Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

Área de Ingeniería Telemática

TRABAJO FIN DE GRADO Nº XXYYZZ

Monitorización de una planta fotovoltaica mediante un portal web

Pablo Fernández Pita

TUTOR: D. Manuel Arsenio Barbon Alvarez

TUTOR: D. Luis Froilán Bayón Arnau

FECHA: Junio de 2023

Contents

List of Figures

List of Tables

List of Code Fragments

Acronyms

AC alternating current. 7, 8

CSV comma-separated values. 12, 18, 19, 20, 21, 23, 24, 29, 30, 35, 38

DC direct current. 7, 8

EPI Escuela Politécnica de Ingeniería de Gijón. 12

IDE integrated development environment. 34

NaN Not a Number. 23, 24, 31

PV Photovoltaic. 1, 3, 6, 7, 9, 10, 16, 33, 35, 37, 38

UTC Coordinated Universal Time. 13, 29, 30

1. Introduction

In recent years, there has been a growing interest in renewable energy sources as a means of mitigating the effects of climate change and reducing reliance on fossil fuels. Among the most popular sources of renewable energy are solar Photovoltaic (PV) panels, which convert sunlight into electricity. As the cost of PV panels continues to decline, their adoption has become increasingly widespread, particularly in the residential and commercial sectors.

However, with this increased adoption comes the need for efficient monitoring and management of PV systems to ensure optimal performance and longevity. This is where the focus of this project lies - on the monitoring of PV panels. Through a series of sensors placed in our photovoltaic park, we can keep track of the efficiency of our PV panels and identify possible problems that could arise. In order to interact with this data, one of the most popular solutions is to access an app or a website dedicated to showing the current state, as well as the evolution of the data, gathered.

Some of the more popular providers of these services are SolarEdge [10], Fronius [1] and SMA Solar [2]. These companies do not only sell high-efficiency solar panels and inverters but also the tools to monitor them easily. As seen in figures 1.1 and 1.2, these services are mainly focused on delivering value to their customers. They are also keen on reassuring the decision they made when hiring their products by showing them clear metrics such as money saved on electricity or CO_2 not emitted to the atmosphere.

The monitoring will be conducted for academic purposes, specifically focusing on the power generation process and the efficiency of the panels. By understanding the complexities of PV technology, we can better appreciate the benefits of this technology and help to advance the adoption of renewable energy sources.

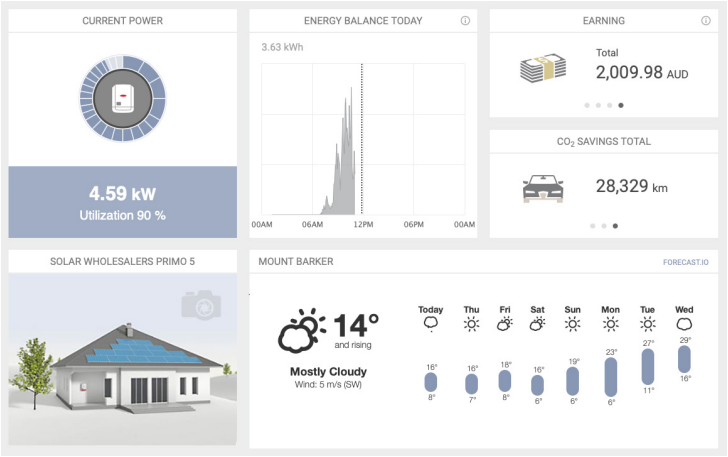


Figure 1.1.- Fronius monitoring service [3]

Figure 1.2.- Solar SMA monitoring service [4]

1.1.- Motivation of the project

I like data science and renewable energies.

1.2.- Project's goals

This final year project aims to build a web service to monitor the state of the solar panels installed in our college, specifically the ones located behind the west department building. This project is part of a bigger investigation, also present in Arganda del Rey, where more sensors are installed in another photovoltaic park. These two sites work together in an investigation performed by professors D. Luis Froilán Bayón Arnau and D. Manuel Arsenio Barbón Alvarez.

The requirements that the website must abide by are: to provide graphical and numerical information of the electrical energy generated and analyze the efficiency of the panels to evaluate the state of the PV panels and see if a problem has occurred. Also, this project must provide a system to access previously cleaned data in a format suitable to be analyzed with external mathematical tools such as Matlab or Mathematica.

To provide this service, our PV monitoring system must have an architecture designed to fulfil these requirements as best as possible. In the figure 1.3 we can see a schematic approach to this process and the steps the data goes through before it is consumed on the website.

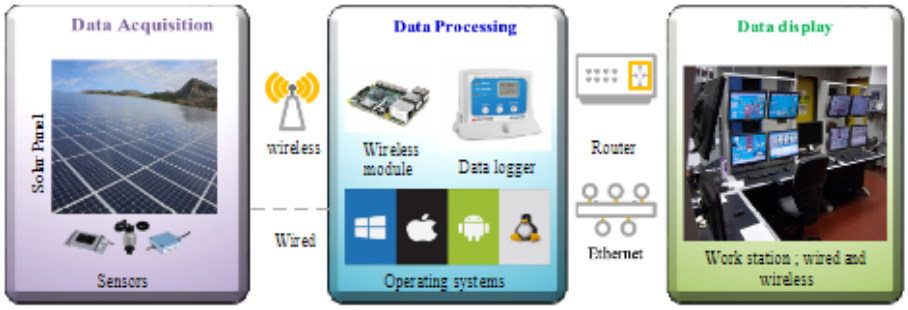
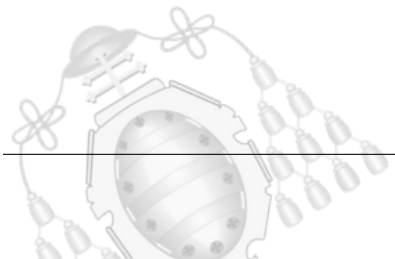


Figure 1.3.- A basic architecture of a solar PV monitoring system. Image obtained from [5]

1.3.- Document structure

Now that the goal and context of this project are set it is important to define how this document is going to be structured. Filler text until the document is finished.



2. Photovoltaic plants

In Spain, a country blessed with an abundance of sunshine, solar plants play a pivotal role in the pursuit of clean and sustainable energy generation. The vast potential of harnessing solar power in this region has spurred significant investments in solar infrastructure. With a commitment to reducing carbon emissions and diversifying its energy sources, Spain has emerged as a leading player in the solar energy sector. This chapter focuses on the working principle of solar plants, the various categories of solar plants along with their constituent components.

2.1.- Working principle

The working principle of a solar cell is based on the phenomenon of the photovoltaic effect, which enables the conversion of sunlight directly into electricity. At the heart of a solar cell lies a semiconductor material, typically silicon, with specific chemical properties that facilitate the process. When sunlight, composed of photons, strikes the surface of the solar cell, the photons transfer their energy to the atoms in the semiconductor. This energy absorption promotes the generation of electron-hole pairs, where electrons are excited from their original positions, leaving behind positively charged holes. The built-in electric field within the solar cell then guides the separated electrons and holes in opposite directions. By placing metal contacts on the top and bottom surfaces of the cell, an external circuit can be completed, allowing the flow of electrons as an electric current. This flow of electrons represents the desired output of the solar cell, which can be used to power various devices or stored for later use. **Figure 2.1** shows a simplified diagram of this process.

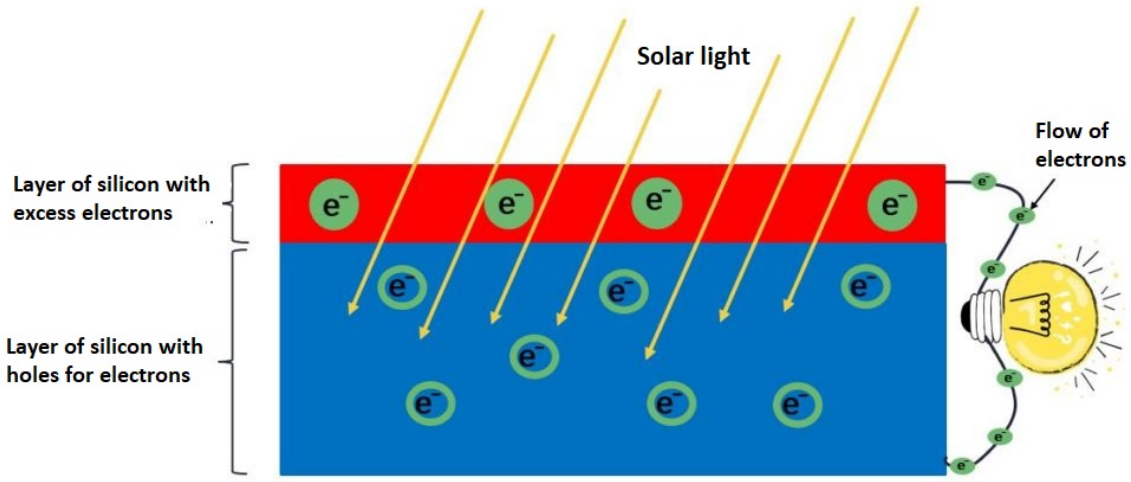


Figure 2.1.- Basic diagram of a solar cell. Image obtained from [6]

2.1.1.- Typical values of energy and power generated

Around 200W Efficiency of 20% Talk about how much we are generating right now with the current setup.

