DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UN APLICATIVO PARA GENERACIÓN AUTOMATIZADA DE DATASET EN ENTORNOS IOT SMART CAMPUS

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

AUTOR
DANIEL URREA GREGORIO

DIRECTOR
IGNACIO MARTÍNEZ RUIZ

Universidad Zaragoza
1542

JUNIO, 2025

instituto de investigación en ingeniería de Aragón
Universidad de Zaragoza

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Dedicatorias y agradecimientos

A mi tutor Ignacio Martínez Ruiz.
Por su orientación, implicación y
entusiasmo en este trabajo.

A mis padres.
Por su incondicional apoyo,
confianza y cariño.

RESUMEN

Diseño, desarrollo e implementación de un aplicativo para generación automatizada de dataset en entornos IoT smart campus.

Realizado por: Daniel Urrea Gregorio **Dirigido por:** Ignacio Martínez Ruiz

En los últimos años, en el Campus Río Ebro de la Universidad de Zaragoza se han ido instalando sensores de adquisición de datos ambientales (temperatura, CO_2, etc.), en diversos espacios (aulas, pasillos, entradas, salas de estudio, etc.). Este Trabajo de Fin de Grado (TFG) tiene como objetivo la creación de un aplicativo completo para generación de conjuntos de datos (*datasets*) a partir de un ecosistema IoT. Se han desarrollado herramientas Python para limpiar, unificar y analizar datos registrados, permitiendo: procesado, depuración y filtrado de datos de forma automatizada y directa (eliminando procesos manuales) y diversos tipos de análisis multivariable para la toma de decisiones. Este conocimiento permite entender mejor las condiciones medioambientales de los *smart campus*, contribuyendo a su planificación sostenible, y promoviendo entornos universitarios más eficientes y habitables.

Con esta idea, la presente memoria se ha estructurado en 6 bloques principales, cada uno con objetivos específicos que abordan distintos aspectos del desarrollo y validación de *datasets*. En el primer capítulo se incluyen los antecedentes, el contexto y el objetivo del TFG, así como la estructura de la memoria. En el segundo capítulo se da un contexto sobre el ecosistema IoT y se especifica de dónde provienen los datos. El tercer capítulo detalla toda la problemática en el acondicionamiento de los datos. El cuarto capítulo explica la implementación, utilizando Python, del aplicativo y su funcionamiento, tanto en *frontend* como en *backend*. En el quinto capítulo se aborda la validación técnica de los *datasets*, incluyendo las pruebas de integridad, consistencia y fiabilidad de los datos obtenidos. Finalmente, el sexto capítulo analiza los resultados obtenidos y explora posibles líneas futuras de investigación.

Los resultados obtenidos en este trabajo representan un avance respecto a trabajos previos en el ámbito de la recopilación y generación de conjuntos de datos útiles, a la vez que abren nuevas vías para futuras investigaciones en la gestión de edificios inteligentes mediante tecnologías IoT, con el objetivo de avanzar hacia un *smart campus* eficiente y sostenible.

Índice de contenidos

Índice de figuras y tablas

1 Introducción y objetivos

La creciente complejidad de los edificios inteligentes (*smart buildings*) hace imprescindible la adopción de técnicas avanzadas de análisis de datos procedentes de su operación real. Estos métodos permiten acortar significativamente la distancia entre las especificaciones de diseño y el rendimiento observado en el día a día, proporcionando un *feedback* continuo. Gracias al tratamiento adecuado de la información, además de detectar patrones (consumo energético, ocupación, etc.), se puede anticipar comportamientos y anomalías antes de que se traduzcan en ineficiencias o fallos operativos. Esto hace que haya una mejor toma de decisiones en tiempo real, capaz de maximizar la eficiencia energética, equilibrando factores medioambientales, de ocupación y de coste.

Sin embargo, para garantizar el rigor de cualquier análisis, es fundamental partir de unos conjuntos de datos (*datasets*) que estén debidamente organizados, completamente limpios y estructurados bajo un criterio unificado. La carencia de una metodología sistemática en la recolección, normalización y enriquecimiento de estos *datasets* conlleva, con frecuencia, duplicidades, lagunas de información y alto riesgo de errores manuales, lo que dificulta enormemente la escalabilidad de los procesos analíticos y compromete la validez de los resultados.

