have a significant influence. When solely using dynamic pricing, it decreases the dependency over a year by 8%. Overall, implementing only a storage solution would be the best option.

Neighborhood with a solution obtaining a 70% dependency decrease The goal set in Section 1.3 was to decrease the dependency with 70%. Using the realistic solutions we have thus far been able to reduce the dependency of an entire neighborhood with 49%. Now, we will look into increasing the response rate for dynamic pricing and the capacity of the storage solutions to see what is needed to obtain the 70% decrease. We start with the neighborhood using both solutions as we deemed realistic and adapt from there.

When keeping the storage solution at its current maximum capacity, only adapting the demand response cannot get us to a 70% decrease. We set the demand response to 100%; then the dependency decrease equals 69%. However, as happened previously on a household level, the demand now gets piled up to a very large peak at the end of the year and is thus not realistic at all.
Keeping the demand response at our previous 44%, with half of the houses shifting long-term, does not add onto the dependency decrease compared to solely using a storage solution. When wanting to obtain a 70% decrease using only the storage solution, a storage capacity of approximately 28.000 kWh is needed. This equals 215 flywheels, more than 7 per household, or 6 neighborhood batteries, that can go up to 5 MW ([13]). Unfortunately, increasing the demand response does not mean the maximum capacity can decrease to still obtain the 70% decrease. We will thus look into the case of only using a storage solution with the capacity of 28.000 kWh. The results of this are shown in Figures 72-74 and Table 23.

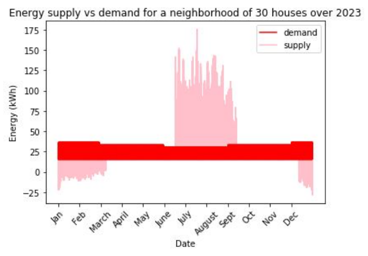

Figure 72: Result using a storage solution of 2800 kWh for a neighborhood over 2023

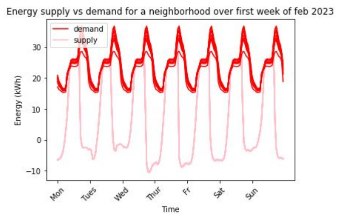

Figure 73: Result using a storage solution of 2800 kWh for a neighborhood over winter week in 2023

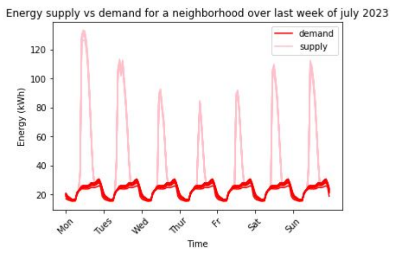

Figure 74: Result using a storage solution of 2800 kWh for a neighborhood over summer week in 2023

Time	Shortage (kWh)	Surplus (kWh)	Dependency (kWh)	Dependency per household (kWh)
Entire year	35885.0	33028.6	68913.6	2297.1
Winter week	2776.3	24.3	2800.6	93.4
Summer week	49.2	2507.6	2556.8	85.2

Table 23: Average dependency amount over 5000 runs for a neighborhood with 30 houses using a storage solution of 28000 kWh

As desired, this enormous storage solution results in a dependency decrease of 70%.

4.6 Summary of possible solutions for a neighborhood

For a neighborhood consisting of 30 houses, we have now looked into several solutions. A summary of the results is given in Table 24.

Possible solution	Dependency (kWh)	Dependency decrease
Current situation: no solution	227866.9	
Storage solution	116085.3	49%
Dynamic pricing	209525.3	8%
Both solutions	115373.8	49%
A storage solution of 28.000kWh	68913.6	70%

Table 24: Dependency decrease in percentage per possible solution for a neighborhood of 30 houses

5 Discussion

In the previous section, the results of our simulation can be found, from which we now discuss the main findings.

5.1 Results from simulation

When focusing on household level, the three different demand scenarios provide similar outcomes. Scenario 3 is taken as a high demand scenario, which could become more realistic in the future. Thus, adding more solar panels onto that house is needed to obtain enough energy supply. After doing this, the dependency becomes larger, but the pattern over time stays the same.
We looked into two different solutions, where a storage solution seems to be the better option. Not only when looking at a current realistic scenario, but also for the possibilities when battery technology advances the storage solution seems to have a very high potential. It was expected that implementing both the storage solution and the dynamic pricing would improve the solution even more. However, dynamic pricing does not have a significant impact when added onto a storage solution. It does have an impact when implemented on its own, though significantly smaller than the impact of a storage solution. This could also be because of the fact that only very short-term and a total long-term change was implemented. The outcome of this solution is likely to change when for example the next 12 hours of pricing are taken into account, such that people will schedule their upcoming day around it. It can already be seen that implementing a 100% short-term demand change results in a possible decrease of around 70%. Thus, we expect that implementing an influence on several hours instead of just one will return a larger dependency decrease than the current 5% to 14%.

5.2 The ISBEP group project

As mentioned in Section 1.1, this report is part of a larger project. In this project, a flywheel storage solution was used as this was advised by a mechanical engineer in the group. He has looked into several storage solutions and found this to be the best. The other mechanical engineer has focused on individual appliances in a household and scheduling those; if more information is gathered on the energy usage of the clients, this will also give a better overview of the current situation and possible shift for all appliances used. In this report specifically, this could result in a better and more realistic version of the demand change implemented. Lastly, a software scientist has looked into short term predicting, which can be more precise. He has also gathered insights on the possible savings for households. By combining this short term forecasting with the overall overview of the supply behaviour throughout the year, a better recommendation could be made to clients.

5.3 Energy supply source

For the results in this project, we have focused on obtaining energy supply via solar panels. However, it is also worth noting that in the Netherlands wind energy can and will also play a big role. As wind is less dependent on both daily and yearly fluctuations, this might be a more stable base for energy supply. However, this will most likely be implemented on larger scales than on a 30 house neighborhood level. Thus it might still cause a dependency on the grid when looking at smaller scales, but this is worth investigating further.

6 Conclusion and future work

6.1 Conclusion

In conclusion, we simulated a neighborhood of thirty houses for the current situation and implemented possible solutions. It was found that the best option is to use a storage solution. With the current possibilities of one flywheel per household, this would result in a 49% decrease of dependency. The desired 70% decrease can be achieved with a storage solution that has a capacity of 28.000kWh, which equals 215 flywheels or 6 neighborhood batteries.

Supply To get to this result, first a linear regression model was created for the solar radiation, and thus the solar energy supply. Two models were made based on time and air temperature variables. For time, periodic cycles of both a day and a year were used. The first model used a regression model for temperature as input for the variable. The best model, however, used data on the temperature over 2022. This model returns an adjusted R-squared of 0.774. We multiplied this model by several variables to obtain the amount of kWh a household generates with their solar panels from the radiation in Joules/cm^2.

Demand For the demand of a household, three scenarios were created. Two of those used the current average usage of a household with an added heat pump on top of that to compensate for the current use of gas for heating. The first scenario consists solely of a peak and off-peak demand, while the second scenario follows a traditional pattern with the peak usage in the evening. The first one is more simplistic, but could be more suitable for people working from home. The third scenario looked into a possible future scenario when both heat pumps and electric vehicles become part of the household, which results in a high electricity demand. This high demand does follow the same traditional pattern with a peak in the evening.

Simulation To create a neighborhood, a random number of houses with the three different scenarios were taken. It was assumed that all houses contained the number of solar panels that they needed for their demand, namely 16 for demand scenarios 1 and 2 and 23 for demand scenario 3. Both on this neighborhood and on household level, the dependency on the grid was determined for the current scenario. Then a flywheel storage solution and dynamic pricing were implemented. Adding the storage solution resulted in a decrease in dependency of around 50%, both on household and neighborhood level. When adding dynamic pricing, the dependency decreased with around 5% to 14% for households, resulting in an 8% decrease for the total neighborhood. So when choosing either option, currently the flywheel storage solution would be desired. When implementing both solutions, the decrease in dependency was equal to only using a storage solution, around 50%.

6.2 Future work

On all sections of this report, there are some possibilities to extend our research and make the situations more realistic. We will go through them to state what could be done to possibly improve the section and finally give a better outcome.
On the supply part, it would be interesting to see if the linear regression model can be made to fit better. This could for example be done by adding explanatory variables such as humidity, clouds and other weather conditions. For demand, obtaining a more precise demand scenario of the current clients would be ideal. If the company could get insights in how their clients use their energy during the day, more accurate demand scenarios could be created. Lastly, on the simulation, the current solutions implemented are simplistic with a lot of assumptions. When making these more realistic, the dynamic pricing solution could change a lot by taking the effect on multiple hours into account. The storage solution could also be made more realistic by implementing a maximum amount of time the energy can be stored, and even a decrease in energy put back into the house as some energy will be lost. When creating the neighborhood, we will want to investigate how the electricity grid connects all houses together. In this report, it was assumed that supply and demand was summed up for all houses and could be distributed over all houses in any way we wanted. However, realistically, this is not the case and there are likely some obstacles in the way houses are connected.

7 Advice

For the current situation, implementing the two solutions described in this report, a flywheel storage solution and dynamic pricing, would decrease the dependency by approximately 50%. However, an immediate note is that this is based on three general demand scenarios. A first advice would namely be to get to know your clients better; if possible, using data from smart meters would create a much more precise overview of the current demand. To implement a flywheel storage system, the advice is to simply recommend it to the clients of the company and let another company install them. Financially, this would not cost the company anything except for the time put into recommending this solution to the client. However, there is also the chance that people will not go for it. For dynamic pricing, more investment is needed. This would mean creating, or letting someone create, an app for users to check their current electricity prices. Since large electricity suppliers also fall under the clients of the company, it could again simply be stated as a recommendation without immediately investing in it. For the private clients, i.e. the households, using an app and changing their behaviour when they can is easier done than investing in a flywheel storage solution.
Overall, the storage solution has a larger influence on the dependency and a large possibility of scaling up with the advancing technology. If a choice had to be made, this would therefore be the best option out of two.

8 References

[1] Amar A Abd El-Sallam' and Salim Kayhan. *Bootstrap and Backward Elimination Based Approaches For Model Selection.* Tech. rep.

[2] Aníbal de Almeida et al. *Residential Monitoring to Decrease Energy Use and Carbon Emissions in Europe.* Tech. rep.

[3] Ahmad Arabkoohsar. "Classification of energy storage systems". In: (2021). DOI: 10.1016/B978-0-12-820023-0.00001-8. URL: https://doi.org/10.1016/B978-0-12-820023-0.00001-8.

[4] Johan Bring. *Variable importance by partitioning R 2.* Tech. rep. 1995, pp. 173–189.

[5] Ahmad Faruqui and Sanem Sergici. *Household response to dynamic pricing of electricity: A survey of 15 experiments.* 2010. DOI: 10.1007/s11149-010-9127-y.

[6] *Gemiddeld energieverbruik in Nederland — Milieu Centraal.* URL: https://www.milieucentraal.nl/energie-besparen/inzicht-in-je-energierekening/gemiddeld-energieverbruik/.

[7] *Gemiddeld stroomverbruik 2023 — Is jouw verbruik hoog of laag?* URL: https://www.groene-energie.blog/energie/gemiddeld-stroomverbruik/.

[8] *Hoeveel zonnepanelen heb ik nodig? — Consumentenbond.* URL: https://www.consumentenbond.nl/zonnepanelen/hoeveel-zonnepanelen.