2.2.- Types of Solar Power Installations

Photovoltaic plants have emerged as crucial contributors to the renewable energy landscape, utilizing solar power to generate clean electricity. In this context, understanding the different types of PV plants is essential for optimizing their deployment and maximizing energy production. Two primary types of PV plants include utility-scale installations designed for large-scale electricity generation and distributed systems integrated into buildings for localized power generation. Each type offers unique advantages and contributes to the global transition towards a sustainable energy future.

2.2.1.- Utility-Scale Photovoltaic Plants

Utility-scale PV plants are designed for large-scale electricity generation. These plants typically consist of a vast array of PV modules installed over a considerable land area. The modules are then connected in series and parallel to form strings and arrays, enabling

higher voltage and power output. Utility-scale PV plants often employ tracking systems to maximize solar exposure throughout the day, enhancing energy production. These plants are integrated with power conditioning units, inverters, and transformers to convert the direct current (DC) power generated by the PV modules into alternating current (AC) power suitable for grid connection. Their extensive capacity and connection to the grid make utility-scale PV plants essential contributors to renewable energy generation, supporting regional power demands and reducing reliance on fossil fuels.

2.2.2.- Distributed generation

On the other hand, there are also smaller-scale PV plants designed for distributed generation. These systems, often referred to as rooftop or building-integrated PV plants, are installed on residential, commercial, or industrial rooftops. In this case, normally their orientation is fixed on a certain angle, calculated to maximize energy production throughout the year. They consist of a relatively smaller number of PV modules customized to fit the available roof space. Building-integrated PV plants offer several advantages, including the utilization of unused rooftop areas, reduced transmission losses, and proximity to energy consumers, minimizing grid infrastructure requirements.

These plants are typically grid-connected, allowing excess electricity to be fed back to the grid through net metering or utilized for local consumption. Distributed PV plants contribute to decentralizing the power generation landscape, promoting energy self-sufficiency, and reducing the carbon footprint at community level. Their modular design and adaptability make them suitable for various applications, ranging from individual residences to large commercial complexes.

2.2.3.- Off-Grid Installations

Another type of solar plant, usually found in remote locations and agricultural farms is off-grid installations. They operate independently and are placed to meet lighting demands, provide support for telecommunications, and power irrigation systems. **Figure 2.2** shows a diagram of an installation

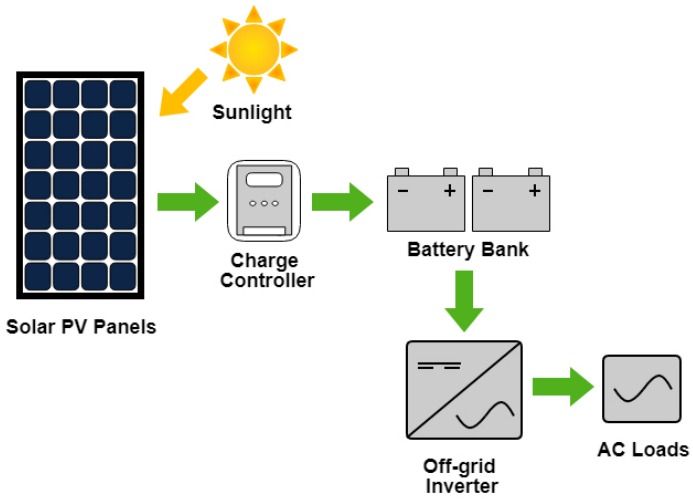


Figure 2.2.- Off-grid installation diagram. Image obtained from [7]

2.2.4.- Current working setup at EPI Gijón

In relation to the previous classification talk about how our installation is set up and maybe even the current impact on the electricity of the campus.

2.3.- Components of a solar plant

In order for a solar plant to work it needs to have the following components

- **Solar panel**: the most visible element of our system. It consists of a group of photovoltaic cells in charge of capturing the solar radiation and converting it into a DC current.
- **Inverter**: converts the DC electrical current produced by the cells into AC current, available for consumption. It may come with an **optimizer** to maximize the DC energy from the panel before sending it to the inverter
- **Transformer**: the alternating current generated by the inverters, is generally low voltage (300-800 V), for this reason, a transformer is used to elevate it to medium voltage (until 36kV) for transportation-

The solar plant will need two extra components to work properly in case of an off-grid set-up.

- **Battery**: to store the energy produced by the panels that is not consumed at the same time it is generated, the stored energy can then be used when needed, for example, at night.
- **Charge controller**: to protect the battery from overcharging and prevent inefficient use of the battery.

It would be ideal to add here photos of our own installation and of its components. Talk about the specific models used even protocols used. And add tables with relevant data and specifications

Figure 2.3.- A solar panel

2.4.- Relevant magnitudes to monitor

When monitoring a solar plant, several variables play a crucial role in assessing its performance and ensuring optimal operation. The first essential variable is solar irradiance, which represents the intensity of sunlight reaching the photovoltaic (PV) panels. By continuously measuring solar irradiance, operators can evaluate the available solar resource and identify any deviations from expected levels. This information is vital for determining the potential energy production and diagnosing any issues that may affect the system's efficiency.

Another critical variable is the temperature of the PV panels. Solar panels are sensitive to temperature changes, and the excessive heat can reduce their performance.

Monitoring panel temperature allows operators to assess the impact of temperature on the electrical output and efficiency of the system. By tracking panel temperature in real-time, it becomes possible to implement cooling measures or adjust operating parameters to mitigate the negative effects of elevated temperatures.

Additionally, the electrical output of the solar plant is a fundamental variable to monitor. This includes measuring the current and voltage generated by the PV modules, as well as the power output of the entire system. By monitoring electrical parameters, operators can evaluate the overall performance of the solar plant, detect any abnormalities or deviations from expected values, and promptly address any issues that may arise.

Furthermore, monitoring the environmental conditions surrounding the solar plant is essential. Factors such as ambient temperature, humidity, and wind speed can impact the performance and efficiency of the system. Understanding the relationship between these environmental variables and the solar plant's operation allows for improved system management, such as optimizing energy generation during favourable weather conditions or implementing protective measures during extreme weather events.

As demonstrated by Hussein A Kazem and Miqdam T Chaichan, in their paper "Effect of Humidity on Photovoltaic Performance Based on Experimental Study" [8] higher humidity decreases solar panels' efficiency. The conclusion of this study states that relative humidity has an inverse strong correlation with current, voltage and power. In the case of power, the correlation factor R was -0.98395.

Figure 2.4 shows the results of their experimental investigation. This decrease in power due to higher humidity could be attributed to small water droplets, as well as water vapour, with the potential to accumulate on the surface of solar panels. Consequently, they can deflect or alter the path of sunlight, diminishing its direct impact on the solar cells. As a result, the reduced sunlight exposure leads to a decrease in electricity generation. This would mean that it can also be wise to measure the dew point, which represents the temperature at which air becomes saturated and condensation occurs, as part of the monitoring process.

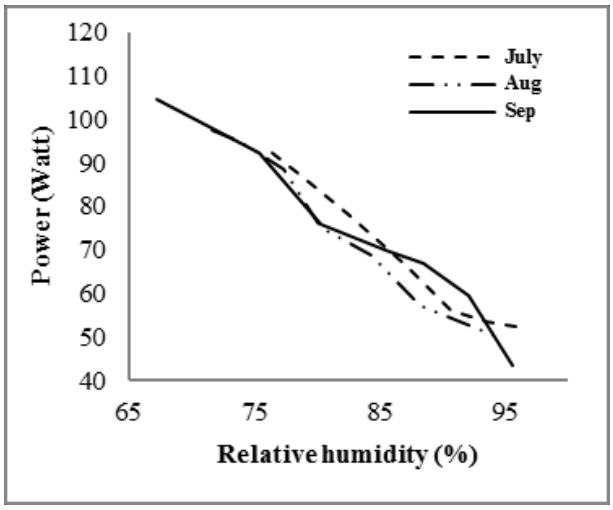


Figure 2.4.- Relationship between relative humidity and power output. Image obtained from [8]

In conclusion, the variables that are most important to monitor in a solar plant include solar irradiance, panel temperature, electrical output, and environmental conditions. By continuously assessing these variables, operators can gain valuable insights into the system's performance, identify potential issues, and take proactive measures to optimize energy production and ensure the long-term reliability of the solar plant.