Por todo ello, el presente Trabajo Fin de Grado (TFG) se orienta al diseño e implementación de un aplicativo capaz de generar de forma automatizada *datasets* en entornos *Internet of Things* (IoT) de tipo *smart buildings* en los campus universitarios (*smart campus*). La finalidad última es proveer de una herramienta ágil y robusta que sirva como base fiable para cualquier estudio posterior de gestión energética, mantenimiento predictivo o mejora de la experiencia de usuario en edificios inteligentes.

1.1 Antecedentes y contexto

Los edificios representan aproximadamente el 40% del consumo energético y el 36% de las emisiones de gases de efecto invernadero a escala global, posicionándose como uno de los sectores clave para la transición hacia la sostenibilidad [1]. Dado que pasamos entre el 80% y el 90% de nuestro tiempo en espacios interiores [2], optimizar la operación y el uso de la energía en el entorno construido se erige como una prioridad, tanto para cumplir con los compromisos del Acuerdo de París [3], como para alcanzar los Objetivos de Desarrollo Sostenible (ODS) de la Unión Europea, que exige una reducción de al menos el 32,5% en el consumo energético de los edificios antes de 2030.

En este escenario, la Universidad de Zaragoza ha puesto en marcha su iniciativa *smart campus* [4], que integra más de 200 sensores ambientales (midiendo variables como CO_2, temperatura, humedad y ocupación, etc.) distribuidos y geolocalizados en más de 100 espacios del Campus Río Ebro. Estos dispositivos IoT generan en tiempo real miles de registros cada hora, lo cual abre enormes oportunidades para la mejora continua de la gestión energética, la planificación de espacios y el bienestar de la comunidad universitaria.

No obstante, el flujo de extracción y tratamiento de esta cantidad masiva de datos se ha venido efectuando hasta ahora en un entorno Matlab y de manera manual, a base de *scripts* aislados que, con el tiempo, han ido acumulando múltiples casuísticas y excepciones solapadas. Esta práctica ha supuesto una elevada carga de trabajo, largos tiempos de procesado y un riesgo elevado de errores humanos, dificultando la reproducibilidad de los resultados y el escalado de nuevas líneas de análisis. Así, tras los primeros desarrollos de análisis sobre estos proyectos, se detectó la necesidad de estandarizar y automatizar el ciclo completo de generación de los *datasets*, desde la ingesta y normalización inicial hasta la incorporación de fuentes externas. Por ello, este TFG plantea el diseño, desarrollo e implantación de un aplicativo, constituido como herramienta modular de generación de *datasets*, basado en un modelo de múltiples capas:

- **Capa de recogida y normalización:** Unifica formatos, nombres de variables y esquemas de metadatos para garantizar la coherencia de todos los ficheros de entrada.
- **Módulo de limpieza y alineación temporal:** Detecta y gestiona valores nulos o erróneos; ajusta series con diferentes frecuencias de muestreo; sincroniza los *timestamps* de todas las variables.
- **Componente de enriquecimiento:** Integra datos de fuentes externas como: información meteorológica (consultando el servicio de la Agencia Estatal de Meteorología, AEMET) [5], información de la ocupación (consultando el servicio de reserva de aulas de la Universidad de Zaragoza) [6], entre otras.
- ***Frontend* de configuración y descarga:** Interfaz web intuitiva que permite a usuarios sin conocimientos avanzados introducir los parámetros para las ejecuciones y descargar los *datasets* resultantes.
- ***Backend* optimizado para grandes volúmenes:** Arquitectura escalable, optimizada y organizada.

Como se describe a lo largo del TFG, con este aplicativo se busca agilizar el preprocesado de datos, reducir los tiempos de configuración, y asegurar la obtención de *datasets* fiables, completos y reproducibles. Todo ello ayudará en pasos posteriores como el desarrollo de modelos predictivos y de optimización que conduzcan a una gestión energética más eficiente, un coste operativo más bajo y una experiencia de usuario más satisfactoria en los edificios inteligentes del campus. A continuación, se detallan los objetivos específicos del proyecto y la estructura de la memoria.