[9] *How Does Temperature Affect Solar Panel Energy Production? - Ilum Solar.* URL: https://ilumsolar.com/how-does-temperature-affect-solar-panel-energy-production/.

[10] Ulrich Knief and Wolfgang Forstmeier. "Violating the normality assumption may be the lesser of two evils". In: (). DOI: 10.3758/s13428-021-01587-5/Published. URL: https://doi.org/10.3758/s13428-021-01587-5.

[11] KNMI. *Uurwaarden van weerstations.* URL: https://www.daggegevens.knmi.nl/klimatologie/uurgegevens.

[12] *KNMI - Zomerse dagen.* URL: https://www.knmi.nl/kennis-en-datacentrum/uitleg/zomerse-dagen.

[13] *Neighbourhood batteries.* URL: https://www.energy.vic.gov.au/renewable-energy/batteries-energy-storage-projects/neighbourhood-batteries.

[14] Nibud. *Kosten van energie en water - Nibud.* URL: https://www.nibud.nl/onderwerpen/uitgaven/kosten-energie-water/.

[15] M M Rahman, M Hasanuzzaman, and N A Rahim. "Effects of various parameters on PV-module power and efficiency". In: (). DOI: 10.1016/j.enconman.2015.06.067. URL: http://dx.doi.org/10.1016/j.enconman.2015.06.067.

[16] Amand F Schmidt and Chris Finan. "Linear regression and the normality assumption". In: (). DOI: 10.1016/j.jclinepi.2017.12.006. URL: https://doi.org/10.1016/j.jclinepi.2017.12.006.

[17] K. Sudhakar and R. Mamat. "Artificial Leaves: Towards Bio-Inspired Solar Energy Converters". In: *Reference Module in Earth Systems and Environmental Sciences* (2019). DOI: 10.1016/B978-0-12-409548-9.11799-3.

[18] *Uniek, arrange your connections for your new home in 2 minutes.* URL: https://www.aansluitingregelen.nl/wl/corporate-housing-solutions-en1117/meerinformatie.php?pakket=stroom-gas_1&U=.

[19] Oscar Van Vliet et al. "Energy use, cost and CO 2 emissions of electric cars". In: *Journal of Power Sources* 196 (2011), pp. 2298–2310. DOI: 10.1016/j.jpowsour.2010.09.119.

[20] Vattenfall. *Zoveel zonnepanelen heb ik nodig — Vattenfall.* URL: https://www.vattenfall.nl/kennis/hoeveel-zonnepanelen-heb-ik-nodig/.

[21] Else Veldman et al. "Scenario-based modelling of future residential electricity demands and assessing their impact on distribution grids". In: (2013). DOI: 10.1016/j.enpol.2012.12.078. URL: http://dx.doi.org/10.1016/j.enpol.2012.12.078.

[22] *Volledig elektrische warmtepomp: alle info — Milieu Centraal.* URL: https://www.milieucentraal.nl/energie-besparen/duurzaam-verwarmen-en-koelen/volledige-warmtepomp/?gclid=EAIaIQobChMIq86F-_rq_QIVWA8GAB0JsQqMEAMYASAAEgIvq_D_BwE.

9 Appendix

9.1 Code in R for supply

9.1.1 Code for solar radiation models

```
library(ggplot2)
library(scales)
library(ggbreak)
library(lubridate)
library(dplyr)
library(readr)
library(broom)
library(MASS)
library(fitdistrplus)
library("car")
library(tidyr)
library(olsrr)

df <- `data.for.r.solar.radiation+temperature.2012.2022.hourly`
df$YYYYMMDD <- as.Date(as.character(df$YYYYMMDD), format = "%Y%m%d")
df$Date <- as.POSIXct(paste(df$YYYYMMDD, df$H), format="%Y-%m-%d %H")
which(! complete.cases(df))
df <- subset(df, select = -c(YYYYMMDD,H))
#Add "missing" values manually
df$Date[2018] <- "2012-03-25 02:00:00"
df$Date[10922] <- "2013-03-31 02:00:00"
df$Date[19658] <- "2014-03-30 02:00:00"
df$Date[28394] <- "2015-03-29 02:00:00"
df$Date[37130] <- "2016-03-27 02:00:00"
df$Date[45866] <- "2017-03-26 02:00:00"
df$Date[54602] <- "2018-03-25 02:00:00"
df$Date[63506] <- "2019-03-31 02:00:00"
df$Date[72242] <- "2020-03-29 02:00:00"
df$Date[80978] <- "2021-03-28 02:00:00"
df$Date[89714] <- "2022-03-27 02:00:00"
df <- df[c("Date", "Q", "T")]

plot(df$Date, df$Q, xlab="Date", ylab="Radiation (J/cm^2)")
title("Solar radiation in De Bilt")

#just to see R-squared value, try linear model
lm <- lm(df$Q ~ df$Date, data=df)
summary(lm)

#create polynomial regression model
df$date2 <- as.numeric(df$Date)
df$date2

model1 <- lm(df$Q ~ df$T + poly(df$date2 * df$T, 7),data=df)
summary(model1)

radiationPredict1 <- predict(model1)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
```

```
title("Data 2012-2022 with polynomial regression of degree 7")
lines(df$Date, radiationPredict1, col="red")

res1<- resid(model1)
plot(fitted(model1), res1)


#create sine/cosine model with period of a year
c <-cos(2*pi*df$date2/(3600*24*365))
s <- sin(2*pi*df$date2/(3600*24*365))
c2 <- cos(2*2*pi*df$date2/(3600*24*365))
s2 <- sin(2*2*pi*df$date2/(3600*24*365))
c3 <- cos(2*3*pi*df$date2/(3600*24*365))
s3 <- sin(2*3*pi*df$date2/(3600*24*365))
c4 <- cos(2*4*pi*df$date2/(3600*24*365))
s4 <- sin(2*4*pi*df$date2/(3600*24*365))
c5 <- cos(2*5*pi*df$date2/(3600*24*365))
s5 <- sin(2*5*pi*df$date2/(3600*24*365))
model2 <- lm(df$Q ~ (c+  c3 ), data=df)
summary(model2)

radiationPredict2 <- predict(model2)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2012-2022 using periods of a year")
lines(df$Date, radiationPredict2, col="red")

plot(model2, which=1)

confint(model2)

#some combined model
model3 <- lm(df$Q ~ df$T * (poly(df$date2, 7) + c + s + c2 + s2 + c3 + s3 + c4 + s4),
↪  data=df)
summary(model3)


radiationPredict3 <- predict(model3)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Data 2012-2022 with combined polynomial and periodic model")
lines(df$Date, radiationPredict3, col="red")

#only looking at first week of july 2022
firstday = df$date2[df$Date== "2022-07-01 00:00:00 CET"]
lastday = df$date2[df$Date== "2022-07-07 23:00:00 CET"]
df_july <- df[(df$date2>=firstday) & (df$date2 <= lastday),]

c_j <-cos(pi*df_july$date2/(3600*24))
s_j <- sin(pi*df_july$date2/(3600*24))
c2_j <- cos(2*pi*df_july$date2/(3600*24))
s2_j <- sin(2*pi*df_july$date2/(3600*24))
c3_j <- cos(3*pi*df_july$date2/(3600*24))
s3_j <- sin(3*pi*df_july$date2/(3600*24))
c4_j <- cos(4*pi*df_july$date2/(3600*24))
```

```
s4_j <- sin(4*pi*df_july$date2/(3600*24))

model5 <- lm(df_july$Q ~ df_july$T * (c_j + s_j + c2_j + s2_j + c3_j + s3_j + c4_j + s4_j)
            ,data=df_july)
summary(model5)

radiationPredict5 <- predict(model5)

plot(df_july$Date, df_july$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("First week of july 2022 with periodic regression")
lines(df_july$Date, radiationPredict5, col="red")

#periodic regression with one day periods with temperature
c <-cos(pi*df$date2/(24))
s <- sin(pi*df$date2/(24))
c2 <- cos(2*pi*df$date2/(24))
s2 <- sin(2*pi*df$date2/(24))
c3 <- cos(3*pi*df$date2/(24))
s3 <- sin(3*pi*df$date2/(24))
c4 <- cos(4*pi*df$date2/(24))
s4 <- sin(4*pi*df$date2/(24))
c5 <- cos(5*pi*df$date2/(24))
s5 <- sin(5*pi*df$date2/(24))
c6 <- cos(6*pi*df$date2/(24))
s6 <- sin(6*pi*df$date2/(24))
model6 <- lm(df$Q ~ df$T + (c + s + c2 + s2 + c3 + s3 + c4 + s4 +c5 +s5 +c6 +s6) + df$T *