3. Data lifecycle

In order to obtain valuable insights from our data, it must first go through a process where the data is properly extracted, revised, and stored. This process was performed with two scripts written in Python, making use of two of its most commonly used libraries in the field of data science, Pandas and NumPy. One for the data coming from the dataloggers and another for the records generated by the inverter's software.

3.1.- Data acquisition

The first stage in any data life-cycle is to acquire the data itself. This process is called data acquisition and it essentially encompasses the gathering of information to comprehend an electrical or physical phenomenon by utilizing sensors, measurement devices, and an electronic device capable of registering that information.

In this study's case, two dataloggers are available on-site that perform that operation. One dedicated to the variables measured on the terrace, and the other to those located on the garden. Both dataloggers are located rather closely, as they are both inside the fourth module of the West department building of the Escuela Politécnica de Ingeniería de Gijón (EPI).

Both dataloggers are from manufacturer Kipp&Zonnen, more specifically model LogboxSE [9], shown in **figure 3.1**. **Table 3.1** shows the characteristics of these devices. The sampling rate is set to one minute, and, as it is configured to create one comma-separated values (CSV) file per day, that means it should produce 1440 values per day. Nevertheless, as it is a non-ideal world, where errors occur on a daily basis, our system must be ready to deal with corrupt, incomplete, or incorrect data.

The sensors communicate with the datalogger vía **xxxxxxxxx I would like to add here some facts about the sensors used, why there are two dataloggers and**

what is the logic behind it. Why the manual gathering of the files?? How the datalogger is connected to the sensors??

An important aspect when dealing with time-referenced logs is to know which convention it is using regarding time zones and daylight saving time. In the case of our study, both dataloggers work with Coordinated Universal Time (UTC) times. This practice is widely adopted when working with solar measurements due to its consistency throughout the year. Using UTC simplifies the handling of dates affected by daylight saving time adjustments, ensuring a more straightforward and reliable analysis of the data.

On the other hand, the software used by our string inverter (SolarEdge [10]) provides the timestamp related to the local time of the installation. This poses a problem, as one of the key elements of this study is the comparison of the irradiances provided by the sensors through the datalogger and the energy generated measured by the string inverter. In **section 3.3.1.1** the process of synchronizing these two sources will be explained in detail.

Figure 3.1.- LOGBOX SE DataLogger

Table 3.1.- Specifications on LogboxSE

Variable	Description
Analogue inputs	4 differential 24-bit and 4 single-ended 12-bit
Input ranges	19 mV to 2.5 V differential, 2 x 2.5 V and 2 x 3 V single-ended
Accuracy	0.05 % for 24-bit, 0.1 % for 12-bit
Digital inputs	4, maximum input 15 V, logic levels 3 x 3 V and 1 x 0.5 V
Serial input	RS-485 port for up to 8x compatible Modbus® devices from Kipp&Zonen, Lufft or IMT
Supply voltage DC	6 x AA internal batteries or external supply 4 to 24 V
Power consumption	1 mA standby, 7 mA typical during measurement, 50 mA with modem on
Memory card type	SD card (512 MB included)
Communication	RS-232, USB, quad-band GSM / GPRS modem with external antenna

The inverter 3.2 is in charge of measuring the energy generated each hour and the power. It is manufactured by **xxxxxx** and gives options regarding **xxxxx**

Figure 3.2.- Inverter

Table 3.2.- Specifications of Inveter

Variable	Description
Analogue inputs	4 differential 24-bit and 4 single-ended 12-bit
Input ranges	19 mV to 2.5 V differential, 2 x 2.5 V and 2 x 3 V single-ended
Accuracy	0.05 % for 24-bit, 0.1 % for 12-bit
Digital inputs	4, maximum input 15 V, logic levels 3 x 3 V and 1 x 0.5 V
Serial input	RS-485 port for up to 8x compatible Modbus® devices from Kipp&Zonen, Lufft or IMT
Communication	RS-232, USB, quad-band GSM / GPRS modem with external antenna

3.1.1.- Measured variables

As explained in **section 2.4** there are many variables that can be useful to monitor a solar plant's performance. At the moment, there are 13 and 19 variables being monitored on a daily basis on each datalogger. These variables are described in **tables 3.3** and **3.4**.

Table 3.3.- Variables measured on Datalogger 1

Parameter	Variable name	Units
Global irradiance over horizontal surface	VM4T_E_Irrad.Global_Avg	W/m^2
Diffuse irradiance over horizontal surface	VM4T_E_Irrad.Difusa_Avg	W/m^2
Ambient temperature	VM4T_E_T.Amb_Avg	ºC
Relative humidity	VM4T_E_Hum.Rel_Avg	%
Dew point	VM4T_E_P.Rocio_Avg	ºC
Wind speed	VM4T_E_Vel.Viento_Avg	m/s
Wind direction	VM4T_E_Dir.Viento_Avg	º dir. North
Atmospheric pressure	VM4T_E_P.atm_Avg	hPa/mBar
Precipitation	VM4T_E_Precip.int_Avg	mm
Precipitation per hour	VM4T_E_Precip_Avg	mm

Table 3.4.- Variables measured on Datalogger 2

Parameter	Variable name	Units
Reflected irradiance over horizontal surface	VM4J_E_Irrad.Reflejada_Avg	W/m^2
Total irradiance over Tracker 1, frontal part	VM4J_T1_D_Irrad.RT1_Avg	W/m^2
Total irradiance over Tracker 1, back part	VM4J_T1_T_Irrad.RT1_Avg	W/m^2
Temperature Tracker 1, frontal part	VM4J_T1_D_Temp.RT1_Avg	ºC
Temperature Tracker 1, back part	VM4J_T1_T_Temp.RT1_Avg	ºC
Angle of inclination of PV module over tracker	VM4J_T1_Inclinometro_Avg	º
Total irradiance of polar system at half latitude inclination	VM4J_SP_IML_Irrad.RT1_Avg	W/m^2
Total irradiance of polar system at latitude inclination	VM4J_SP_IL_Irrad.RT1_Avg	W/m^2
Temperature of polar system, frontal part	VM4J_SP_D_Temp.RT1_Avg	ºC
Temperature of solar system, back part	VM4J_SP_T_Temp.RT1_Avg	ºC
Temperature of table 1, over frontal part of the PV module	VM4J_M1_D_Temp.RT1_Avg	ºC
¿Temperature of table 1, over frontal part of the PV module?	VM4J_M1_TI_Temp.RT1_Avg	ºC
¿Temperature of table 1, over frontal part of the PV module?	VM4J_M1_TS_Temp.RT1_Avg	ºC
Total irradiance of table 1, frontal part of the module, with optimal yearly angle proposed by IDAE	VM4J_M1_D_IDAE_Irrad.RT1_Avg	W/m^2

Continued on next page

Table 3.4 – continued from previous page

Parameter	Variable name	Units
Total irradiance of table 1, frontal part of the module, with optimal yearly angle (Bayón Method)	VM4J_M1_D_A_Irrad.RT1_Avg	W/m^2
Total irradiance of table 1, frontal part of the module, with optimal angle proposed by Lorenzo	VM4J_M1_D_L_Irrad.RT1_Avg	W/m^2
Total irradiance of table 1, frontal part of the module, with optimal angle proposed by Jacobson	VM4J_M1_D_J_Irrad.RT1_Avg	W/m^2
Number of 2.5 µm particles in the air	PM2.5_Avg	
Number of 10 µm particles in the air	PM10_Avg	

Here I would like to talk about the data from the inverter. I still don´t know how to use it or what to do with it. It is very irregular, how it is generated? Every 15 minutes? Energy and power from each of the 8 PV modules + the total of the 8 together. Where this data is going to be stored and how can I process it.