1.2 Objetivos

El propósito general de este TFG radica en la creación de una herramienta automatizada para la recogida, limpieza y organización de conjuntos de datos (*datasets*) en entornos *smart campus*. Con este objetivo, el TFG se estructura en torno a 6 objetivos claramente definidos, que orientan tanto el proceso de desarrollo de la herramienta como la evaluación de su eficacia. A continuación, se detalla cada uno, explicando su alcance y motivación:

- Identificar la problemática inicial: realizar un diagnóstico detallado de los flujos de datos procedentes de los sensores desplegados en el campus. Esto implica inventariar las fuentes (CO_2, temperatura, humedad, etc.), confirmar su correcta nomenclatura, etc.
- Desarrollar funciones de acondicionamiento y procesado de datos: el núcleo técnico del proyecto es la creación de un conjunto de librerías en Python que automaticen tareas de limpieza y normalización, como: detección y tratamiento de valores “Not a Number” (NaN), remuestreo y sincronización temporal, unificación de variables con diferentes cadencias, mapeo semántico garantizando que todas las variables compartan un vocabulario y formato común, etc.
- Diseñar y esquematizar el aplicativo: con el diagnóstico en mano, se procederá al diseño de la arquitectura de la herramienta, siguiendo un enfoque modular en capas: recogida, preprocesado, almacenamiento y acceso/exportación.
- Crear el aplicativo completo: una vez disponibles las funciones de bajo nivel, se integrarán en una aplicación que combine *frontend* y *backend* y. El *backend* ejecutará de forma orquestada los pipelines de datos; mientras que el *frontend* proveerá formularios de selección de aulas, selección del formato de exportación, valores de interpolación y rangos de fechas y horas. Para ello se utilizarán las funciones de limpieza y preprocesado ya creadas en Python.
- Validar la calidad técnica de los *datasets*, definir e implementar protocolos de verificación de integridad, consistencia y fiabilidad de los datos procesados, incluyendo análisis de completitud, detección de anomalías y comprobación de interpolaciones para asegurar la excelencia del *dataset* final.
- Proponer, a partir de los resultados obtenidos, una serie de líneas futuras de trabajo sobre las que seguir desarrollando este TFG partiendo de la herramienta creada.

1.3 Estructura de la memoria

La presente memoria se compone de 6 capítulos, con los siguientes contenidos:

- El segundo capítulo, tras la introducción y la definición de los objetivos, presenta la infraestructura de sensores del ecosistema IoT, y la proveniencia de todas las variables utilizadas.
- El tercer capítulo explica todas las problemáticas que se han ido solucionando durante la fase de el acondicionamiento de datos, incluyendo valores ausentes y huecos temporales, NaN (apartado 3.1.1), desalineación temporal en los *timestamps* (apartado 3.1.2), heterogeneidad semántica y de formatos (apartado 3.1.3) y detección y tratamiento de anomalías (apartado 3.1.4).
- El cuarto capitulo empieza hablando de la metodología de trabajo (apartado 4.1), continúa explicando la etapa de diseño esquemático (apartado 4.2). Para, a continuación, explicar todo el proceso de desarrollo y deseño del *frontend* (apartado 4.3) y todas las funciones utilizadas en el *backend* (apartado 4.4). Centrándose en las diversas funcionalidades como la selección de intervalos entre datos, el formato de exportación, el rango de fechas y horas para realizar un estudio más preciso.
- El quinto capitulo describe las diferentes casuísticas (apartado 5.1) que permiten la validación del aplicativo, además de los diferentes mecanismos (apartado 5.2) utilizados para poder comprobar la calidad de los datos y poder saber que intervalos de tiempo tienen los datos más completos.
- Finalmente, el sexto capitulo desglosa las principales conclusiones obtenidas en el TFG, como lecciones aprendidas a partir de las que seguir construyendo conocimiento en este ámbito, y plantea líneas futuras a seguir para dar continuidad a este trabajo.

Además, se incluyen 2 anexos, con los siguientes contenidos:

- En el anexo I se detalla el código Python del *script* `app.py`
- En el anexo II se detalla el código Python del *script* `procesar_datos.py`

2 Ecosistema IoT

2.1 Ecosistema IoT

Como se menciona en el capítulo 1, este TFG se desarrolla dentro de un ecosistema IoT, que lleva varios años implementándose en el Campus Rio Ebro de la Universidad de Zaragoza, como parte de su iniciativa *smart campus*. Se trata de un ecosistema "medir-analizar-decidir y actuar", cuyo objetivo es recopilar información para comprender en profundidad el funcionamiento de los edificios [7]. Esta información podrá ser estudiada en ámbitos variados y con distintas metodologías para extraer conclusiones y tomar decisiones que contribuyan a una mayor eficiencia y mejor uso de las infraestructuras. El ecosistema IoT del Campus Río Ebro cuenta actualmente con más de 200 de sensores desplegados por los distintos edificios de la EINA, permanentemente tomando y enviando medidas en tiempo real. Estos sensores se dividen en distintos grupos según sus características y los parámetros de calidad ambiental que miden. Resumidamente, los dispositivos desplegados son:

- Milesight AM307 [8]: sensores que miden temperatura, nivel de CO_2, humedad, presión atmosférica, luminosidad y concentración de partículas en el aire. Se encuentran instalados en la mayoría de las aulas del campus.

- ARANET 4 PRO [9]: sensores que toman medidas de temperatura, nivel de CO_2, humedad y presión atmosférica. Se complementan con los dispositivos Milesight AM307, habiendo un sensor de uno u otro tipo instalado en casi cada aula.

- Milesight EM500-PT100 [10]: sensores de temperatura exterior que miden la temperatura de las fachadas de los edificios mediante una sonda que penetra en el hormigón. Se encuentran instalados principalmente en el edificio Ada Byron, situados en fachadas con distintas orientaciones.

- Khomp METEO [11]: estaciones meteorológicas que se instalan en el exterior de los edificios y toman diversas medidas relativas a la temperatura, presión atmosférica, humedad, luminosidad y velocidad del viento. Hay una estación instalada en el tejado del Ada Byron y otra en el Instituto de Investigación de Ingeniería de Aragón (I3A).

- Siemens QPA1004 [12]: sensores que miden temperatura, nivel de CO_2 y humedad. Se ubican dentro de las pantallas que muestran estos datos como, por ejemplo, la que está instalada en la cafetería del Ada Byron.

- Milesight VS121 [13]: sensores de ocupación instalados en los techos de los espacios educativos para monitorizar el flujo de personas.

- Xovis PC2SE [14]: sensores de conteo de personas ubicados en los umbrales de los edificios.

La tabla que se muestra a continuación resume las características de los distintos tipos de sensores, incluyendo los nombres por los que son referidos comúnmente, las medidas que toman y el intervalo en el que envían datos. Podemos observar que los sensores también envían datos como el nivel de batería o la relación señal-ruido, que no se utilizan en el análisis de datos, pero sirven para monitorizar su correcto funcionamiento. La tabla 1 mostrada a continuación resume las características de los distintos tipos de dispositivos

Marca y modelo	Nombre	Identificador	Medidas	Intervalo
Milesight AM307	AM307	(Código *SIGEUZ*) CRE…	atmosphericPressure, battery, co2, fCnt, humidity, integer, lightLevel, pir, pressure, rssi, snr, temperature, tvoc	10 min
ARANET 4 PRO	ARANET	ARA-PRO	atmosphericPressure, battery, co2, humidity, rssi, temperature	10 min
Milesight EM500-PT100	EM500	MIL-TEX	battery, fCnt, rssi, snr, temperature	10 min
Khomp Meteo	METEO	KHO-MET	atmosphericPressure, averageWindSpeed, fCnt, gustWindSpeed, humidityInterior, humidityInternal, lightLevel, lux, rain, rainLvl, rssi, snr, temperatureExterior, temperatureInterior, temperatureInternal, uv, windDirection	5 min
Siemens QPA1004	SIEMENS	IMP-SLM	Battery, co2, fCnt, humidity, rssi, snr, temperature	10 min
Milesight VS121	VS121	MIL-121	bw, fCnt, fw, rssi, snr	1 min
Xovis PC2SE	XOVIS	XOV-2SE	bw, fw	1 min

Tabla 1. Tipos de sensores, sus identificadores, las telemetrías que proporcionan y su periodo de medida (min).

En este caso, se hará uso principalmente de los sensores AM307, ARANET 4 PRO y EM500 (en el caso de la información sobre la ocupación de las aulas, queda pendiente integrar los datos de los sensores VS121 y XOVIS una vez se confirme la fiabilidad de sus medidas monitorizadas).