(c + s + c2 + s2 + c3 + s3 + c4 + s4 +c5 +s5 +c6 +s6), data=df)
summary(model6)

radiationPredict6 <- predict(model6)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2012-2022 including temperature variable")
lines(df$Date, radiationPredict6, col="red")

plot(model6, which=1)

plot(model6, which=3)

#periodic regression with one day periods without temperature
c <-cos(2*pi*df$date2/(3600*24))
s <- sin(2*pi*df$date2/(3600*24))
c2 <- cos(2*2*pi*df$date2/(3600*24))
s2 <- sin(2*2*pi*df$date2/(3600*24))
c3 <- cos(2*3*pi*df$date2/(3600*24))
s3 <- sin(2*3*pi*df$date2/(3600*24))
c4 <- cos(2*4*pi*df$date2/(3600*24))
s4 <- sin(2*4*pi*df$date2/(3600*24))
c5 <- cos(2*5*pi*df$date2/(3600*24))
s5 <- sin(2*5*pi*df$date2/(3600*24))

model6b <- lm(df$Q ~ c + s + c2 + s2 + c3 + s4+ c5, data=df)
summary(model6b)

radiationPredict6b <- predict(model6b)
```

```
plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2012-2022 using periods of a day")
lines(df$Date, radiationPredict6b, col="red")

plot(model6b, which=1)
plot(model6b, which=2)
plot(model6b, which=3)

confint(model6b)

#check last model for the last 5 years only
starting_day = df$date2[df$Date== "2018-01-01 00:00:00 CET"]
final_day = df$date2[df$Date== "2022-12-31 23:00:00 CET"]
df_5 <- df[(df$date2>=starting_day) & (df$date2 <= final_day),]

c <-cos(pi*df_5$date2/(3600*24))
s <- sin(pi*df_5$date2/(3600*24))
c2 <- cos(2*pi*df_5$date2/(3600*24))
s2 <- sin(2*pi*df_5$date2/(3600*24))
c3 <- cos(3*pi*df_5$date2/(3600*24))
s3 <- sin(3*pi*df_5$date2/(3600*24))
c4 <- cos(4*pi*df_5$date2/(3600*24))
s4 <- sin(4*pi*df_5$date2/(3600*24))
model7 <- lm(df_5$Q ~ df_5$T * (c + s + c2 + s2 + c3 + s3 + c4 + s4), data=df_5)
summary(model7)

radiationPredict7 <- predict(model7)

plot(df_5$Date, df_5$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2018-2022 with periodic model, periods of a day")
lines(df_5$Date, radiationPredict7, col="red")

#check only 2022
start_day = df$date2[df$Date== "2022-01-01 00:00:00 CET"]
last_day = df$date2[df$Date== "2022-12-31 23:00:00 CET"]
df_2022 <- df[(df$date2>=start_day) & (df$date2 <= last_day),]

c <-cos(pi*df_2022$date2/(3600*24))
s <- sin(pi*df_2022$date2/(3600*24))
c2 <- cos(2*pi*df_2022$date2/(3600*24))
s2 <- sin(2*pi*df_2022$date2/(3600*24))
c3 <- cos(3*pi*df_2022$date2/(3600*24))
s3 <- sin(3*pi*df_2022$date2/(3600*24))
c4 <- cos(4*pi*df_2022$date2/(3600*24))
s4 <- sin(4*pi*df_2022$date2/(3600*24))
model8 <- lm(df_2022$Q ~ (c + s + c2 + s2 + c3 + s3 + c4 + s4), data=df_2022)
summary(model8)

radiationPredict8 <- predict(model8)

plot(df_2022$Date, df_2022$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation in 2022 using periods of a day")
lines(df_2022$Date, radiationPredict8, col="red")
```

```
#Combined sine/cosine model with periods of both a day and a year
c <-cos(2*pi*df$date2/(24*3600))
s <- sin(2*pi*df$date2/(24*3600))
c2 <- cos(2*2*pi*df$date2/(24*3600))
s2 <- sin(2*2*pi*df$date2/(24*3600))
c3 <- cos(3*2*pi*df$date2/(24*3600))
s3 <- sin(3*2*pi*df$date2/(24*3600))
c4 <- cos(4*2*pi*df$date2/(24*3600))
s4 <- sin(4*2*pi*df$date2/(24*3600))

cy <-cos(2*pi*df$date2/(24*3600*365))
sy <- sin(2*pi*df$date2/(24*3600*365))
c2y <- cos(2*2*pi*df$date2/(24*3600*365))
s2y <- sin(2*2*pi*df$date2/(24*3600*365))
c3y <- cos(3*2*pi*df$date2/(24*3600*365))
s3y <- sin(3*2*pi*df$date2/(24*3600*365))
c4y <- cos(4*2*pi*df$date2/(24*3600*365))
s4y <- sin(4*2*pi*df$date2/(24*3600*365))

model9 <- lm(df$Q ~ df$T + (c + s + c2 + s2 + c3 + s3 +cy +sy +c2y +s2y +c3y +s3y) +
df$T * (c + s + c2 + s2 + c3 + s3 +cy +sy +c2y +s2y +c3y + s3y) , data=df)
summary(model9)

radiationPredict9 <- predict(model9)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2012-2022 using periods of both a year and a day, including Temp.
↪  variable")
lines(df$Date, radiationPredict9, col="red")

plot(model9, which=1)

coefficients(model9)
confint(model9)

#Combined sine/cosine model without temp with periods of both a day and a year
c <-cos(2*pi*df$date2/(24*3600))
s <- sin(2*pi*df$date2/(24*3600))
c2 <- cos(2*2*pi*df$date2/(24*3600))
s2 <- sin(2*2*pi*df$date2/(24*3600))
c3 <- cos(3*2*pi*df$date2/(24*3600))
s3 <- sin(3*2*pi*df$date2/(24*3600))
c4 <- cos(4*2*pi*df$date2/(24*3600))
s4 <- sin(4*2*pi*df$date2/(24*3600))
c5 <- cos(5*2*pi*df$date2/(24*3600))
s5 <- sin(5*2*pi*df$date2/(24*3600))

cy <-cos(2*pi*df$date2/(24*3600*365))
sy <- sin(2*pi*df$date2/(24*3600*365))
c2y <- cos(2*2*pi*df$date2/(24*3600*365))
s2y <- sin(2*2*pi*df$date2/(24*3600*365))
c3y <- cos(3*2*pi*df$date2/(24*3600*365))
s3y <- sin(3*2*pi*df$date2/(24*3600*365))
c4y <- cos(4*2*pi*df$date2/(24*3600*365))
s4y <- sin(4*2*pi*df$date2/(24*3600*365))
```

```
c5y <- cos(5*2*pi*df$date2/(24*3600*365))
s5y <- sin(5*2*pi*df$date2/(24*3600*365))

model10 <- lm(df$Q ~ (c + s + c2 + s2 +c3 +cy +c3y) , data=df)
summary(model10)

radiationPredict10 <- predict(model10)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2012-2022 using periods of both a year and a day, without Temp.
↪  variable")
lines(df$Date, radiationPredict10, col="red")

plot(model10, which=1)
confint(model10)


#regression using temperature regression model
c <-cos(2*pi*df$date2/(24*3600))
s <- sin(2*pi*df$date2/(24*3600))
c2 <- cos(2*2*pi*df$date2/(24*3600))
s2 <- sin(2*2*pi*df$date2/(24*3600))
c3 <- cos(3*2*pi*df$date2/(24*3600))
s3 <- sin(3*2*pi*df$date2/(24*3600))
c4 <- cos(4*2*pi*df$date2/(24*3600))
s4 <- sin(4*2*pi*df$date2/(24*3600))

cy <-cos(2*pi*df$date2/(24*3600*365))
sy <- sin(2*pi*df$date2/(24*3600*365))
c2y <- cos(2*2*pi*df$date2/(24*3600*365))
s2y <- sin(2*2*pi*df$date2/(24*3600*365))
c3y <- cos(3*2*pi*df$date2/(24*3600*365))
s3y <- sin(3*2*pi*df$date2/(24*3600*365))
c4y <- cos(4*2*pi*df$date2/(24*3600*365))
s4y <- sin(4*2*pi*df$date2/(24*3600*365))
c5y <- cos(5*2*pi*df$date2/(24*3600*365))
s5y <- sin(5*2*pi*df$date2/(24*3600*365))
c6y <- cos(6*2*pi*df$date2/(24*3600*365))
s6y <- sin(6*2*pi*df$date2/(24*3600*365))
c7y <- cos(7*2*pi*df$date2/(24*3600*365))
s7y <- sin(7*2*pi*df$date2/(24*3600*365))

T1 <- (109.89 - 61.56*cy - 39.56*sy + 2.32*c2y + 4.57 * s2y + 2.58*c5y + 5.33*s5y + 3.16*c6y
+ 2.66*s6y + 3.43 *s7y + 3.62*s7y)

model11 <- lm(df$Q ~ T1 + c + s + c2 + s2 + c3 + s3 +cy +sy +c2y + s2y +c3y +s3y + T1*c +
↪  T1*s
+ T1*c2 + T1*s2 + T1*c3 + T1*s3 +T1*cy +T1*sy + T1*c2y +T1*s2y + T1*c3y +T1*s3y, data=df)
summary(model11)

radiationPredict11 <- predict(model11)
max(df$date2)

plot(df$Date, df$Q, xlab="Date", ylab="Solar radiation (J/cm^2)")
title("Solar radiation 2012-2022 based on temperature model")
```

```
lines(df$Date, radiationPredict11, col="red")

plot(model11, which=1)
confint(model11)
```