Table 3.5.- Variables measured by EdgeSolar Software

Parameter	Variable name	Units
Energy panel 1 (Tracker 1 PV1 module)	VM4SC_IT1_PV1_Ener.P1.1.1	Wh
Power panel 1 (Tracker 1 PV1 module)	VM4SC_IT1_PV1_Pote.P1.1.1	W
Energy panel 2 (Tracker 1 PV2 module)	VM4SC_IT1_PV2_Ener.P1.1.2	Wh
Power panel 2 (Tracker 1 PV2 module)	VM4SC_IT1_PV2_Pote.P1.1.2	W
Energy panel 3 (Tracker 1 PV3 module)	VM4SC_IT1_PV1_Ener.P1.1.3	Wh
Power panel 3 (Tracker 1 PV3 module)	VM4SC_IT1_PV1_Pote.P1.1.3	W

Continued on next page

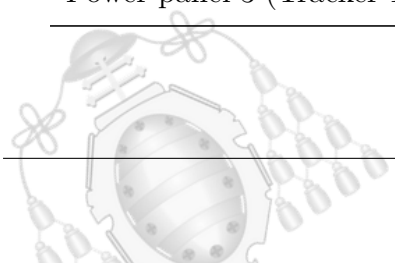


Table 3.5 – continued from previous page

Parameter	Variable name	Units
Energy panel 4 (Tracker 1 PV4 module)	VM4SC_IT1_PV1_Ener.P1.1.4	Wh
Power panel 4 (Tracker 1 PV4 module)	VM4SC_IT1_PV1_Pote.P1.1.4	W
Energy panel 5 (Tracker 1 PV5 module)	VM4SC_IT1_PV1_Ener.P1.1.5	Wh
Power panel 5 (Tracker 1 PV5 module)	VM4SC_IT1_PV1_Pote.P1.1.5	W
Energy panel 6 (Tracker 1 PV6 module)	VM4SC_IT1_PV1_Ener.P1.1.6	Wh
Power panel 6 (Tracker 1 PV6 module)	VM4SC_IT1_PV1_Pote.P1.1.6	W
Energy panel 7 (Tracker 1 PV7 module)	VM4SC_IT1_PV1_Ener.P1.1.7	Wh
Power panel 7 (Tracker 1 PV7 module)	VM4SC_IT1_PV1_Pote.P1.1.7	W
Energy panel 8 (Tracker 1 PV8 module)	VM4SC_IT1_PV1_Ener.P1.1.8	Wh
Power panel 8 (Tracker 1 PV8 module)	VM4SC_IT1_PV1_Pote.P1.1.8	W
Energy panel of the 8-module set (Tracker 1 String)	VM4SC_IT1_STR_Ener.Str.1.1	Wh
Power panel of the 8-module set (Tracker 1 String)	VM4SC_IT1_STR_Pote.Str.1.1	W

3.2.- Data cleaning

As mentioned earlier in the chapter, the less-than-optimal conditions of our environment necessitate the design of a structure capable of accommodating measurement failures. In order to standardize these raw files generated by the datalogger, a program in Python was developed, bearing in mind the necessities of the requirements.

When dealing with raw data there are three main concerns:

- **Missing values**: when some records are missing, in our case, minutes or even hours where no new rows were added to the CSV file.
- **Incorrect measurements**: due to the precision of the sensors or some external factor, wrong or impossible measurements might be added to a certain row.
- **Corrupt data**; the file itself is corrupt due to a malfunction and cannot be opened.

The process of data cleaning in this project has a very specific goal, add the 1440 records expected each day to the data repository making sure no incorrect measurements are appended. In the **sections 3.2.1** and **3.2.2** the process is expressed step by set. Also as it is going to be explained in s**ection 3.3**, once the previous goal is achieved, incorrect fixable measurements will get modified.

3.2.1.- File examination

To start off, the program must receive a file to process. The script is designed to check repeatedly on a folder on the lab computer it is running on for new raw CSV files. As shown in code fragment **3.1**, the program is always inside an infinite loop that sleeps for ten seconds if no file is found in the input directory. Once it finds a new file it will try and open it to check if it is a valid file and extract its header. The header must be read to differentiate which datalogger produced that specific log. The name the dataloggers give to the file is set to "KLOG" + a number. This numerical value represents the cumulative count of logs generated by the datalogger since its initial operation. Due to that particular circumstance, the filename alone is not enough to differentiate which datalogger the file comes from, therefore not allowing to later check whether that day's data has been parsed already.

Code fragment 3.1.- Infinite loop to check for CSV files

```
while True:
for file in os.listdir(PATH_INPUT_FILES):
    if file.lower().endswith('.csv'):
        file_to_process = os.path.join(PATH_INPUT_FILES, file)
        main(file_to_process)
time.sleep(10)
```

With that header and filename, a function in charge of checking if the file has already been parsed is called. The idea behind this is to reduce human error while extracting the raw files from the datalogger and copying them onto the computer. Thanks to the header, it is possible to know which datalogger created that file, and so with this information check

the corresponding log of already parsed files to see if that particular CSV file has been processed already by the system.

It is important to note that, if the user wants to re-process a particular file, the program also gives him the chance to do so by asking via the terminal if he is willingly re-adding a file to the system to modify some previous data.

In case a human error was made, and the user does not want to re-process that file, the program deletes that original file from the input folder and continues to check if any more CSV files are available in that folder. If on the other hand, the file is new, or the user wants to process it again, the program continues its execution.

Code fragment 3.2.- Function to ask the user if he wants to re-process a file

```
def ask_to_continue(filename_original):
    user_input = input(f"The file {filename_original} has already been parsed. "
                       f"Do you wish to continue and rewrite the data? [y/n] ")
    while user_input not in ["y", "n"]:
        print("Invalid input. Please enter 'y' or 'n' ")
        user_input = input(f"The file {filename_original} has already been parsed. "
                           f"Do you wish to continue and rewrite the data? [y/n] ")
    if user_input == "y":
        return True
    else:
        return False
```

The next step in the operation is to format the first two lines to leave just the relevant information and rename the file for better human understanding. With the extracted header from the previous step and the original filename, the file is copied into a folder where all the processed files are stored. The new name of this file for storage will be the addition of its original name and “Data”, if it comes from the second datalogger or “DataEstaci” if it comes from the first one. The name of this folder is **“RawDataParsed”**.

The following verification to be made is if the file has any records, or is it just composed of the variable’s names. If this file is empty, an error message is returned and it goes back

to the looped execution to check for new files. In case the file is, in fact empty, it is simply deleted from the input folder and an error is returned.

3.2.2.- Cleaning the archives

At this point, it has been verified that the file is uncorrupted and has some records. It is now time to start going through the file to see if all the measures have been taken and recorded correctly.

The first verification to be made is the number of records. As both dataloggers produce a log from each sensor every minute, and it generates one file every day, a correct file must have 1440 rows. Function **3.3** shows how this operation is performed. While loading the CSV file passed as an argument to a pandas data-frame, an extra check is performed. Due to a malfunction of the device, it is possible that a row has more columns than expected. When this happens, these lines must be skipped to have a consistent row size.

The function shown in **3.3** has two possible outcomes. If everything is correct and the file has 1440 rows, it is copied onto the folder **"TratarDatos"**. This folder stores the files, without any automatic changes made to them. In the opposite case, when there are less than 1440 records, the file is copied onto **"RevisarDatos"**. Inside this folder, the incomplete files can be stored to manually check what errors occurred and perhaps try and find a solution.

Code fragment 3.3.- Function to check if a file has 1440 records

```
import shutil
import pandas as pd

def check_1440_records(file_path, output_ok_directory, output_error_directory):
    df = pd.read_csv(file_path, on_bad_lines='skip')
    if len(df) == 1440:
        file_name = os.path.basename(file_path)
        output_path = os.path.join(output_ok_directory, file_name)
        shutil.copyfile(file_path, output_path)
        print(f"[OK] The file {file_name} has 1440 records.")
```

```
        return True
    else:
        file_name = os.path.basename(file_path)
        output_path = os.path.join(output_error_directory, file_name)
        shutil.copyfile(file_path, output_path)
        print(f"[ERROR] The file {file_name} has some missing records")
        return False
```

The Boolean value returned by function **3.3** is evaluated afterwards on the main function as now it must decide the succeeding action to be taken. Assuming everything was OK and the function returned true, a function called "datetime_together(path, name)" **(3.4)** is invoked. This method takes as arguments the path of the file stored in folder **"RawDataParsed"** and a new filename. This new filename is the result of the addition of the date of the records inside and "DataEstaci" or "Data" following the same criteria as previously.