La información detallada de cada sensor con su lugar de instalación se registra en una hoja Excel, mostrada en la Figura 1. Esta Excel, llamada *IoB_spaces_devices.xlsx* es el documento de referencia que contiene los datos actualizados de todos los dispositivos y que emplearemos para obtener cualquier información necesaria relativa a ellos. La Excel incluye los nombres de los distintos espacios del Campus Río Ebro (tanto aulas como despachos, zonas comunes, fachadas, etc.) y su código SIGEUZ asociado, que es un código de espacio geolocalizado asignado por el Servicio de Información Geográfica de la Universidad de Zaragoza (SIGEUZ) [15]. También se puede encontrar la zona a la que pertenece cada espacio, determinada por sus características físicas y su ubicación.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	
1	space	SIGEUZ	campus	building	floor	zone	use	max occup	size	area (m2)	height	volume (m3)	orient.sp	orient.wn	Información
2	Ada.0.01	CRE.1200.00.040	CRE	AB	0	AB.A	CR	120	L	150,48	3,55	534,20	S	SO	Aula control (AM307 + EM500 + VS121)
3	Ada.0.02	CRE.1200.00.050	CRE	AB	0	AB.A	CR	120	L	150,47	3,55	534,17	S	SE	Aula control (AM307 + EM500 + VS121)
4	Ada.0.03	CRE.1200.00.060	CRE	AB	0	AB.B	CR	80	M	118,82	3,55	421,81	S	1	Aula control (AM307 + EM500 + VS121)
5	Ada.0.04	CRE.1200.00.070	CRE	AB	0	AB.B	CR	80	M	118,82	3,55	421,81	S	1	Aula grande Planta 0
6	Ada.0.05	CRE.1200.00.080	CRE	AB	0	AB.B	CR	80	M	118,82	3,55	421,81	S	1	Aula grande Planta 0
7	Ada.0.06	CRE.1200.00.090	CRE	AB	0	AB.B	CR	80	M	118,82	3,55	421,81	N	1	Aula grande Planta 0
8	Ada.0.07	CRE.1200.00.100	CRE	AB	0	AB.B	CR	80	M	118,82	3,55	421,81	N	1	Aula grande Planta 0
9	Ada.1.11	CRE.1200.01.030	CRE	AB	1	AB.C	CR	80	M	118,82	2,95	350,52	S	1	Aula mediana Planta 1
10	Ada.1.12	CRE.1200.01.040	CRE	AB	1	AB.C	CR	80	M	118,82	2,95	350,52	S	1	Aula mediana Planta 1
11	Ada.1.13	CRE.1200.01.050	CRE	AB	1	AB.C	CR	80	M	118,82	2,95	350,52	S	1	Aula control (AM307 + EM500 + VS121)
12	Ada.1.14	CRE.1200.01.060	CRE	AB	1	AB.C	CR	80	M	118,82	2,95	350,52	N	1	Aula mediana Planta 1
13	Ada.1.15	CRE.1200.01.070	CRE	AB	1	AB.C	CR	80	M	118,82	2,95	350,52	N	1	Aula mediana Planta 1
14	Ada.2.21	CRE.1200.02.030	CRE	AB	2	AB.D	SM	40	S	74,68	2,90	216,57	S	2	Seminario Planta 2
15	Ada.2.22	CRE.1200.02.040	CRE	AB	2	AB.D	SM	40	S	74,68	2,90	216,57	S	2	Seminario Planta 2
16	Ada.2.23	CRE.1200.02.050	CRE	AB	2	AB.D	SM	40	S	74,68	2,90	216,57	S	2	Aula control (AM307 + EM500 + VS121)
17	Ada.2.24	CRE.1200.02.070	CRE	AB	2	AB.D	SM	40	S	74,68	2,90	216,57	N	2	Seminario Planta 2
18	Ada.2.25	CRE.1200.02.080	CRE	AB	2	AB.D	SM	40	S	74,68	2,90	216,57	N	2	Seminario Planta 2

Figura 1. Captura de la hoja excel IoB_spaces_devices.xlsx

El ecosistema IoT monitoriza en tiempo real los datos de todos los sensores desplegados y almacena dichos datos en una base de datos que puede consultarse en una estructura de carpetas, como muestran las Figura 2 y 3. Esta base de datos es de la que se han extraído los valores empleados en este TFG, como se detalla a continuación en el apartado 2.2.