9.1.2 Code for temperature models

```
library(readr)
library(broom)
library(MASS)
library(fitdistrplus)
library("car")
library(tidyr)
library(olsrr)

df <- `data.for.r.solar.radiation+temperature.2012.2022.hourly`
df$YYYYMMDD <- as.Date(as.character(df$YYYYMMDD), format = "%Y%m%d")
df$Date <- as.POSIXct(paste(df$YYYYMMDD, df$H), format="%Y-%m-%d %H")
which(! complete.cases(df))
df <- subset(df, select = -c(YYYYMMDD,H))
#Add "missing" values manually
df$Date[2018] <- "2012-03-25 02:00:00"
df$Date[10922] <- "2013-03-31 02:00:00"
df$Date[19658] <- "2014-03-30 02:00:00"
df$Date[28394] <- "2015-03-29 02:00:00"
df$Date[37130] <- "2016-03-27 02:00:00"
df$Date[45866] <- "2017-03-26 02:00:00"
df$Date[54602] <- "2018-03-25 02:00:00"
df$Date[63506] <- "2019-03-31 02:00:00"
df$Date[72242] <- "2020-03-29 02:00:00"
df$Date[80978] <- "2021-03-28 02:00:00"
df$Date[89714] <- "2022-03-27 02:00:00"
df <- df[c("Date", "T")]

plot(df$Date, df$T, xlab="Date", ylab="Temperature (0.1 degrees Celsius)")
title("Temperature in De Bilt")

#create polynomial regression model
df$date2 <- as.numeric(df$Date)

model1 <- lm(df$T ~ poly(df$date2, 7),data=df)
summary(model1)

radiationPredict1 <- predict(model1)

plot(df$Date, df$T, xlab="Date", ylab="Temperature (0.1 degrees Celsius)")
title("Data 2012-2022 with polynomial regression of degree 7")
lines(df$Date, radiationPredict1, col="red")


#periodic regression with one day periods
c <-cos(pi*df$date2/(3600*24))
s <- sin(pi*df$date2/(3600*24))
c2 <- cos(2*pi*df$date2/(3600*24))
s2 <- sin(2*pi*df$date2/(3600*24))
```

```
c3 <- cos(3*pi*df$date2/(3600*24))
s3 <- sin(3*pi*df$date2/(3600*24))
c4 <- cos(4*pi*df$date2/(3600*24))
s4 <- sin(4*pi*df$date2/(3600*24))
model6 <- lm(df$T ~ (c + s + c2 + s2 + c3 + s3 + c4 + s4), data=df)
summary(model6)

radiationPredict6 <- predict(model6)

plot(df$Date, df$T, xlab="Date", ylab="Temperature (0.1 degrees Celsius)")
title("Solar radiation 2012-2022 with periodic model")
lines(df$Date, radiationPredict6, col="red")

#create sine/cosine model with period of a year
c <-cos(2*pi*df$date2/(3600*24*365))
s <- sin(2*pi*df$date2/(3600*24*365))
c2 <- cos(2*2*pi*df$date2/(3600*24*365))
s2 <- sin(2*2*pi*df$date2/(3600*24*365))
c3 <- cos(3*2*pi*df$date2/(3600*24*365))
s3 <- sin(3*2*pi*df$date2/(3600*24*365))
c4 <- cos(4*2*pi*df$date2/(3600*24*365))
s4 <- sin(4*2*pi*df$date2/(3600*24*365))
c5 <- cos(5*2*pi*df$date2/(3600*24*365))
s5 <- sin(5*2*pi*df$date2/(3600*24*365))
c6 <- cos(6*2*pi*df$date2/(3600*24*365))
s6 <- sin(6*2*pi*df$date2/(3600*24*365))
c7 <- cos(7*2*pi*df$date2/(3600*24*365))
s7 <- sin(7*2*pi*df$date2/(3600*24*365))
c8 <- cos(8*2*pi*df$date2/(3600*24*365))
s8 <- sin(8*2*pi*df$date2/(3600*24*365))

model2 <- lm(df$T ~ (c + s + c2 + s2 + c3 + s3 + c4 + s4 +c5 +s5 +c6 +s6 + c7 +s7 +c8 +s8),
↪  data=df)
summary(model2)

radiationPredict2 <- predict(model2)
radiationPredict2[4356]

plot(df$Date, df$T, xlab="Date", ylab="Temperature (0.1 degrees Celsius)")
title("Temperature 2012-2022 using periods of a year")
lines(df$Date, radiationPredict2, col="red")


plot(model2, which=1)
plot(model2, which=2)
plot(model2, which=3)

min(df$date2)

#combined model
c <-cos(2*pi*df$date2/(3600*24))
s <- sin(2*pi*df$date2/(3600*24))
c2 <- cos(2*2*pi*df$date2/(3600*24))
s2 <- sin(2*2*pi*df$date2/(3600*24))
c3 <- cos(2*3*pi*df$date2/(3600*24))
```

```
s3 <- sin(2*3*pi*df$date2/(3600*24))
c4 <- cos(2*4*pi*df$date2/(3600*24))
s4 <- sin(2*4*pi*df$date2/(3600*24))
c5 <- cos(2*5*pi*df$date2/(3600*24))
s5 <- sin(2*5*pi*df$date2/(3600*24))

cy <-cos(2*pi*df$date2/(3600*24*365))
sy <- sin(2*pi*df$date2/(3600*24*365))
c2y <- cos(2*2*pi*df$date2/(3600*24*365))
s2y <- sin(2*2*pi*df$date2/(3600*24*365))
c3y <- cos(3*2*pi*df$date2/(3600*24*365))
s3y <- sin(3*2*pi*df$date2/(3600*24*365))
c4y <- cos(4*2*pi*df$date2/(3600*24*365))
s4y <- sin(4*2*pi*df$date2/(3600*24*365))
c5y <- cos(5*2*pi*df$date2/(3600*24*365))
s5y <- sin(5*2*pi*df$date2/(3600*24*365))


model3 <- lm(df$T ~ (c + s + c2 + s2 + c3 + s3 + c4 + s4 + c5 +s5 + cy + sy + c2y + s2y
+ c3y +s3y + c4y +s4y +c5y +s5y), data=df)
summary(model3)

radiationPredict3 <- predict(model3)

plot(df$Date, df$T, xlab="Date", ylab="Temperature (0.1 degrees Celsius)")
title("Solar radiation 2012-2022 with periodic model")
lines(df$Date, radiationPredict3, col="red")





library(readr)
library(broom)
library(MASS)
library(fitdistrplus)
library("car")
library(tidyr)
library(olsrr)
library(tidyverse)
library(dplyr)
library(lubridate)

df <- `data.for.r.solar.radiation+temperature.2012.2022.hourly`
df$YYYYMMDD <- as.Date(as.character(df$YYYYMMDD), format = "%Y%m%d")
df$Date <- as.POSIXct(paste(df$YYYYMMDD, df$H), format="%Y-%m-%d %H")
df <- subset(df, select = -c(YYYYMMDD,H))
#Add "missing" values manually
df$Date[2018] <- "2012-03-25 02:00:00"
df$Date[10922] <- "2013-03-31 02:00:00"
df$Date[19658] <- "2014-03-30 02:00:00"
df$Date[28394] <- "2015-03-29 02:00:00"
df$Date[37130] <- "2016-03-27 02:00:00"
df$Date[45866] <- "2017-03-26 02:00:00"
df$Date[54602] <- "2018-03-25 02:00:00"
```

```
df$Date[63506] <- "2019-03-31 02:00:00"
df$Date[72242] <- "2020-03-29 02:00:00"
df$Date[80978] <- "2021-03-28 02:00:00"
df$Date[89714] <- "2022-03-27 02:00:00"
df <- df[c("Date", "T")]


df <- df %>% mutate(Date = date(Date))
DayStats <- df %>% group_by(Date) %>% summarize(DayAvg = mean(T))
DayStats
plot(DayStats$Date, DayStats$DayAvg, xlab="date", ylab="average temperature per day")

DayStats$Date1 <- as.numeric(DayStats$Date)

#regression model, periods of a year
c <-cos(2*pi*DayStats$Date1/(365))
s <- sin(2*pi*DayStats$Date1/(365))
c2 <- cos(2*2*pi*DayStats$Date1/(365))
s2 <- sin(2*2*pi*DayStats$Date1/(365))
c3 <- cos(2*3*pi*DayStats$Date1/(365))
s3 <- sin(2*3*pi*DayStats$Date1/(365))
c4 <- cos(2*2*4*pi*DayStats$Date1/(365))
s4 <- sin(2*4*pi*DayStats$Date1/(365))
c5 <- cos(2*5*pi*DayStats$Date1/(365))
s5 <- sin(2*5*pi*DayStats$Date1/(365))
c6 <- cos(2*6*pi*DayStats$Date1/(365))
s6 <- sin(2*6*pi*DayStats$Date1/(365))
c7 <- cos(2*7*pi*DayStats$Date1/(365))
s7 <- sin(2*7*pi*DayStats$Date1/(365))
c8 <- cos(2*8*pi*DayStats$Date1/(365))
s8 <- sin(2*8*pi*DayStats$Date1/(365))
c9 <- cos(2*9*pi*DayStats$Date1/(365))
s9 <- sin(2*9*pi*DayStats$Date1/(365))

model <- lm(DayStats$DayAvg ~ (c + s + c2 + s2 +c5 +s5 +c6 +s6 +c7 +s7), data=DayStats)
summary(model)

radiationPredict <- predict(model)

plot(DayStats$Date, DayStats$DayAvg, xlab="Date", ylab="Temperature (0.1 degrees Celsius)")
title("Daily temperature 2012-2022 using periods of a year")
lines(DayStats$Date1, radiationPredict, col="red")

plot(model, which=1)
plot(model, which=2)
plot(model, which=3)

max(DayStats$Date1)

confint(model)
```