Code fragment 3.4.- Function to join columns Date and Time

```
def datetime_together(path_input_data, new_file_name):
    df = pd.read_csv(path_input_data)
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], dayfirst=True)

    # Format the Datetime column
    df['Datetime'] = df['Datetime'].dt.strftime('%Y-%m-%d %H:%M:%S')

    df.set_index('Datetime', inplace=True)
    # Drop 'Date' and 'Time' columns
    df = df.drop(['Date', 'Time'], axis=1)

    df.reset_index(inplace=True)
    df.replace('---', 0.000, inplace=True) # Replace non taken measures to 0.
    path_treat_data = os.path.join(PATH_TREAT_DATA, new_file_name)
    df.to_csv(path_treat_data, index=False)
```

Once all these modifications are done, the program writes a new file into **"TratarDatos"** and deletes the previous file in this folder with columns "Date" and "Time" separated.

On the contrary, if there were less than 1440 records present on the file, the method called is "revision(path, name)". The file that this function reads and loads into a data frame comes from the previously mentioned folder **"RevisarDatos"**. The goal this subroutine aims to achieve is to output a file with 1440 records without any external human intervention. Another important requirement is that the least amount of valid information should be discarded. Finally, ideally, avoid generating new data that might provide wrong information due to it being estimated without good enough reference. To fulfil this three use cases were defined:

- **Missing intermediate data**: when the datalogger or the sensors had a temporary malfunction that resulted in some minutes or even hours not recording any data.
- **Missing data until one point**: the datalogger has not been working properly from the previous day and therefore is missing data from the start of the day until the problem got fixed.
- **Missing data from one point onwards**: the datalogger stopped working at a specific moment in the day of the file that is being parsed. No more rows from that point.

The three use cases are non-mutually exclusive and the code in charge of taking care of this problem should be able to deal with more than one at the same time. It is because of this that the same function will be in charge of checking if the goal of the method has been achieved. This way no matter the circumstances the output will be consistent. The procedure to solve each of those use cases is the following:

1. Once the CSV file is loaded into a data frame, all the rows but the first one get selected. These rows now get re-sampled with a frequency of one minute to ensure that all missing instances of the day have a record. Then these re-sampled rows get concatenated with the initial measurement. After that, all rows but those of type *"datetime64"* get cast to floats. When this is done, a linear interpolation [15] is performed to fill the Not a Number (NaN) values on the newly created rows.
2. Afterwards, the NaN remaining; those from columns where a sensor did not measure in all day due to a failure, get filled with zeroes. And only then do all values in the

data frame get rounded to three decimal places, the same as the precision of the sensors.

3. With the first use case solved, our focus shifts to the other two. When the values missing are from the start or the end of the data frame, re-sampling will not create the rows missing. It is because of that, the first time and the last time of the day must be set manually.

4. By creating a new data frame with a value per minute from the start time to the end time, an empty data frame with 1440 rows is obtained. By merging this data frame with the original one, a complete data frame is formed.

5. In this case the values that remain NaN are not replaced by zeroes because that would mean estimating values at times when no valid reference to do so exists.

6. Finally, this data frame with 1440 records is saved in a new CSV file on "TratarDatos".

Code fragment 3.5.- Automatic revision of missing rows

```
def revision(path_raw_data, new_file_name):
    df = pd.read_csv(path_raw_data, on_bad_lines='skip').dropna()

    # Format the Datetime column
    df['Datetime'] = pd.to_datetime(df['Date'] + ' ' + df['Time'], dayfirst=True)
    df.set_index('Datetime', inplace=True)

    first_row = df.head(1) # First row does not follow the pattern
    remaining_rows = df.iloc[1:]
    # Resample the remaining rows using a time-frequency of 1 minute
    resampled_rows = remaining_rows.resample('1T').asfreq()

    # Concatenate the first row with the resampled rows
    new_df = pd.concat([first_row, resampled_rows])
    # Drop 'Date' and 'Time' columns
    new_df = new_df.drop(['Date', 'Time'], axis=1)
    new_df.replace('---', 0.000, inplace=True) # Replace non taken measures to 0.

    cols_to_cast = new_df.select_dtypes(exclude=['datetime64']).columns
```

```
new_df[cols_to_cast] = new_df[cols_to_cast].astype(float)
# Interpolate missing values
new_df = new_df.interpolate(method='linear')
new_df.fillna(0.0, inplace=True) # Cleaning empty cells
new_df = new_df.round(decimals=3) # Round to 3 decimal places
# Reset index to Datetime
new_df.reset_index(inplace=True)

# Missing the first or last few measures
if len(new_df) != 1440:
    # Get the start and end time of the day
    start_time = new_df['Datetime'].min().replace(hour=0, minute=0, second=0,
        microsecond=0)
    end_time = new_df['Datetime'].max().replace(hour=23, minute=59, second=0,
        microsecond=0)

    # Create a new data frame with one row per minute of the day
    index = pd.date_range(start=start_time, end=end_time, freq='T')
    empty_df = pd.DataFrame(index=index)

    # Merge the new DataFrame with the original DataFrame
    new_df = pd.merge(empty_df, new_df, left_index=True, right_on='Datetime',
        how='outer')

    new_df.set_index('Datetime', inplace=True)
    # Reset index to Datetime
    new_df.reset_index(inplace=True)

# Save the output DataFrame to a CSV file
path_treat_data = os.path.join(PATH_TREAT_DATA, new_file_name)
new_df.to_csv(path_treat_data, index=False)
```

3.3.- Treating the data

Thus far the data cleaning of the input files has primarily focused on missing or unusable data. This is a very important process in data cleaning, but it would not be wise

to stop there. Thanks to powerful Python libraries such as Pandas [19] or Suntime [20], there are still many ways in which we can depurate the input files before concatenating their content into the data repository file.

To avoid errors adding up while calculating energy generated, before sunrise and after sunset, irradiance values must be set to zero. Some sensors may still give very small values during these hours, however, this data is useless as no power is actually being generated. Another possible problem that the data may have is the fact that at night, irradiance sensors can give negative results. This situation is physically impossible, making it illogical to store such values. If this were to happen, the appropriate action would be to set the corresponding value to zero.

Code fragments **3.6** and **3.7** show how all of the operations previously mentioned are performed. Impossible measures and sunrise and sunset time.

Code fragment 3.6.- Function treat the data

```
def treating_data(filename):
    basefilename = os.path.basename(filename)
    df = pd.read_csv(filename)
    # print(df.dtypes)
    # Irradiance measurements columns
    irradiance_cols = [col for col in df.columns if 'Irrad' in col]
    df.replace('---', 0.0, inplace=True) # Replace non taken measures to 0.
    df[irradiance_cols] = df[irradiance_cols].apply(pd.to_numeric)

    # Set negative values in irradiance_cols to 0
    for col in df.columns:
        if col in irradiance_cols:
            df[col] = df[col].apply(lambda x: 0 if x < 0 else x)

    # Convert 'DateTime' column to datetime data type
    df['Datetime'] = pd.to_datetime(df['Datetime'])
    # Get the date (day) of the first value
    date = df['Datetime'].dt.date.iloc[0]
    sunrise_t, sunset_t = get_sunrise_sunset(date)
```

```
    # Set irradiance_cols values to 0 before sunrise_t and after sunset_t
    for col in irradiance_cols:
        df.loc[df['Datetime'] < sunrise_t, col] = 0
        df.loc[df['Datetime'] > sunset_t, col] = 0

    filename1440 = basefilename.split('Data')[0] + 'Data1440' +
        basefilename.split('Data')[1]
    output = os.path.join(PATH_DATA_GIJON_1440, filename1440)
    df.to_csv(output, index=False)

    return output
```

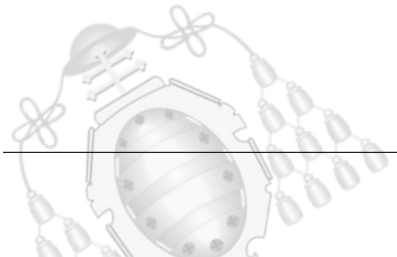
Code fragment 3.7.- Function to get the sunrise and sunset time at Gijón

```
from suntime import Sun, SunTimeException

def get_sunrise_sunset(date):
    # Coordinates of Gijon
    latitude = 43.5359
    longitude = -5.6619
    sun = Sun(latitude, longitude)
    try:
        # Get the sunrise and sunset times for the given date
        sunrise = sun.get_sunrise_time(date)
        sunset = sun.get_sunset_time(date)

        # Format the sunrise and sunset times
        sunrise_time = sunrise.strftime("%Y-%m-%d %H:%M:%S")
        sunset_time = sunset.strftime("%Y-%m-%d %H:%M:%S")
        return sunrise_time, sunset_time

    except SunTimeException:
        return None, None
```



3.3.1.- Re-sampling for twenty-four measurements

Even though the files with 1440 records are finally parsed, and ready to be added to our data source, one more process is going to be done with the document before moving on to the next one. We are going to transform our measures from one per minute to one per hour. This will result in a file with 24 records and a few modified variable names and magnitudes

As explained in **section 2.4** probably the most relevant variables recorded on both dataloggers are those regarding solar irradiance. Measurements stored on variables with the word "Irrad" on them are representing the average value of the solar irradiance over that minute. The measurement is expressed in units of Watts per square meter, as demonstrated in the tables labeled as **Table 3.3** and **3.4**.