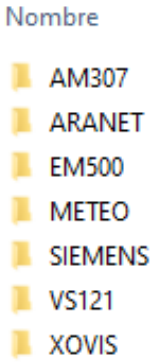


Figura 2. Estructura de carpetas por tipo de sensor

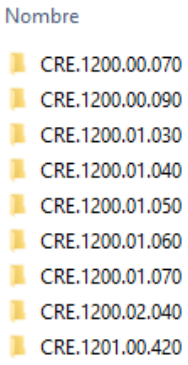


Figura 3. Datos asociados a cada sensor

2.2 Adquisición de datos

Un *dataset* es el conjunto de datos que se utilizará para analizar información del *smart campus*, estudiar patrones de comportamiento, entrenar modelos predictivos, etc. Por todo ello, su proceso de generación resulta crucial para garantizar la calidad y precisión de los resultados obtenidos.

La selección de las variables y datos adecuados es un paso fundamental en la creación de subconjuntos de datos (a partir del *dataset* principal) ya que influirá directamente en la capacidad de futuros análisis para ofrecer resultados precisos y relevantes. Es importante tener en cuenta que la composición de los diversos *dataset* variará en función del tipo de estudio que se desee realizar. Cada estudio requiere datos y variables específicas que reflejen las características del sistema a predecir, por lo que es necesaria una cuidadosa elección de las variables utilizadas para evitar sesgos y garantizar la efectividad del modelo. Por ello se han seleccionado un conjunto de variables que da un contexto completo de la situación en un aula.

Así, el primer paso para la generación de *datasets* es clasificar los datos disponibles y conocer el proceso de obtención de cada uno de ellos. En este caso, se dispone de tres tipos de datos:

- **Datos provenientes de sensores IoT (DATA_IoB).** Estos son los datos que provienen del ecosistema IoT del Campus Río Ebro. Estos sensores recogen en tiempo real medidas de intervalos de 10 minutos del CO_2, temperatura, humedad, luz, presión, etc. Anteriormente se utilizaba una aplicación en Matlab (IoB_App) [4], con la que se podían exportar estos datos, pero ahora con la nueva herramienta más automatizada es más sencillo aún, pudiendo elegir el intervalo de fechas y horas, el formato y la carpeta de exportación.
- **Datos de fuentes externas (DATA_EXT).** Son los datos que se obtienen desde fuentes externas como: AEMET [5], reserva de aulas [6], servicio de mantenimiento de la Universidad de Zaragoza [16], etc. Para la obtención de este tipo de datos se han creado funciones con Python y se han integrado en el *backend*. Estas funciones extraen la información de la fuente externa y la ajustan al formato de fechas e intervalos seleccionado. En el siguiente apartado se explicarán cada una de las funciones creadas para cada variable externa.
- **Datos creados "artificialmente" (DATA_ART).** Son fuentes de datos construidas a partir de las anteriores, que no corresponden directamente ni a un dato de un sensor ni a un dato externo sino a partir de operaciones con datos de sensores o externos. Los procesos de generación de estos datos se detallarán en el siguiente apartado.

2.3 Variables de estado

Una vez descrito el proceso de obtención de los datos según su origen (DATA_IoB, DATA_EXT o DATA_ART), en este apartado se van a definir las variables que compondrán los *datasets*. De entre todas las variables posibles, se ha hecho una selección de 11 variables que aportan una información y dan un contexto lo suficientemente amplio como para poder extraer conclusiones y poder realizar modelos con dichos datos.

A cada variable se le ha asignado un nombre de variable siguiendo un formato ordenado y que deje fuera las ambigüedades a la hora de referirse a cada variable. En primer lugar, se asigna el tipo de variable (`tmp`, `co2`, `hvac`, `occ`), después se le pone un "apellido" (por ejemplo, `indoor`, `meteo`, `aemet`) y finalmente el tratamiento de la variable (`max`, `min`, etc.). Además, se han separado todas estas variables en tres disposiciones distintas: *base keys*, *external keys* y *advanced keys*. Según su nivel de criticidad y ámbito de interacción con el sistema: *base keys* agrupan las variables esenciales para la operación interna, *external keys* incluyen aquellas que gestionan la comunicación con servicios externos, y *advanced keys* contienen los parámetros de configuración avanzada que permiten ajustes más precisos. Siguiendo estas reglas, he asignado los siguientes nombres a las variables como se puede ver en la tabla 2:

base keys	external keys	advanced keys
co2	tmp_aemet.max	humidity
tmp_indoor	tmp_aemet.min	light
tmp_meteo	occ_boolean	pressure
	occ_people	
	hvac_boolean	
	calendar_categories	