9.2 Code in Python for simulation

```
import pandas as pd
from numpy.ma.core import sqrt, ones, zeros, mean, std, sort, floor, cos, sin
from scipy import stats
from scipy.stats import norm
import matplotlib.pyplot as plt
from math import pi
import numpy as np
import random
from operator import add

#Temperature using daily averages regression
xT = list(range(19358, 19723, 1))
T = []
for i in range(len(xT)):
    cT = cos(2*pi*xT[i]/(365))
    sT = sin(2*pi*xT[i]/(365))
    cT2 = cos(2*2*pi*xT[i]/(365))
    sT2 = sin(2*2*pi*xT[i]/(365))
    cT3 = cos(2*3*pi*xT[i]/(365))
    sT3 = sin(2*3*pi*xT[i]/(365))
    cT4 = cos(2*4*pi*xT[i]/(365))
    sT4 = sin(2*4*pi*xT[i]/(365))
    cT5 = cos(2*5*pi*xT[i]/(365))
    sT5 = sin(2*5*pi*xT[i]/(365))
    y = 109.898 - 61.55*cT-39.56*sT+2.34*cT2 +4.58 * sT2 + 1.35*cT3 - 0.38 * sT3 + 0.34 *
    ↪  cT4 + 1.09*sT4 + 2.59*cT5 + 5.34*sT5
    T.append(y)

plt.plot(xT,T)
plt.xlabel('Time')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(xT, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Temperature (0.1 degrees Celsius)')
plt.title('Temperature prediction over 2023 from regression model')
plt.legend()
plt.show()

#temperature data compared till now 2023
temperature_hourly_2023 = pd.read_excel(r"C:\Users\20182405\Documents\Year 5\Q3+Q4 -
↪  BEP\temp data 2023 till 31-05-2023.xlsx")
temperature_hourly_2023 = temperature_hourly_2023.rename(index = lambda x: 1672527600 +
↪  x*3600)
temperature_hourly_2023 = temperature_hourly_2023.values

x = list(range(1672527600, 1704063600, 3600))
T_hourly = list(np.repeat(T, 24))
temperature_hourly_2023 = list(np.repeat(temperature_hourly_2023,1))
x_data = list(range(1672527600, 1685401200, 3600))


plt.plot(x,T_hourly, label='predicted temperatures')
plt.plot(x_data, temperature_hourly_2023, label='temperature data')
plt.xlabel('Time')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
```

```
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Temperature (0.1 degrees Celsius)')
plt.title('Temperature prediction vs data over 2023')
plt.legend()
plt.show()

#get solar radiation over 2023 from regression model
x = list(range(1672527600, 1704063600, 3600))
T_hourly = list(np.repeat(T, 24))
Q = []
for i in range(len(x)):
    c = cos(2*pi*x[i]/(24*3600))
    s = sin(2*pi*x[i]/(24*3600))
    c2 = cos(2*2*pi*x[i]/(24*3600))
    s2 = sin(2*2*pi*x[i]/(24*3600))
    c3 = cos(3*2*pi*x[i]/(24*3600))
    s3 = sin(3*2*pi*x[i]/(24*3600))
    c4 = cos(4*2*pi*x[i]/(24*3600))
    s4 = sin(4*2*pi*x[i]/(24*3600))

    cy = cos(2*pi*x[i]/(24*3600*365))
    sy = sin(2*pi*x[i]/(24*3600*365))
    c2y = cos(2*2*pi*x[i]/(24*3600*365))
    s2y = sin(2*2*pi*x[i]/(24*3600*365))
    c3y = cos(3*2*pi*x[i]/(24*3600*365))
    s3y = sin(3*2*pi*x[i]/(24*3600*365))
    c4y = cos(4*2*pi*x[i]/(24*3600*365))
    s4y = sin(4*2*pi*x[i]/(24*3600*365))
    z = -44.516 + 0.679*T_hourly[i] - 5.122*c - 7.998*s + 16.61*c2 - 3.516*s2 - 7.76*c3 +
    ↪  8.621*s3 + 0.218*c4 - 1.447*s4 + 59.676*cy + 30.011*sy - 68.62*c2y - 16.28*s2y +
    ↪  45.293*c3y + 15.94*s3y - 16.05*c4y - 12.224*s4y - 0.49*T_hourly[i]*c +
    ↪  0.324*s*T_hourly[i] - 0.0089*T_hourly[i]*c2 - 0.125*T_hourly[i]*s2 +
    ↪  0.059*T_hourly[i]*c3 - 0.0775*T_hourly[i]*s3 - 0.0019*T_hourly[i]*c4 +
    ↪  0.0252*T_hourly[i]*s4 - 0.376*T_hourly[i]*cy - 0.0868*T_hourly[i]*sy +
    ↪  0.441*T_hourly[i]*c2y + 0.061*T_hourly[i]*s2y - 0.262*T_hourly[i]*c3y -
    ↪  0.0491*T_hourly[i]*s3y + 0.078*T_hourly[i]*c4y + 0.0432*T_hourly[i]*s4y
    Q.append(z)

plt.plot(x,Q)
plt.xlabel('Time')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Solar radiation (J/cm^2)')
plt.title('Solar radiation prediction over 2023 from regression model')
plt.legend()
plt.show()

#solar supply vs average demand for one household over 2023
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q) * 0.2 * 2.77777778 * 10**-7)
```

```
supply_solar_panels = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels
daily_demand = [0.48]*7+[0.79]*16+[0.48]
y2 = daily_demand * 365

plt.plot(x,y2, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],y2[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[504:672], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
```

```
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],y2[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#solar supply vs average demand for one household over 2023 SCENARIO 2
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels
heatpump = 24*[0.37]
daily_demand_spring =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.48,
↪  0.5, 0.47, 0.46,0.48,0.48,0.45,0.35]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.55,
↪  0.61, 0.58, 0.55,0.55,0.56,0.45,0.35]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.39,
↪  0.38, 0.37, 0.38,0.39,0.45,0.45,0.35]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
```

```
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
```

```
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#solar supply vs average demand for one household over 2023 SCENARIO 3
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 23
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels
daily_demand_spring =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05]
daily_demand_winter =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05]
daily_demand_summer =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 23 solar panels over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 23 solar panels over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 23 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#using temperature data over 2022
temperature_hourly_total = pd.read_excel(r"C:\Users\20182405\Documents\Year 5\Q3+Q4 -
↪  BEP\temperature hourly.xlsx")
temperature_hourly = temperature_hourly_total[temperature_hourly_total["YYYYMMDD"]>=
↪  20220101]
temperature_hourly = temperature_hourly.drop(['YYYYMMDD'], axis=1)
temperature_hourly = temperature_hourly.rename(index = lambda x: 1640991600 +
↪  (x-87672)*3600)
temperature_hourly = temperature_hourly.values

x = list(range(1672527600, 1704063600, 3600))
temperature_hourly = list(np.repeat(temperature_hourly,1))
Q1 = []
for i in range(len(x)):
    c = cos(2*pi*x[i]/(24*3600))
    s = sin(2*pi*x[i]/(24*3600))
    c2 = cos(2*2*pi*x[i]/(24*3600))
    s2 = sin(2*2*pi*x[i]/(24*3600))
    c3 = cos(3*2*pi*x[i]/(24*3600))
    s3 = sin(3*2*pi*x[i]/(24*3600))
    c4 = cos(4*2*pi*x[i]/(24*3600))
    s4 = sin(4*2*pi*x[i]/(24*3600))

    cy = cos(2*pi*x[i]/(24*3600*365))
    sy = sin(2*pi*x[i]/(24*3600*365))
    c2y = cos(2*2*pi*x[i]/(24*3600*365))
    s2y = sin(2*2*pi*x[i]/(24*3600*365))
    c3y = cos(3*2*pi*x[i]/(24*3600*365))
    s3y = sin(3*2*pi*x[i]/(24*3600*365))
    c4y = cos(4*2*pi*x[i]/(24*3600*365))
    s4y = sin(4*2*pi*x[i]/(24*3600*365))
    z1 = -0.185 + 0.274*temperature_hourly[i] - 12.63*c - 1.653*s + 9.155*c2 - 3.18*s2 -
        ↪  4.114*c3 + 3.967*s3 + 0.7996*c4 - 0.907*s4 + 22.15*cy + 2.557*sy - 14.94*c2y -
        ↪  5.884*s2y + 1.891*c3y + 3.884*s3y - 1.399*c4y - 1.08*s4y -
        ↪  0.319*temperature_hourly[i]*c + 0.257*s*temperature_hourly[i] +
        ↪  0.0209*temperature_hourly[i]*c2 - 0.103*temperature_hourly[i]*s2 +
        ↪  0.0283*temperature_hourly[i]*c3 - 0.0446*temperature_hourly[i]*s3 -
        ↪  0.0000377*temperature_hourly[i]*c4 + 0.01597*temperature_hourly[i]*s4 -
        ↪  0.3678*temperature_hourly[i]*cy + 0.0842*temperature_hourly[i]*sy +
        ↪  0.01994*temperature_hourly[i]*c2y - 0.0156*temperature_hourly[i]*s2y +
        ↪  0.01616*temperature_hourly[i]*c3y - 0.0338*temperature_hourly[i]*s3y +
        ↪  0.0206*temperature_hourly[i]*c4y + 0.00843*temperature_hourly[i]*s4y
    Q1.append(z1)

plt.plot(x,Q1)
plt.xlabel('Time')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
 ↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Solar radiation (J/cm^2)')
plt.title('Solar radiation over 2022 from regression model, using temperature data')
plt.legend()
plt.show()

#solar supply vs average demand for one household over 2023 using data 2022 SCENARIO 1
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
 ↪  number_of_solar_panels)
y = supply_solar_panels_2022
print(sum(y))
daily_demand = [0.48]*7+[0.79]*16+[0.48]
y2 = daily_demand * 365

plt.plot(x,y2, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
 ↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply for a household with 16 solar panels over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh", "while the total
↪  surplus equals", total_surplus, "kWh. Which results in a dependency of", dependency)

#first week of feb only
plt.plot(x[696:864],y2[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[504:672], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],y2[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 1 using storage solution
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand = [0.48]*7+[0.79]*16+[0.48]
y2 = daily_demand * 365

#making storage solution and seeing how much it should be able to store at a time
#in case of unlimited storage solution run this
# battery_storage = 0
# battery = []
# for i in range(8760):
#     if y[i]>=y2[i]:                      #more supply than demand
#         surplus = y[i]-y2[i]
#         battery_storage+=surplus
#         battery.append(battery_storage)
#         y[i] -= surplus
#     elif y[i]<y2[i]:                      #more demand than supply
#         if battery_storage <= y2[i] - y[i]:
#             y[i] += battery_storage
#             battery_storage = 0
#             battery.append(battery_storage)
#         else:
#             battery_storage -= (y2[i] - y[i])
#             y[i] = y2[i]
#             battery.append(battery_storage)
#             battery.sort()
# print("The maximum amount of kWh stored at a time equals", max(battery), "which is used
↪  over ", battery.count(max(battery)), "time periods")
# print(battery)

#in case of limited storage solution run this
battery_storage = 0
max_storage = 130
for i in range(8760):
    if y[i]>=y2[i]:                      #more supply than demand
        surplus = y[i]-y2[i]
        battery_storage+=surplus
        if battery_storage > max_storage:
            y[i] -= (max_storage - (battery_storage-surplus))
            battery_storage = max_storage
        else:
            y[i] -= surplus
    elif y[i]<y2[i]:                      #more demand than supply
        if battery_storage <= y2[i] - y[i]:
            y[i] += battery_storage
            battery_storage = 0
        else:
            battery_storage -= (y2[i] - y[i])
            y[i] = y2[i]

#solar supply vs demand using battery
plt.plot(x,y2, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Hours')
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over 2023')
plt.legend()
plt.show()


#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh", "while the total
↪  surplus equals", total_surplus, "kWh. Which results in a dependency of", dependency)

#first week of feb only
plt.plot(x[696:864],y2[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over first week
↪  of feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
```

```
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],y2[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over last week
↪  of july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if y2[i]>=y[i]:
        shortage = y2[i]-y[i]
        total_shortage+=shortage
    elif y2[i]<y[i]:
        surplus = y[i] - y2[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 1 using demand change
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand = [0.48]*7+[0.79]*16+[0.48]
total_demand = daily_demand * 365

#demand change implementation:
#run this in case of total demand change
# for i in range(len(x)-1):
#     if total_demand[i]>=y[i]:                       #more demand than supply
#         total_demand[i+1] = total_demand[i+1] + (total_demand[i] - y[i])
#         total_demand[i] = total_demand[i] - (total_demand[i] - y[i])
#     elif total_demand[i]<y[i]:                      #more supply than demand
#         if total_demand[i+1]<=(y[i] - total_demand[i]):
#             total_demand[i] = total_demand[i] + total_demand[i+1]
#             total_demand[i+1] = total_demand[i+1] - total_demand[i+1]
#         else:
#             total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
#             total_demand[i] = total_demand[i] + (y[i] - total_demand[i])
```

```
#run this in case of realistic elasticity demand change
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                      #more demand than supply
        total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                     #more supply than demand
        if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])


plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh", "while the total
↪  surplus equals", total_surplus, "kWh. Which results in a dependency of", dependency)

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
```

```
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 1 using both (realistic) storage solution and demand change
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand = [0.48]*7+[0.79]*16+[0.48]
total_demand = daily_demand * 365

#realistic elasticity demand change
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                     #more demand than supply
        total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                     #more supply than demand
        if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

#realistic storage solution
battery_storage = 0
max_storage = 130
for i in range(8760):
    if y[i]>=total_demand[i]:                      #more supply than demand
        surplus = y[i]-total_demand[i]
        battery_storage+=surplus
        if battery_storage > max_storage:
            y[i] -= (max_storage - (battery_storage-surplus))
            battery_storage = max_storage
        else:
            y[i] -= surplus
    elif y[i]<total_demand[i]:                      #more demand than supply
        if battery_storage <= total_demand[i] - y[i]:
            y[i] += battery_storage
            battery_storage = 0
        else:
            battery_storage -= (total_demand[i] - y[i])
            y[i] = total_demand[i]

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh", "while the total
↪  surplus equals", total_surplus, "kWh. Which results in a dependency of", dependency)

#first week of feb only
```

```
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#solar supply vs average demand for one household over 2023 using data 2022 SCENARIO 2
size_solar_panel = 16500    # in cm^2
```

```
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
print(sum(y))
heatpump = 24*[0.37]
daily_demand_spring =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.48,
↪  0.5, 0.47, 0.46,0.48,0.48,0.45,0.35]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.55,
↪  0.61, 0.58, 0.55,0.55,0.56,0.45,0.35]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.39,
↪  0.38, 0.37, 0.38,0.39,0.45,0.45,0.35]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 16 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 2 including storage
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
```

```
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
heatpump = 24*[0.37]
daily_demand_spring =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.48,
↪  0.5, 0.47, 0.46,0.48,0.48,0.45,0.35]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.55,
↪  0.61, 0.58, 0.55,0.55,0.56,0.45,0.35]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.39,
↪  0.38, 0.37, 0.38,0.39,0.45,0.45,0.35]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#making storage solution and seeing how much it should be able to store at a time
#in case of unlimited storage solution run this
# battery_storage = 0
# battery = []
# for i in range(8760):
#     if y[i]>=total_demand[i]:                      #more supply than demand
#         surplus = y[i]-total_demand[i]
#         battery_storage+=surplus
#         battery.append(battery_storage)
#         y[i] -= surplus
#     elif y[i]<total_demand[i]:                      #more demand than supply
#         if battery_storage <= total_demand[i] - y[i]:
#             y[i] += battery_storage
#             battery_storage = 0
#             battery.append(battery_storage)
#         else:
#             battery_storage -= (total_demand[i] - y[i])
#             y[i] = total_demand[i]
#             battery.append(battery_storage)
#             battery.sort()
# print("The maximum amount of kWh stored at a time equals", max(battery), "which is used
↪  over ", battery.count(max(battery)), "time periods")
# print(battery)

#in case of limited storage solution run this
battery_storage = 0
max_storage = 130
for i in range(8760):
    if y[i]>=total_demand[i]:                      #more supply than demand
        surplus = y[i]-total_demand[i]
        battery_storage+=surplus
        if battery_storage > max_storage:
            y[i] -= (max_storage - (battery_storage-surplus))
            battery_storage = max_storage
        else:
            y[i] -= surplus
    elif y[i]<total_demand[i]:                      #more demand than supply
        if battery_storage <= total_demand[i] - y[i]:
            y[i] += battery_storage
            battery_storage = 0
        else:
            battery_storage -= (total_demand[i] - y[i])
            y[i] = total_demand[i]

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over first week
↪  of feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
```

```
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over last week
↪  of july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 2 using (short term) demand change
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
heatpump = 24*[0.37]
daily_demand_spring =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.48,
↪  0.5, 0.47, 0.46,0.48,0.48,0.45,0.35]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.55,
↪  0.61, 0.58, 0.55,0.55,0.56,0.45,0.35]
```

```
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.39,
↪  0.38, 0.37, 0.38,0.39,0.45,0.45,0.35]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#demand change implementation:
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                      #more demand than supply
        total_demand[i+1] = total_demand[i+1] + (total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - (total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                     #more supply than demand
        if total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

#run this in case of realistic elasticity demand change
# for i in range(len(x)-1):
#     if total_demand[i]>=y[i]:                      #more demand than supply
#         total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
#         total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
#     elif total_demand[i]<y[i]:                     #more supply than demand
#         if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
#             total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
#             total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
#         else:
#             total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
#             total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 10 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 2 using alternative demand scenario
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
heatpump = 24*[0.37]
daily_demand_spring = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.48, 0.5, 0.47,
↪  0.46,0.48,0.48,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.55, 0.61, 0.58,
↪  0.55,0.55,0.56,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.39, 0.38, 0.37,
↪  0.38,0.39,0.45,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#demand change implementation:
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                     #more demand than supply
        total_demand[i+1] = total_demand[i+1] + (total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - (total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                    #more supply than demand
        if total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

#run this in case of realistic elasticity demand change
# for i in range(len(x)-1):
#     if total_demand[i]>=y[i]:                     #more demand than supply
#         total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
#         total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
#     elif total_demand[i]<y[i]:                    #more supply than demand
#         if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
#             total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
#             total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
#         else:
#             total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
#             total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
```

```
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 10 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 2 using both (realistic) storage solution and demand change
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 16
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
heatpump = 24*[0.37]
daily_demand_spring =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.48,
↪  0.5, 0.47, 0.46,0.48,0.48,0.45,0.35]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.55,
↪  0.61, 0.58, 0.55,0.55,0.56,0.45,0.35]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.39,
↪  0.38, 0.37, 0.38,0.39,0.45,0.45,0.35]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
```

```
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#realistic demand change implementation:
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                     #more demand than supply
        total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                    #more supply than demand
        if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])


#realistic storage solution
battery_storage = 0
max_storage = 130
for i in range(8760):
    if y[i]>=total_demand[i]:                     #more supply than demand
        surplus = y[i]-total_demand[i]
        battery_storage+=surplus
        if battery_storage > max_storage:
            y[i] -= (max_storage - (battery_storage-surplus))
            battery_storage = max_storage
        else:
            y[i] -= surplus
    elif y[i]<total_demand[i]:                     #more demand than supply
        if battery_storage <= total_demand[i] - y[i]:
            y[i] += battery_storage
            battery_storage = 0
        else:
            battery_storage -= (total_demand[i] - y[i])
            y[i] = total_demand[i]

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#solar supply vs average demand for one household over 2023 using data 2022 SCENARIO 3
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 23
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
print(sum(y))
daily_demand_spring =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05]
daily_demand_winter =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05]
daily_demand_summer =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 23 solar panels over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
```

```
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 23 solar panels over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with 23 solar panels over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
```

```
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 3 using storage solution
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 23
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand_spring =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05]
daily_demand_winter =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05]
daily_demand_summer =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#making storage solution and seeing how much it should be able to store at a time
#in case of unlimited storage solution run this
# battery_storage = 0
# battery = []
# for i in range(8760):
#     if y[i]>=total_demand[i]:                       #more supply than demand
#         surplus = y[i]-total_demand[i]
#         battery_storage+=surplus
#         battery.append(battery_storage)
#         y[i] -= surplus
#     elif y[i]<total_demand[i]:                       #more demand than supply
#         if battery_storage <= total_demand[i] - y[i]:
#             y[i] += battery_storage
#             battery_storage = 0
#             battery.append(battery_storage)
#         else:
#             battery_storage -= (total_demand[i] - y[i])
#             y[i] = total_demand[i]
#             battery.append(battery_storage)
#             battery.sort()
# print("The maximum amount of kWh stored at a time equals", max(battery), "which is used
↪  over ", battery.count(max(battery)), "time periods")
# print(battery)

# #in case of limited storage solution run this
battery_storage = 0
max_storage = 130
for i in range(8760):
    if y[i]>=total_demand[i]:                      #more supply than demand
        surplus = y[i]-total_demand[i]
        battery_storage+=surplus
        if battery_storage > max_storage:
            y[i] -= (max_storage - (battery_storage-surplus))
            battery_storage = max_storage
        else:
            y[i] -= surplus
    elif y[i]<total_demand[i]:                     #more demand than supply
        if battery_storage <= total_demand[i] - y[i]:
            y[i] += battery_storage
            battery_storage = 0
        else:
            battery_storage -= (total_demand[i] - y[i])
            y[i] = total_demand[i]

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over first week
↪  of feb 2023')
plt.legend()
plt.show()
```

```
#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with a flywheel solution over last week
↪  of july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 3 using demand change
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 23
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand_spring =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05]
daily_demand_winter =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05]
```

```
daily_demand_summer =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#demand change implementation:
#run this in case of total demand change
# for i in range(len(x)-1):
#     if total_demand[i]>=y[i]:                      #more demand than supply
#         total_demand[i+1] = total_demand[i+1] + (total_demand[i] - y[i])
#         total_demand[i] = total_demand[i] - (total_demand[i] - y[i])
#     elif total_demand[i]<y[i]:                     #more supply than demand
#         if total_demand[i+1]<=(y[i] - total_demand[i]):
#             total_demand[i] = total_demand[i] + total_demand[i+1]
#             total_demand[i+1] = total_demand[i+1] - total_demand[i+1]
#         else:
#             total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
#             total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

#run this in case of realistic elasticity demand change
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                      #more demand than supply
        total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                     #more supply than demand
        if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 3 using alternative demand scenario
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 23
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand_spring = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
daily_demand_winter = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
daily_demand_summer = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.15, 1.15, 1.15,
↪  1.2,1.3,1.3,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#demand change implementation:
#run this in case of total demand change