Solar irradiance is often integrated over a period of time, for example, an hour, to calculate solar irradiation. This magnitude represents the solar energy generated in a given period. It is usually measured in Wh/m^2 or J/m^2. This variable is very useful, as it can be compared with the real energy generated by the inverter once the efficiencies of all the components are considered.

When re-sampling our data set from one per minute to one per hour, we are in fact calculating the energy generated during that period of time using the rectangle rule **(3.1)**. Because the measurements are taken every minute, the calculation of the energy generated during that hour can be done the same way you would calculate the mean value. For the rest of the variables that are being re-sampled, we would also calculate their mean value normally to obtain a relevant measure for the hour. Therefore, as shown in function **3.8**, the new data frame is filled with the mean values of each hour.

$$\int_a^b f(x)\,dx \approx (b-a)\cdot f\left(\frac{a+b}{2}\right) \tag{3.1}$$

3.3.1.1.- Time Zone Synchronization

Synchronizing timezone measurements between comparable rows on two different CSV files is of utmost importance when conducting data analysis or comparisons. In the realm of data processing, especially when dealing with multiple data sources or systems, ensuring that the timestamps align correctly is crucial for accurate and meaningful results. Timezone discrepancies can lead to inconsistencies and erroneous conclusions, potentially impacting the reliability and validity of the analysis.

As it has been stated previously in **section 3.1**, the dataloggers mark each measurement with the current UTC time, but the inverter does so with the local time, in this case, Spanish time. In consequence, one of the measurements must be translated into the other's time zone.

To simplify the posterior analysis, it was decided that the data from the sensors will get shifted to local time. Either option had its pros and cons, but both posed a similar difficulty we opted to have all measurements in local time so it was more intuitive for following monitoring. Code fragment **3.8** shows the function in charge of reading the file with a record per minute in UTC time and outputting a new file using the local time as reference and with the energies calculated as expressed in **3.3.1**.

It is important to also note that a similar function to **3.8** is called just before, and it also re-samples the log file with 1440 records but then outputs a file with the UTC time.

Code fragment 3.8.- Function to get 24 records per day on local time

```
def resample_24HLocal(path_file_1440):
    df = pd.read_csv(path_file_1440)
    # Convert 'Datetime' column to datetime type
    df['Datetime'] = pd.to_datetime(df['Datetime'])
    # Set 'Datetime' column as the index
    df.set_index('Datetime', inplace=True)

    # Set index to UTC timezone
    df = df.tz_localize('utc')
    # Shift time to Madrid
```

```
df = df.tz_convert('Europe/Madrid')

# Resample to 1 record per hour and calculate the mean of each hour
df_resampled = df.resample('H').mean()
df_resampled = df_resampled.round(decimals=3)

# Reset the index to have 'Datetime' as a column again
df_resampled.reset_index(inplace=True)

# Remove the timezone from the column
df_resampled['Datetime'] = df_resampled['Datetime'].dt.tz_localize(None)

basefilename = os.path.basename(path_file_1440)
filename24 = basefilename.split('Data')[0] + 'Data24Local' +
    basefilename.split('Data')[1]
output = os.path.join(PATH_DATA_GIJON_24HORALOCAL, filename24)
# Save the resampled data to a new CSV file
df_resampled.to_csv(output, index=False)
return output
```

3.4.- Storing the data

The file is now ready to be stored in its permanent location where it will be consumed by our web application. But first, it must go through a final step.

In order to access the data and also to maintain all records in the same file, the parsed file now will be appended to the file used as a repository for all records. As we have two different data sources and two different time interval measurements, there will be four different CSV files in charge of acting as the data source for the web application. Two for the data generated by the first datalogger, one with data on UTC time and a log per minute, and another with local time and a log per hour. In the case of the second datalogger, the same logic is followed. These files will be called **“Datalogger1.csv”**, **“Datalogger24hLocalEstaci.csv”**, **“Datalogger2.csv”** and **“Datalogger24hLocal.csv”**.

The idea behind having two different repositories with repeated data, as the 24-record files are just a partial view of the original 1440 entries per day is to allow for analysis from two different perspectives. If the objective is to analyze the potential of the solar installation, the more in-depth records per minute allow for a deeper study. However if, as it is more likely, the goal is to measure the efficiency of the cells and ensure the correct maintenance of all the components, the more useful insights will come from the comparison between the measured theoretical energy and the obtained energy directly acquired from the string inverter.

Because the variables measured each day may differ, the automatic appending of the records cannot be done carelessly. As shown in code fragment **3.9** a number of considerations are made before concatenating the new records. Three cases can be distinguished depending on the nature of the new data about to be added:

- New record with same or fewer variables: the simplest case possible. The new rows will simply get concatenated at the end of the file and every variable-value pair has a column to put these data in. Those columns where no measurement was taken will be left as a NaN.
- New record with more variables: if a new measure that previously was not being monitored is present on the file, the column must be created and filled with NaN values for the previous records that did not include it.
- Old record being re-processed: a file that is purposefully being reprocessed is going to be added to the data source where records of that specific day and time are already in place. As it is assumed that this new data is useful, the new rows must replace the old ones. This is performed by removing duplicate records and prioritizing the last occurrence.

Code fragment 3.9.- Function to append new file records

```
def push_into_db(filename, datalogger_db):
    new_data = pd.read_csv(filename)
    database = pd.read_csv(datalogger_db)

    # Check if the new file has more columns than the database
```

```
new_columns = set(new_data.columns) - set(database.columns)
if new_columns:
    # Add the new columns to the database and set values to 0 for previous rows
    for column in new_columns:
        database[column] = np.nan

# Append the new data to the database, overwriting rows with the same 'Datetime'
database = pd.concat([database, new_data]).drop_duplicates(subset='Datetime',
    keep='last')

# Order the database by 'Datetime' in ascending order
database.sort_values(by='Datetime', inplace=True)

# Delete columns with no values
database.dropna(axis=1, how='all', inplace=True)
print(f"File: {os.path.basename(filename)} has been added to database\n")
# Save the updated database back to the CSV file
database.to_csv(datalogger_db, index=False)
```

All throughout this section, it has been stated where each function got its information and where this information was stored. However, to have a clearer picture of the whole file system here is the complete folder architecture shown in diagram 3.3.

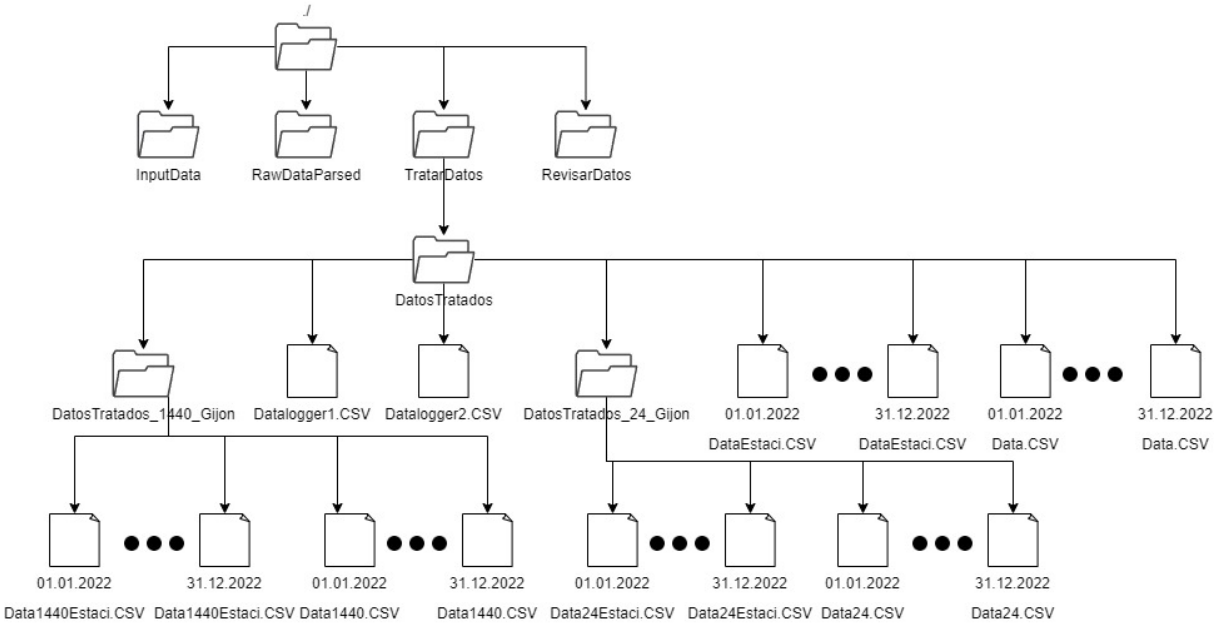


Figure 3.3.- Local file storage organisation

3.5.- Technologies used

In undertaking the tasks outlined in this chapter, numerous technologies were considered, recognizing the importance of aligning the chosen methods with the requisites, contextual factors, and potential future enhancements.

