**Greenlabel dashboard validation**

To validate our python code and subsequently the greenlabel dashboard we remade our code and main visual in power bi.

To get the right data to validate the code we first started with exporting the dataframe straight from the API call in our python code. Power bi had trouble with the format the API uses so we exported the raw tables from our python code and used this for the validation before any transformations and calculations are made.

Below you can see the API call along with the transformations and calculations we used afterwards for the greenlabel dashboard in our python code.

A screen shot of a computer program

Description automatically generated

A computer screen with text

Description automatically generated

The exported tables for both locations look like this when loaded into power bi.

A screenshot of a table

Description automatically generatedA screenshot of a data

Description automatically generated

The tables are currently grouped by every hour of the day which we will change to daily meaning we are going to aggregate the tables.

To aggregate the tables, we used the following M code in power bi.

A screenshot of a computer

Description automatically generated

This M code in power bi makes the same transformations as this part of the code in our python script.

A screen shot of a computer code

Description automatically generated

After running these transformations, we get the following table in python.

A screenshot of a computer

Description automatically generated

And after running this same M code script we get the following table in power bi.

A screenshot of a computer

Description automatically generated

As you can see the values in power bi align with the values in our python code which means that the transformation in python seems to work as intended.

After this first step, we continued with remaking the columns we made in our python script, but now in power bi. These columns include the following:

A computer screen with text

Description automatically generated

Here in our python script we made a few new columns such as Ew, Es and Total\_green\_energy. We are going to make these in power bi too and see if the outcome aligns with the outcome of the python script.

Here are the outcomes of the new column calculations in the python script.

A screenshot of a computer

Description automatically generated

And here are the outcomes of the new column calculations in power bi.

A screenshot of a computer

Description automatically generated

As you can see the values align between the python script and the power bi queries.

Now we have all the columns needed to validate our main visual. Because all the outcomes align between the programs we can assume that the dashboard code and calculations perform as expected.

The following visual is the main visual of our greenlabel dashboard. This visual is made with the use of the newly created columns in the code.

A graph of energy difference by date

Description automatically generated

This is the visual that we got out of power bi by creating the same transformations and calculations as we did in our python script. As you can see the visual shows the same values on the same dates. This provides us with the clarity that our python code is performing as expected and that power bi can be a good tool to validate dashboards like these.

A graph with green squares and yellow squares

Description automatically generated

**Invalid API call?**

During the validation process we came to the understanding that our API call was not formulated correctly.

Out API request called for sunshine\_duration and wind\_speed\_10m from the table hourly which provides hourly weather data.



As you can see below the API has two tables. One daily and one hourly table. The daily table gives us daily weather information and the hourly table gives us hourly weather information.

A screenshot of a weather variable

Description automatically generated

A screenshot of a computer

Description automatically generated

The API call we made requests the data from only the hourly table while the sunshine\_duration is only available in the daily table. The API call in our python script gave no warning or error regarding this discrepancy unfortunately, but we found this during the validation process.

By looking at the provided API json data we could look back at the open weather data from the API source and see that it graps the sunshine\_duration from the daily table instead of the requested hourly table. The API automatically transforms this column to fit in with the hourly table by dividing the values over each hour of the day.

If used in this format the data would not be correct, because the API wouldn’t know what hours should be what value. Fortunately for our dashboard we wanted to show daily changes in weather and not hourly. By getting the sum of all hours we could transform it back to daily again regardless of the invalid API call. The same would be done for the wind\_speed\_10m column. We transformed this column to show the average of a certain day by taking the average of all hours.

By researching this discrepancy we found out that the data that we get from the API is now valid as it performs a transformation that we later switch back anyways.

At the end of the validation process we manually looked at the raw data that we got in our python script from the API call and put it next to the data shown on the website where we call the API. The values from the website and our raw tables from the API in python aligned, which means that the data we get is correct even though we made an invalid API call by the looks of it.