# for i in range(len(x)-1):
#     if total_demand[i]>=y[i]:                      #more demand than supply
#         total_demand[i+1] = total_demand[i+1] + (total_demand[i] - y[i])
#         total_demand[i] = total_demand[i] - (total_demand[i] - y[i])
#     elif total_demand[i]<y[i]:                     #more supply than demand
#         if total_demand[i+1]<=(y[i] - total_demand[i]):
#             total_demand[i] = total_demand[i] + total_demand[i+1]
#             total_demand[i+1] = total_demand[i+1] - total_demand[i+1]
#         else:
#             total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
#             total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

#realistic demand change implementation:
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                      #more demand than supply
        total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                     #more supply than demand
        if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])
```

```
plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with dynamic pricing over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#SCENARIO 3 using both (realistic) storage solution and demand change
size_solar_panel = 16500    # in cm^2
number_of_solar_panels = 23
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)
supply_solar_panels_2022 = list(np.array(supply_per_cm2) * size_solar_panel *
↪  number_of_solar_panels)
y = supply_solar_panels_2022
daily_demand_spring =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05]
daily_demand_winter =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05]
daily_demand_summer =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y1 = daily_demand_winter * 59
y2 = daily_demand_spring * 92
y3 = daily_demand_summer * 92
y4 = daily_demand_spring * 91
y5 = daily_demand_winter * 31
total_demand = y1 + y2 + y3 + y4 + y5

#realistic demand change implementation:
for i in range(len(x)-1):
    if total_demand[i]>=y[i]:                     #more demand than supply
        total_demand[i+1] = total_demand[i+1] + 0.44*(total_demand[i] - y[i])
        total_demand[i] = total_demand[i] - 0.44*(total_demand[i] - y[i])
    elif total_demand[i]<y[i]:                    #more supply than demand
```

```
        if 0.44*total_demand[i+1]<=(y[i] - total_demand[i]):
            total_demand[i] = total_demand[i] + 0.44*total_demand[i+1]
            total_demand[i+1] = total_demand[i+1] - 0.44*total_demand[i+1]
        else:
            total_demand[i+1] = total_demand[i+1] - (y[i] - total_demand[i])
            total_demand[i] = total_demand[i] + (y[i] - total_demand[i])

#in case of limited storage solution run this
battery_storage = 0
max_storage = 130
for i in range(8760):
    if y[i]>=total_demand[i]:                     #more supply than demand
        surplus = y[i]-total_demand[i]
        battery_storage+=surplus
        if battery_storage > max_storage:
            y[i] -= (max_storage - (battery_storage-surplus))
            battery_storage = max_storage
        else:
            y[i] -= surplus
    elif y[i]<total_demand[i]:                    #more demand than supply
        if battery_storage <= total_demand[i] - y[i]:
            y[i] += battery_storage
            battery_storage = 0
        else:
            battery_storage -= (total_demand[i] - y[i])
            y[i] = total_demand[i]

plt.plot(x,total_demand, label="demand", color="red", zorder=2)
plt.plot(x,y, label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(8760):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#first week of feb only
plt.plot(x[696:864],total_demand[696:864], label="demand", color="red", zorder=2)
plt.plot(x[696:864],y[696:864], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over first week of
↪  feb 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(696,864):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#last week of july only
plt.plot(x[4896:5064],total_demand[4896:5064], label="demand", color="red", zorder=2)
plt.plot(x[4896:5064],y[4896:5064], label="supply", color="pink", zorder=1)
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a household with both solutions over last week of
↪  july 2023')
plt.legend()
plt.show()

#shortage
total_shortage=0
total_surplus=0
for i in range(4896,5064):
    if total_demand[i]>=y[i]:
        shortage = total_demand[i]-y[i]
        total_shortage+=shortage
    elif total_demand[i]<y[i]:
        surplus = y[i] - total_demand[i]
        total_surplus+=surplus
dependency= total_shortage+total_surplus
print("The total energy shortage over 2023 equals", total_shortage, "kWh. While the surplus
↪  equals", total_surplus, ", which accounts to a dependency of", dependency, "kWh.")

#overview of demand over a week scenario 1
x = list(range(1640991600, 1641596400, 3600))
daily_demand = [0.48]*7+[0.79]*16+[0.48]
```

```
y = daily_demand * 7

plt.plot(x,y, label="demand", color="blue")
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Electricity demand for a household over a week in scenario 1')
plt.legend()
plt.show()

print(sum(y)*52)

#overview of demand over a day scenario 2
x = list(range(1640991600, 1641078000, 3600))
heatpump = 24*[0.37]
daily_demand_spring =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.48,
↪  0.5, 0.47, 0.46,0.48,0.48,0.45,0.35]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
daily_demand_winter =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.55,
↪  0.61, 0.58, 0.55,0.55,0.56,0.45,0.35]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer =
↪  [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33,0.39,
↪  0.38, 0.37, 0.38,0.39,0.45,0.45,0.35]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_spring
y2 = daily_demand_winter
y3 = daily_demand_summer

plt.plot(x,y1, label="spring/fall demand")
plt.plot(x,y2, label="winter demand")
plt.plot(x,y3, label="summer demand")
plt.xlabel('Time')
ticks = ["00:00", "04:00", "08:00", "12:00", "16:00", "20:00"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=6)
plt.ylabel('Energy (kWh)')
plt.title('Electricity demand for a household in scenario 2')
plt.legend()
plt.show()

print(sum(daily_demand_spring)*91.25*2 + sum(daily_demand_summer)*91.25 +
↪  sum(daily_demand_winter)*91.25)

#demand scenario 2 adapted: (hours 16:00-23:00 become 8:00-15:00)
x = list(range(1640991600, 1641078000, 3600))
heatpump = 24*[0.37]
daily_demand_spring = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.48, 0.5, 0.47,
↪  0.46,0.48,0.48,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_spring = list(map(sum, zip(heatpump, daily_demand_spring)))
```

```
daily_demand_winter = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.55, 0.61, 0.58,
↪  0.55,0.55,0.56,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_winter = list(map(sum, zip(heatpump, daily_demand_winter)))
daily_demand_summer = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.39, 0.38, 0.37,
↪  0.38,0.39,0.45,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_summer = list(map(sum, zip(heatpump, daily_demand_summer)))
y1 = daily_demand_spring
y2 = daily_demand_winter
y3 = daily_demand_summer

plt.plot(x,y1, label="spring/fall demand")
plt.plot(x,y2, label="winter demand")
plt.plot(x,y3, label="summer demand")
plt.xlabel('Time')
ticks = ["00:00", "04:00", "08:00", "12:00", "16:00", "20:00", "24:00"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Electricity demand for a household in scenario 2')
plt.legend()
plt.show()

print(sum(daily_demand_spring)*91.25*2 + sum(daily_demand_summer)*91.25 +
↪  sum(daily_demand_winter)*91.25)

#overview of demand over a day scenario 3
x = list(range(1640991600, 1641078000, 3600))
daily_demand_spring =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05]
daily_demand_winter =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05]
daily_demand_summer =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y1 = daily_demand_spring
y2 = daily_demand_winter
y3 = daily_demand_summer

plt.plot(x,y1, label="spring/fall demand")
plt.plot(x,y2, label="winter demand")
plt.plot(x,y3, label="summer demand")
plt.xlabel('Time')
ticks = ["00:00", "04:00", "08:00", "12:00", "16:00", "20:00"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=6)
plt.ylabel('Energy (kWh)')
plt.title('Electricity demand for a household in scenario 3')
plt.legend()
plt.show()

print(sum(daily_demand_spring)*91.25*2 + sum(daily_demand_summer)*91.25 +
↪  sum(daily_demand_winter)*91.25)
```

```
#demand scenario 3 adapted: (hours 16:00-23:00 become 8:00-15:00)
x = list(range(1640991600, 1641078000, 3600))
daily_demand_spring = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
daily_demand_winter = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
daily_demand_summer = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.15, 1.15, 1.15,
↪  1.2,1.3,1.3,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
y1 = daily_demand_spring
y2 = daily_demand_winter
y3 = daily_demand_summer

plt.plot(x,y1, label="spring/fall demand")
plt.plot(x,y2, label="winter demand")
plt.plot(x,y3, label="summer demand")
plt.xlabel('Time')
ticks = ["00:00", "04:00", "08:00", "12:00", "16:00", "20:00"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=6)
plt.ylabel('Energy (kWh)')
plt.title('Electricity demand for a household in scenario 3')
plt.legend()
plt.show()

print(sum(daily_demand_spring)*91.25*2 + sum(daily_demand_summer)*91.25 +
↪  sum(daily_demand_winter)*91.25)

#Neighborhood
#initial values
size_solar_panel = 16500    # in cm^2
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)

def generate_total_neighborhood_demand(n):
    #generate a neighborhood of n houses with demand scenarios 1,2,3
    neighborhood = []
    for i in range(0, n):
        neighborhood.append(random.randint(1,3))
    houses_1 = neighborhood.count(1)
    houses_2 = neighborhood.count(2)
    houses_3 = neighborhood.count(3)

    #demand scenario 1
    daily_demand_1 = [0.48]*7+[0.79]*16+[0.48]
    demand_1 = daily_demand_1 * 365
    total_demand_1 = [i*houses_1 for i in demand_1]
    number_of_solar_panels_1 = 16
    supply_1 = list(np.array(supply_per_cm2) * size_solar_panel * number_of_solar_panels_1)
    total_supply_1 = [i*houses_1 for i in supply_1]

    #demand scenario 2
    heatpump = 24*[0.37]
    daily_demand_spring_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.3⌋
    ↪  1,0.31,0.32,0.33,0.48, 0.5, 0.47,
    ↪  0.46,0.48,0.48,0.45,0.35]
    daily_demand_spring_2 = list(map(sum, zip(heatpump, daily_demand_spring_2)))
    daily_demand_winter_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.3⌋
    ↪  1,0.31,0.32,0.33,0.55, 0.61, 0.58,
    ↪  0.55,0.55,0.56,0.45,0.35]
    daily_demand_winter_2 = list(map(sum, zip(heatpump, daily_demand_winter_2)))
    daily_demand_summer_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.3⌋
    ↪  1,0.31,0.32,0.33,0.39, 0.38, 0.37,
    ↪  0.38,0.39,0.45,0.45,0.35]
    daily_demand_summer_2 = list(map(sum, zip(heatpump, daily_demand_summer_2)))
    y1 = daily_demand_winter_2 * 59
    y2 = daily_demand_spring_2 * 92
    y3 = daily_demand_summer_2 * 92
    y4 = daily_demand_spring_2 * 91
    y5 = daily_demand_winter_2 * 31
    demand_2 = y1 + y2 + y3 + y4 + y5
    total_demand_2 = [i*houses_2 for i in demand_2]
    number_of_solar_panels_2 = 16
    supply_2 = list(np.array(supply_per_cm2) * size_solar_panel * number_of_solar_panels_2)
    total_supply_2 = [i*houses_2 for i in supply_2]

    #demand scenario 3
    daily_demand_spring_3 =
    ↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45,
    ↪  1.5, 1.35,1.25,1.15,1.1,1.05]
    daily_demand_winter_3 =
    ↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55,
    ↪  1.65, 1.5,1.3,1.2,1.1,1.05]
    daily_demand_summer_3 =
    ↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
    ↪  1.15, 1.2,1.3,1.3,1.1,1.05]
    y6 = daily_demand_winter_3 * 59
    y7 = daily_demand_spring_3 * 92
    y8 = daily_demand_summer_3 * 92
    y9 = daily_demand_spring_3 * 91
    y10 = daily_demand_winter_3 * 31
    demand_3 = y6 + y7 + y8 + y9 + y10
    total_demand_3 = [i*houses_3 for i in demand_3]
    number_of_solar_panels_3 = 23
    supply_3 = list(np.array(supply_per_cm2) * size_solar_panel * number_of_solar_panels_3)
    total_supply_3 = [i*houses_3 for i in supply_3]

    #total demand of neighborhood
    total_demand = list(map(sum, zip(total_demand_1,total_demand_2,total_demand_3)))
    total_supply = list(map(sum, zip(total_supply_1,total_supply_2,total_supply_3)))
    return total_demand,total_supply

nr_of_runs = 5000
total_household_demand = []
total_household_supply = []
for i in range(nr_of_runs):
    total_household_demand.append(generate_total_neighborhood_demand(30)[0])
    total_household_supply.append(generate_total_neighborhood_demand(30)[1])
print(total_household_supply)

for i in range(10):
    plt.plot(x,total_household_demand[i], label="_nolabel", color="red", zorder=2)
plt.plot([],[], label= "demand", color="red")
plt.plot(x,total_household_supply[i], label="supply", color="pink", zorder=1)
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a neighborhood of 30 houses over 2023')
plt.legend()
plt.show()

#average dependency
avg_shortage = []
avg_surplus=[]
for j in range(nr_of_runs):
    for i in range(8760):
        if total_household_demand[j][i]>=total_household_supply[j][i]:
            shortage = total_household_demand[j][i]-total_household_supply[j][i]
            avg_shortage.append(shortage)
        elif total_household_demand[j][i]<total_household_supply[j][i]:
            surplus = total_household_supply[j][i] - total_household_demand[j][i]
            avg_surplus.append(surplus)
average_shortage = sum(avg_shortage)/nr_of_runs
average_surplus = sum(avg_surplus)/nr_of_runs
average_dependency = average_shortage + average_surplus
print("The total avg energy shortage over 2023 equals", average_shortage, "kWh. While the
↪  surplus equals", average_surplus, ", which accounts to a dependency of",
↪  average_dependency, "kWh.")

#first week of feb only
for i in range(10):
    plt.plot(x[696:864],total_household_demand[i][696:864], label='_nolabel_', color="red",
    ↪  zorder=2)
    plt.plot(x[696:864],total_household_supply[i][696:864], label="_nolabel_",