The goal of this section is to present the rationale behind choosing Python as the programming language, Pandas as the data manipulation library, and Jupyter Notebook as the interactive environment for efficient data processing and analysis. By examining the advantages and capabilities of each technology, it is possible to understand why this specific combination is well-suited for handling and extracting insights from PV panel sensor data.

3.5.1.- Python

Python [11] was chosen as the primary programming language for processing PV panel sensor data due to several key factors. Firstly, Python's simplicity and ease of use make it accessible to users of varying programming backgrounds, enabling efficient

collaboration and knowledge sharing within the research community. Secondly, Python boasts an extensive collection of libraries tailored for data manipulation, analysis, and visualization, providing a wealth of pre-built functionalities that accelerate development and reduce implementation complexity. Finally, Python's clean and readable syntax promotes code legibility, making it easier to understand, maintain, and debug complex data processing pipelines for future features.

3.5.1.1.- Integrated development environment

The integrated development environment (IDE) of choice to develop the script was PyCharm [12]. PyCharm offers a range of features that enhance the development experience, boost productivity, and support efficient coding practices. The IDE provides advanced code editing functionalities, including autocompletion, syntax highlighting, and error detection, which help developers write clean and error-free code. Additionally, PyCharm supports seamless integration with version control systems, such as Git to ensure code integrity. The interactive debugging capabilities of PyCharm enable developers to identify and resolve issues efficiently, further enhancing the reliability of the data processing workflow.

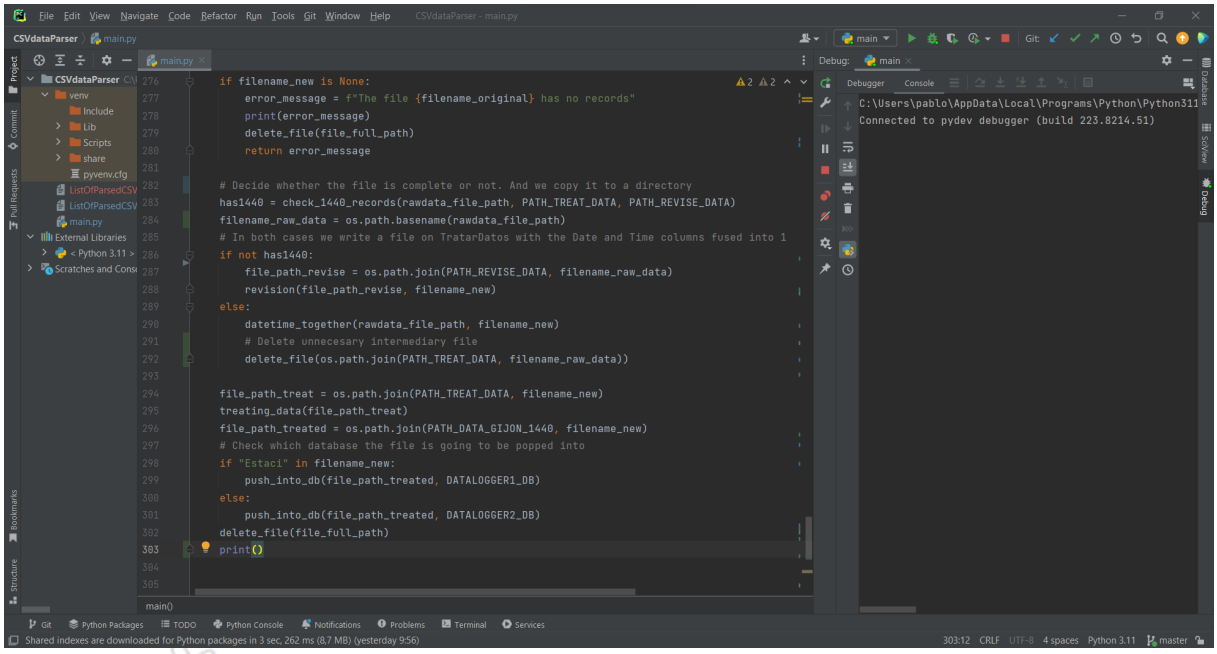


Figure 3.4.- Pycharm IDE. Debug mode

3.5.1.2.- Libraries

As Python is one of the most used programming languages in the field of data science and data analysis [18], there were a number of library options to choose from.

Among the various data manipulation libraries available, Pandas stands out for its versatility and powerful capabilities, making it an ideal choice for PV panel sensor data processing. Pandas offers a highly efficient and intuitive way to handle tabular data through its DataFrame object [13], which seamlessly handles CSV file loading, parsing, and manipulation. With Pandas, researchers and developers alike can effortlessly clean noisy or missing data, aggregate information, perform complex transformations, and filter data sets based on specific criteria. These versatile features empower users to gain valuable insights into their data. Pandas offers an array of functionalities that streamline the data processing workflow. It provides capabilities for data cleaning, such as handling missing values, removing duplicates, and transforming data types. Aggregation and grouping operations enable the calculation of descriptive statistics, averages, and other summary metrics. Furthermore, Pandas enables seamless data merging, concatenation, and reshaping for efficient integration of multiple data sources.

During the evaluation of libraries for processing the sensor data, the Polars library [21] emerged as an alternative to Pandas. Polars offers similar functionalities for data manipulation and analysis, with the added advantage of being designed for high-performance data processing. However, after careful consideration, it was decided to utilize Pandas as the primary library for several reasons. Firstly, Pandas has a larger and more established user base, providing a wealth of community support, resources, and documentation. Secondly, Pandas' extensive ecosystem of complementary libraries and tools makes it highly versatile for various data analysis tasks. Lastly, the familiarity and ease of use associated with Pandas among researchers and developers influenced the decision, as it enables simpler adoption and eases the learning curve. Overall, while Polars exhibits promising performance capabilities, the overall ecosystem, community support, and user-friendliness of Pandas ultimately led to its selection as the preferred library for our data processing pipeline.

3.5.2.- Jupyter Notebook

Jupyter Notebook [14] serves as an invaluable tool for individual developers engaged in processing sensor data, offering streamlined testing capabilities and facilitating easy visualization of resulting DataFrames. With its interactive and iterative nature, Jupyter Notebook allows developers to conveniently test functions and methods in real-time while viewing the modifications made to the DataFrames. This functionality is akin to unit testing, enabling developers to verify the correctness of their code and ensure the expected transformations and manipulations are accurately applied.

By executing code cells within Jupyter Notebook, developers can observe the effects of various operations, such as filtering, sorting, and data transformations, on the DataFrames instantly. This real-time feedback enables quick identification of any issues or discrepancies, improving the efficiency of the development process. Developers can iteratively refine their code and validate the modifications made to the DataFrames, ensuring the data processing pipeline functions as intended. In **figure 3.5** an example of some of the tests performed before adding the code to the main script is shown.