Tabla 2. Lista de variables utilizadas en este TFG

A continuación, se detalla cada variable y se explica el proceso de obtención:

- **`co2` y `tmp_indoor`.** Las variables `co2` y `tmp_indoor` representan el nivel de CO_2 y la temperatura en el interior de cada aula. Estas dos variables se obtienen conjuntamente ya que son tomadas del mismo sensor, por lo tanto, sus *timestamps* a la hora de obtención es el mismo. En la figura 4 se muestra un ejemplo de los datos disponibles de las variables `co2` y `tmp_indoor`.

	A	B	C
1	Time	co2	tmp_indoor
2	02/06/2024 0:07	459.0	23.4
3	02/06/2024 0:17	460.0	23.4
4	02/06/2024 0:27	450.0	23.4
5	02/06/2024 0:37	453.0	23.4
6	02/06/2024 0:47	448.0	23.3
7	02/06/2024 0:57	457.0	23.4
8	02/06/2024 1:07	452.0	23.2
9	02/06/2024 1:17	468.0	23.2
10	02/06/2024 1:27	455.0	23.2
11	02/06/2024 1:37	453.0	23.2
12	02/06/2024 1:47	450.0	23.2
13	02/06/2024 1:57	452.0	23.1
14	02/06/2024 2:07	450.0	23.1
15	02/06/2024 2:17	452.0	23.1

Figura 4. Ejemplo de datos co2 y tmp_indoor

- **`tmp_meteo`.** La variable `tmp_meteo` corresponde a la temperatura exterior obtenida por una estación meteorológica situada en el tejado de los edificios del Campus Rio Ebro. Existen dos estaciones meteorológicas, cuyos nombres e identificadores están en la tabla 3. Para este TFG tras comprobar los datos de ambas estaciones meteorológicas, se usan los datos del sensor KHO-MET-001, ya que tiene una mayor cantidad de datos. Se extraen los datos como si fuese un sensor más del aula se puede ver un ejemplo en la figura 5.

Id IoB_spaces_devices	I3A.WL.E	I3A.WL.O
SENSOR	KHO-MET-001	KHO-MET-003

Tabla 3. Denominación sensores tmp_meteo

	A	B
1	**Time**	**tmp_meteo**
2	02/06/2024 8:12	13.8
3	02/06/2024 8:17	14.2
4	02/06/2024 8:22	14.3
5	02/06/2024 8:27	14.5
6	02/06/2024 8:32	14.8
7	02/06/2024 8:38	14.9
8	02/06/2024 8:43	15.3
9	02/06/2024 8:48	15.6
10	02/06/2024 8:53	15.6
11	02/06/2024 8:58	15.8
12	02/06/2024 9:03	15.9

Figura 5. Ejemplo de datos tmp_meteo

- **`tmp_aemet.max` y `tmp_aemet.min`.** Estas dos variables externas corresponden a la temperatura máxima y mínima obtenidas a partir de registros climatológicos de la agencia estatal de meteorología AEMET. Estos datos muestran la temperatura máxima y mínima por día, por lo que durante el procesado de los datos hace falta ponerlos en el mismo formato de intervalos. Para obtener estas variables se ha desarrollado una función Python (`GetTemperatureAEMET.py`), como se muestra en el ejemplo de la figura 6.

	A	B	C
1	**Time**	**tmp_aemet.max**	**tmp_aemet.min**
2	**03/06/2024 00:00:00**	28,9	14,5
3	**03/06/2024 00:10:00**	28,9	14,5
4	**03/06/2024 00:20:00**	28,9	14,5
5	**03/06/2024 00:30:00**	28,9	14,5
6	**03/06/2024 00:40:00**	28,9	14,5
7	**03/06/2024 00:50:00**	28,9	14,5
8	**03/06/2024 01:00:00**	28,9	14,5
9	**03/06/2024 01:10:00**	28,9	14,5
10	**03/06/2024 01:20:00**	28,9	14,5
11	**03/06/2024 01:30:00**	28,9	14,5
12	**03/06/2024 01:40:00**	28,9	14,5
13	**03/06/2024 01:50:00**	28,9	14,5
14	**03/06/2024 02:00:00**	28,9	14,5
15	**03/06/2024 02:10:00**	28,9	14,5
16	**03/06/2024 02:20:00**	28,9	14,5
17	**03/06/2024 02:30:00**	28,9	14,5