    ↪  color="pink", zorder=1)
plt.plot([],[],label="demand", color="red")
plt.plot([],[],label="supply", color="pink")
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a neighborhood over first week of feb 2023')
plt.legend()
plt.show()

#average dependency
avg_shortage = []
avg_surplus=[]
for j in range(nr_of_runs):
    for i in range(696,864):
        if total_household_demand[j][i]>=total_household_supply[j][i]:
            shortage = total_household_demand[j][i]-total_household_supply[j][i]
            avg_shortage.append(shortage)
        elif total_household_demand[j][i]<total_household_supply[j][i]:
            surplus = total_household_supply[j][i] - total_household_demand[j][i]
            avg_surplus.append(surplus)
average_shortage = sum(avg_shortage)/nr_of_runs
average_surplus = sum(avg_surplus)/nr_of_runs
average_dependency = average_shortage + average_surplus
print("The total avg energy shortage over 2023 equals", average_shortage, "kWh. While the
↪  surplus equals", average_surplus, ", which accounts to a dependency of",
↪  average_dependency, "kWh.")


#last week of july only
for i in range(10):
    plt.plot(x[4896:5064],total_household_demand[i][4896:5064], label="_nolabel_",
     ↪  color="red", zorder=2)
    plt.plot(x[4896:5064],total_household_supply[i][4896:5064], label="_nolabel_",
     ↪  color="pink", zorder=1)
plt.plot([],[],label="demand", color="red")
plt.plot([],[],label="supply", color="pink")
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a neighborhood over last week of july 2023')
plt.legend()
plt.show()

#average dependency
avg_shortage = []
avg_surplus=[]
for j in range(nr_of_runs):
    for i in range(4896,5064):
        if total_household_demand[j][i]>=total_household_supply[j][i]:
            shortage = total_household_demand[j][i]-total_household_supply[j][i]
            avg_shortage.append(shortage)
        elif total_household_demand[j][i]<total_household_supply[j][i]:
            surplus = total_household_supply[j][i] - total_household_demand[j][i]
            avg_surplus.append(surplus)
average_shortage = sum(avg_shortage)/nr_of_runs
average_surplus = sum(avg_surplus)/nr_of_runs
average_dependency = average_shortage + average_surplus
print("The total avg energy shortage over 2023 equals", average_shortage, "kWh. While the
↪  surplus equals", average_surplus, ", which accounts to a dependency of",
↪  average_dependency, "kWh.")


#Neighborhood with (realistic) demand change and storage solution
#supply per household
size_solar_panel = 16500    # in cm^2
x = list(range(1672527600, 1704063600, 3600))
supply_per_cm2 = list(np.array(Q1) * 0.2 * 2.77777778 * 10**-7)

def generate_total_neighborhood_demand(n):
```

```
#generate a neighborhood of n houses with demand scenarios 1,2,3
neighborhood = []
for i in range(0, n):
    neighborhood.append(random.randint(1,3))
houses_1 = neighborhood.count(1)
houses_2 = neighborhood.count(2)
houses_3 = neighborhood.count(3)

#demand scenario 1
daily_demand_1 = [0.48]*7+[0.79]*16+[0.48]
demand_1 = daily_demand_1 * 365

#add all houses with demand scenario 1
total_demand_1 = [i*houses_1 for i in demand_1]
number_of_solar_panels_1 = 16
supply_1 = list(np.array(supply_per_cm2) * size_solar_panel * number_of_solar_panels_1)
total_supply_1 = [i*houses_1 for i in supply_1]


#demand scenario 2
heatpump = 24*[0.37]
daily_demand_spring_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.3⌋
↪  1,0.31,0.32,0.33,0.48, 0.5, 0.47,
↪  0.46,0.48,0.48,0.45,0.35]
daily_demand_spring_2 = list(map(sum, zip(heatpump, daily_demand_spring_2)))
daily_demand_winter_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.3⌋
↪  1,0.31,0.32,0.33,0.55, 0.61, 0.58,
↪  0.55,0.55,0.56,0.45,0.35]
daily_demand_winter_2 = list(map(sum, zip(heatpump, daily_demand_winter_2)))
daily_demand_summer_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.27,0.29,0.35,0.32,0.3⌋
↪  1,0.31,0.32,0.33,0.39, 0.38, 0.37,
↪  0.38,0.39,0.45,0.45,0.35]
daily_demand_summer_2 = list(map(sum, zip(heatpump, daily_demand_summer_2)))
y1 = daily_demand_winter_2 * 59
y2 = daily_demand_spring_2 * 92
y3 = daily_demand_summer_2 * 92
y4 = daily_demand_spring_2 * 91
y5 = daily_demand_winter_2 * 31
demand_2 = y1 + y2 + y3 + y4 + y5

#adapted demand scenario 2
heatpump = 24*[0.37]
daily_demand_spring_2_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.48, 0.5, 0.47,
↪  0.46,0.48,0.48,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_spring_2_2 = list(map(sum, zip(heatpump, daily_demand_spring_2_2)))
daily_demand_winter_2_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.55, 0.61, 0.58,
↪  0.55,0.55,0.56,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_winter_2_2 = list(map(sum, zip(heatpump, daily_demand_winter_2_2)))
daily_demand_summer_2_2 = [0.25,0.2,0.19,0.17,0.16,0.17,0.18,0.24,0.39, 0.38, 0.37,
↪  0.38,0.39,0.45,0.45,0.35,0.27,0.29,0.35,0.32,0.31,0.31,0.32,0.33]
daily_demand_summer_2_2 = list(map(sum, zip(heatpump, daily_demand_summer_2_2)))
y1_2 = daily_demand_winter_2_2 * 59
y2_2 = daily_demand_spring_2_2 * 92
y3_2 = daily_demand_summer_2_2 * 92
y4_2 = daily_demand_spring_2_2 * 91
```

```
y5_2 = daily_demand_winter_2_2 * 31
demand_2_2 = y1_2 + y2_2 + y3_2 + y4_2 + y5_2


#add all houses with demand scenario 2
total_demand_2_1 = [i*0.5*houses_2 for i in demand_2]
total_demand_2_2 = [i*0.5*houses_2 for i in demand_2_2]
total_demand_2 = list(map(sum, zip(total_demand_2_1, total_demand_2_2)))
number_of_solar_panels_2 = 16
supply_2 = list(np.array(supply_per_cm2) * size_solar_panel * number_of_solar_panels_2)
total_supply_2 = [i*houses_2 for i in supply_2]

#demand scenario 3
daily_demand_spring_3 =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.2, 1.45,
↪  1.5, 1.35,1.25,1.15,1.1,1.05]
daily_demand_winter_3 =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.3, 1.55,
↪  1.65, 1.5,1.3,1.2,1.1,1.05]
daily_demand_summer_3 =
↪  [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1,1.15, 1.15,
↪  1.15, 1.2,1.3,1.3,1.1,1.05]
y6 = daily_demand_winter_3 * 59
y7 = daily_demand_spring_3 * 92
y8 = daily_demand_summer_3 * 92
y9 = daily_demand_spring_3 * 91
y10 = daily_demand_winter_3 * 31
demand_3 = y6 + y7 + y8 + y9 + y10

#adapted demand scenario 3
daily_demand_spring_3_3 = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.2, 1.45, 1.5,
↪  1.35,1.25,1.15,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
daily_demand_winter_3_3 = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.3, 1.55, 1.65,
↪  1.5,1.3,1.2,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
daily_demand_summer_3_3 = [0.85,0.75,0.7,0.65,0.6,0.6,0.6,0.65,1.15, 1.15, 1.15,
↪  1.2,1.3,1.3,1.1,1.05,0.8,0.9,1,1.05,1.05,1.05,1.05,1.1]
y6_3 = daily_demand_winter_3_3 * 59
y7_3 = daily_demand_spring_3_3 * 92
y8_3 = daily_demand_summer_3_3 * 92
y9_3 = daily_demand_spring_3_3 * 91
y10_3 = daily_demand_winter_3_3 * 31
demand_3_3 = y6_3 + y7_3 + y8_3 + y9_3 + y10_3

#add all houses with demand scenario 3
total_demand_3 = [i*0.5*houses_3 for i in demand_3]
total_demand_3_2 = [i*0.5*houses_3 for i in demand_3_3]
total_demand_3 = list(map(sum, zip(total_demand_3_1, total_demand_3_2)))
number_of_solar_panels_3 = 23
supply_3 = list(np.array(supply_per_cm2) * size_solar_panel * number_of_solar_panels_3)
total_supply_3 = [i*houses_3 for i in supply_3]

#set total demand and supply of neighborhood
total_demand = list(map(sum, zip(total_demand_1,total_demand_2,total_demand_3)))
total_supply = list(map(sum, zip(total_supply_1,total_supply_2,total_supply_3)))
```

```
    #demand change of entire neighborhood
    for i in range(len(x)-1):
        if total_demand[i]>=total_supply[i]:                      #more demand than supply
            total_demand[i+1] +=0.44*(total_demand[i] - total_supply[i])
            total_demand[i] -= 0.44*(total_demand[i] - total_supply[i])
        elif total_demand[i]<total_supply[i]:                     #more supply than demand
            if 0.44*total_demand[i+1]<=(total_supply[i] - total_demand[i]):
                total_demand[i] += 0.44*total_demand[i+1]
                total_demand[i+1] -= 0.44*total_demand[i+1]
            else:
                total_demand[i+1] -= (total_supply[i] - total_demand[i])
                total_demand[i] += (total_supply[i] - total_demand[i])

    #realistic storage solution
    battery_storage = 0
    max_storage = 130*n
    battery=[]
    for i in range(8760):
        if total_supply[i]>=total_demand[i]:                      #more supply than demand
            surplus = total_supply[i]-total_demand[i]
            battery_storage+=surplus
            if battery_storage > max_storage:
                total_supply[i] -= (max_storage - (battery_storage-surplus))
                battery_storage = max_storage
                battery.append(battery_storage)
            else:
                total_supply[i] -= surplus
                battery.append(battery_storage)
        elif total_supply[i]<total_demand[i]:                      #more demand than supply
            if battery_storage <= total_demand[i] - total_supply[i]:
                total_supply[i] += battery_storage
                battery_storage = 0
                battery.append(battery_storage)
            else:
                battery_storage -= (total_demand[i] - total_supply[i])
                total_supply[i] = total_demand[i]
                battery.append(battery_storage)
    return total_demand,total_supply

nr_of_runs = 5000
total_neighborhood_demand = []
total_neighborhood_supply = []
for j in range(nr_of_runs):
    total_neighborhood_demand.append(generate_total_neighborhood_demand(30)[0])
    total_neighborhood_supply.append(generate_total_neighborhood_demand(30)[1])

for i in range(10):
    plt.plot(x,total_neighborhood_demand[i], label="_nolabel_", color="red", zorder=2)
    plt.plot(x,total_neighborhood_supply[i], label="_nolabel_", color="pink", zorder=1)
plt.plot([],[], label= "demand", color="red")
plt.plot([],[], label="supply", color="pink")
plt.xlabel('Date')
ticks = ["Jan", "Feb", "March", "April", "May", "June", "July", "August", "Sept", "Oct",
↪  "Nov", "Dec"]
plt.xticks(x, ticks, rotation=45)
plt.locator_params(axis='x', nbins=12)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a neighborhood of 30 houses over 2023')
plt.legend()
plt.show()

#average dependency
avg_shortage = []
avg_surplus=[]
for j in range(nr_of_runs):
    for i in range(8760):
        if total_neighborhood_demand[j][i]>=total_neighborhood_supply[j][i]:
            shortage = total_neighborhood_demand[j][i]-total_neighborhood_supply[j][i]
            avg_shortage.append(shortage)
        elif total_neighborhood_demand[j][i]<total_neighborhood_supply[j][i]:
            surplus = total_neighborhood_supply[j][i] - total_neighborhood_demand[j][i]
            avg_surplus.append(surplus)
average_shortage = sum(avg_shortage)/nr_of_runs
average_surplus = sum(avg_surplus)/nr_of_runs
average_dependency = average_shortage + average_surplus
print("The total avg energy shortage over 2023 equals", average_shortage, "kWh. While the
↪  surplus equals", average_surplus, ", which accounts to a dependency of",
↪  average_dependency, "kWh.")

#first week of feb only
for i in range(10):
    plt.plot(x[696:864],total_neighborhood_demand[i][696:864], label='_nolabel_',
     ↪  color="red", zorder=2)
    plt.plot(x[696:864],total_neighborhood_supply[i][696:864], label="_nolabel_",
     ↪  color="pink", zorder=1)
plt.plot([],[],label="demand", color="red")
plt.plot([],[],label="supply", color="pink")
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[696:864], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a neighborhood over first week of feb 2023')
plt.legend()
plt.show()

#average dependency
avg_shortage = []
avg_surplus=[]
for j in range(nr_of_runs):
    for i in range(696,864):
        if total_neighborhood_demand[j][i]>=total_neighborhood_supply[j][i]:
            shortage = total_neighborhood_demand[j][i]-total_neighborhood_supply[j][i]
            avg_shortage.append(shortage)
        elif total_neighborhood_demand[j][i]<total_neighborhood_supply[j][i]:
            surplus = total_neighborhood_supply[j][i] - total_neighborhood_demand[j][i]
            avg_surplus.append(surplus)
average_shortage = sum(avg_shortage)/nr_of_runs
average_surplus = sum(avg_surplus)/nr_of_runs
average_dependency = average_shortage + average_surplus
```

```
print("The total avg energy shortage over 2023 equals", average_shortage, "kWh. While the
↪  surplus equals", average_surplus, ", which accounts to a dependency of",
↪  average_dependency, "kWh.")


#last week of july only
for i in range(10):
    plt.plot(x[4896:5064],total_neighborhood_demand[i][4896:5064], label="_nolabel_",
     ↪  color="red", zorder=2)
    plt.plot(x[4896:5064],total_neighborhood_supply[i][4896:5064], label="_nolabel_",
     ↪  color="pink", zorder=1)
plt.plot([],[],label="demand", color="red")
plt.plot([],[],label="supply", color="pink")
plt.xlabel('Time')
ticks = ["Mon", "Tues", "Wed", "Thur", "Fr", "Sat", "Sun"]
plt.xticks(x[4896:5064], ticks, rotation=45)
plt.locator_params(axis='x', nbins=7)
plt.ylabel('Energy (kWh)')
plt.title('Energy supply vs demand for a neighborhood over last week of july 2023')
plt.legend()
plt.show()

#average dependency
avg_shortage = []
avg_surplus=[]
for j in range(nr_of_runs):
    for i in range(4896,5064):
        if total_neighborhood_demand[j][i]>=total_neighborhood_supply[j][i]:
            shortage = total_neighborhood_demand[j][i]-total_neighborhood_supply[j][i]
            avg_shortage.append(shortage)
        elif total_neighborhood_demand[j][i]<total_neighborhood_supply[j][i]:
            surplus = total_neighborhood_supply[j][i] - total_neighborhood_demand[j][i]
            avg_surplus.append(surplus)
average_shortage = sum(avg_shortage)/nr_of_runs
average_surplus = sum(avg_surplus)/nr_of_runs
average_dependency = average_shortage + average_surplus
print("The total avg energy shortage over 2023 equals", average_shortage, "kWh. While the
↪  surplus equals", average_surplus, ", which accounts to a dependency of",
↪  average_dependency, "kWh.")
```