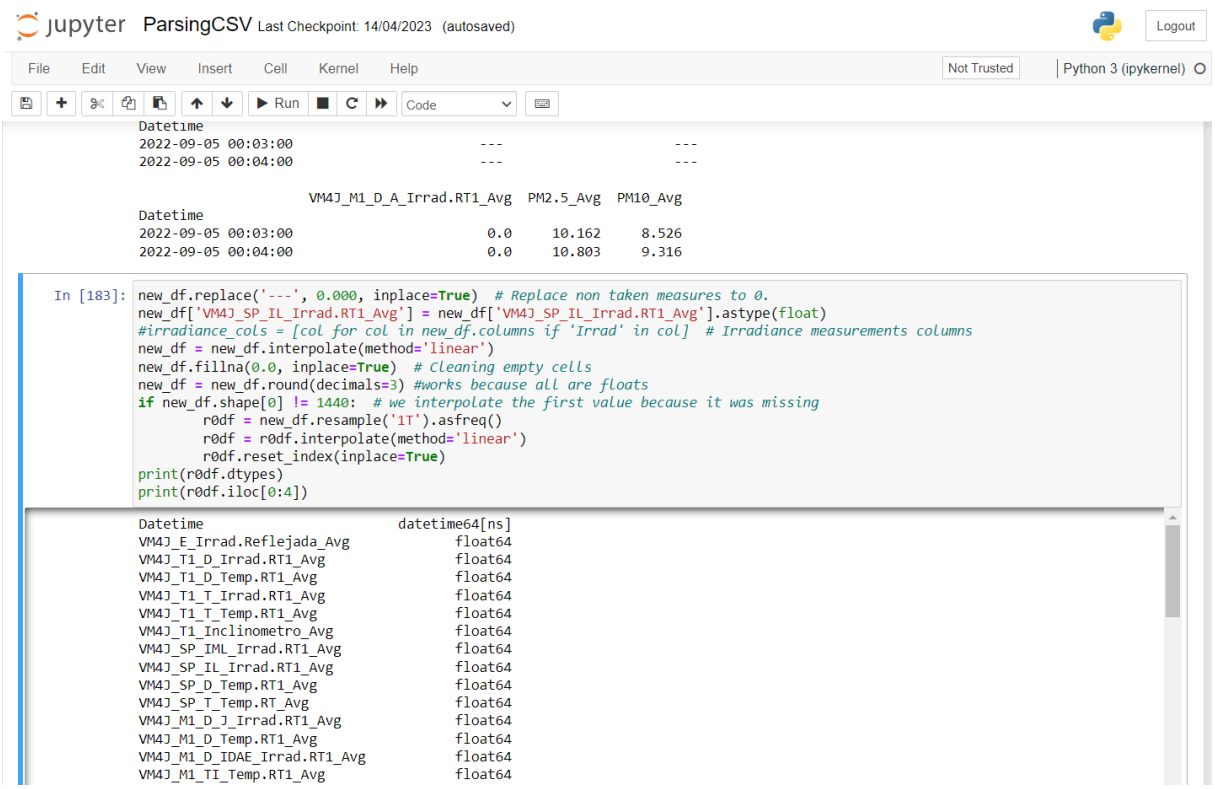


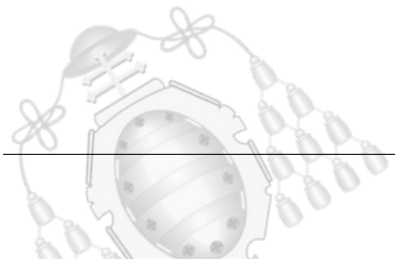
Figure 3.5.- Use case of Jupyter Notebook

Moreover, Jupyter Notebook's ability to integrate visualizations, such as plots and charts, allows developers to gain deeper insights into the transformed data. Visual representations of the modified DataFrames aid in identifying patterns, trends, and anomalies, thereby facilitating data analysis and quality assessment. This visualization aspect further enhances the testing process, enabling developers to visually confirm that the expected modifications have been correctly applied to the data.

In essence, Jupyter Notebook facilitates working on PV panel sensor data processing to test functions and methods efficiently, visually validate DataFrame modifications, and iteratively refine the code. By providing an interactive and visual environment, Jupyter Notebook not only supports the development process but also encourages thorough unit testing practices. This integrated testing and visualization capability ultimately improves the reliability and accuracy of the data processing workflow, contributing to the overall quality of the research outcomes.

3.5.3.- Excel

Excel was selected as the tool for examining CSV files generated by the data-cleaning script in this engineering paper for several reasons. First and foremost, Excel provides a familiar and user-friendly interface, making it accessible to a wide range of users with varying technical backgrounds. Its widespread availability and ease of use make it a practical choice for quickly visualizing and exploring data. Furthermore, Excel offers various data manipulation and analysis functionalities, allowing for simple calculations, filtering, and sorting operations. Additionally, Excel's graphical capabilities enable the creation of charts and graphs, aiding in the visual representation of raw and cleaned data from the PV panels. Overall, Excel proves to be a versatile and efficient tool for data inspection and initial analysis in the context of cleaning data from PV panels.



4. Web development

4.1.- Frontend

Aquí iría el cuerpo del anexo. Si no los necesitas, borra o comenta todo este bloque. Puedes añadir tantos anexos como necesites. Los anexos, como el resto de capítulos, se numeran solos, pero utilizando letras en lugar de números. El primer anexo será el "Anexo A". Todas las subsecciones, figuras y tablas se numerarán acorde a la letra del anexo en cuestión

A continuación, para que se vea como queda, tenemos en la Figura 4.1 a Vicentín, que siempre va a tope!.

Figure 4.1.- Me llamo Vicentín, toco en la banda de Alcoy y ¡siempre voy a tope!

4.2.- Backend

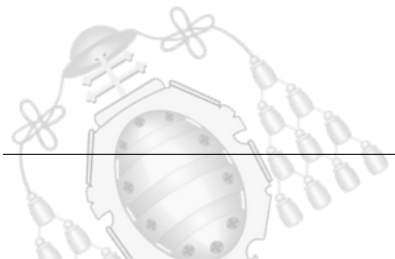
4.3.- Frameworks chosen

4.3.1.- Angular

4.3.2.- .NET Core

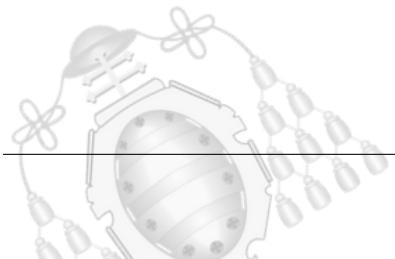
5. Deployment

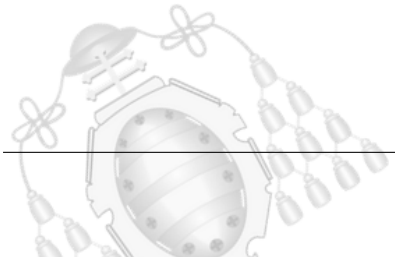
We deployed our app on the computer of the lab. It has Windows 10.



6. Conclusions and future improvements

Esto déjalo para el final, que hay gente que hace las conclusiones antes de empezar a trabajar de verdad.





Bibliography

[1] https://www.fronius.com/es-es/spain/energia-solar/instaladores-y-socios/productos-y-soluciones/herramientas-digitales-y-de-monitorizacion-para-tu-sistema-fotovoltaico/sistema-fv-monitorizacion-solarweb

[2] https://www.sma-iberica.com/

[3] https://www.solarwholesalers.com.au/solar-web-monitoring

[4] https://www.sma.de/es/productos/monitorizacion-y-control/webconnect

[5] Shaheer Ansari, Afida Ayob, *A Review of Monitoring Technologies for Solar PV Systems Using Data Processing Modules and Transmission Protocols: Progress, Challenges and Prospects* https://www.researchgate.net/publication/353355477_A_Review_of_Monitoring_Technologies_for_Solar_PV_Systems_Using_Data_Processing_Modules_and_Transmission_Protocols_Progress_Challenges_and_Prospects

[6] https://tuvatio.es/blog/como-funcionan-paneles-solares/

[7] https://shorturl.at/CEHOY

[8] Hussein A Kazem, Miqdam T Chaichan *Effect of Humidity on Photovoltaic Performance Based on Experimental Study* https://www.researchgate.net/publication/289546072_Effect_of_humidity_on_photovoltaic_performance_based_on_experimental_study

[9] https://www.kippzonen.com/Product/416/LOGBOX-SE-Data-Logger

[10] SolarEdge Software https://www.solaredge.com/en/products/software-tools

[11] https://www.python.org/

[12] https://www.jetbrains.com/es-es/pycharm/

[13] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[14] https://jupyter.org/

[15] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.interpolate.html

[16] Python Pandas Tutorial: Cleaning Data - Casting Datatypes and Handling Missing Values https://www.youtube.com/watch?v=KdmPHEnPJPs&list=RDCMUCCezIgC97PvUuR4_gbFUs5g&start_radio=1&rv=KdmPHEnPJPs&t=11&ab_channel=CoreySchafer

[17] https://www.geeksforgeeks.org/how-to-remove-timezone-from-a-timestamp-column-in-a-pandas-dataframe/

[18] Programming languages used in the field of data science https://shorturl.at/cpzEU

[19] https://pandas.pydata.org/docs/reference/api/pandas.readcsv.html

[20] https://pypi.org/project/suntime/

[21] https://www.pola.rs/