Figura 6. Ejemplo de datos tmp_aemet.max y tmp_aemet.min

- **occ_boolean y occ_people.** occ_boolean es una variable externa con datos binarios que determinan si un aula está vacía u ocupada en un instante de tiempo (vacía 0, ocupada 1). occ_people en cambio determina el número de personas matriculadas en la asignatura que se está impartiendo. Ambas variables se obtienen de forma conjunta con el mismo script de Python llamado generarArchivosOcupacionAnual.py. Los datos se obtienen de la web de reserva de aulas [6] y luego se opera sobre los datos extraídos para obtener ambas variables para los periodos desde el 1 de enero de 2023 hasta la fecha actual. Para ello se dispone de una serie de funciones en Python:
  - **getReservaAulas.py:** Extrae los datos de la web de reserva de aulas.
  - **getTableFromWeb.py:** Complementa la función getReservaAulas.py.
  - **table1_Occupancy.py:** Lee los datos extraídos de la web de reserva de aulas y asigna un 0 o un 1 según si el aula está ocupada o no.
  - **table2_MatriculateOccupancy.py:** Lee los datos extraídos de la web de reserva de aulas y asigna el valor de alumnos matriculados en la asignatura compartida.
  - **generate_anual_Occupancy.py:** Genera ficheros con los datos tanto de ocupación booleana como de ocupación matriculada por año natural, creando un fichero de 2023, otro de 2024 y finalmente de 2025. Un ejemplo de fichero de la variable occ_boolean para el año natural 2025 se muestra en la figura 7.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Hora	A.01 (120)	A.02 (120)	A.03(80)	A.04(80)	A.05(80)	A.06(80)	A.07 (80)	A.11(80)	A.12(80)	A.13(80)	A.14(80)	A.15 (80)	S. A.21(40)	S. A.22(40)	S. A.23(40)	S. A.24(40)	S. A.25(40)
2	01/01/2025 0:00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	01/01/2025 0:10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	01/01/2025 0:20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	01/01/2025 0:30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	01/01/2025 0:40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	01/01/2025 0:50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	01/01/2025 1:00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	01/01/2025 1:10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	01/01/2025 1:20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	01/01/2025 1:30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	01/01/2025 1:40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	01/01/2025 1:50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	01/01/2025 2:00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	01/01/2025 2:10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	01/01/2025 2:20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	01/01/2025 2:30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	01/01/2025 2:40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	01/01/2025 2:50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	01/01/2025 3:00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 7. Ejemplo de datos occ_boolean

- **hvac_boolean.** Esta variable muestra en datos binarios el estado de la climatización del edificio, indicando con un 0 que la calefacción esta apagada y con un 1 que está encendida. La fuente de datos es un documento PDF proporcionado por la Oficina Verde de la Universidad de Zaragoza [16]. Tras convertir los datos externos de los horarios de encendido y apagado de HVAC a un fichero Excel, se crea una función de Python que lee los intervalos de tiempo en los que HVAC está encendida y genera, en una hoja aparte, el vector booleano con intervalos de 10 minutos. Puede verse un ejemplo en la figura 8.

	A	B
1	Time	hvac_boolean
2	05/02/2024 07:10:00	1
3	05/02/2024 07:20:00	1
4	05/02/2024 07:30:00	1
5	05/02/2024 07:40:00	1
6	05/02/2024 07:50:00	1
7	05/02/2024 08:00:00	1
8	05/02/2024 08:10:00	1
9	05/02/2024 08:20:00	1
10	05/02/2024 08:30:00	1
11	05/02/2024 08:40:00	1
12	05/02/2024 08:50:00	1
13	05/02/2024 09:00:00	1
14	05/02/2024 09:10:00	1
15	05/02/2024 09:20:00	1
16	05/02/2024 09:30:00	1
17	05/02/2024 09:40:00	1
18	05/02/2024 09:50:00	1

Figura 8. Ejemplo de datos hvac_boolean

- **`calendar_categories`.** Se han categorizado los distintos días de la semana a lo largo de los diversos meses del año, periodos lectivos, festivos, etc. Para ello, se ha llevado a cabo un análisis y categorización del calendario académico de la Universidad de Zaragoza según su actividad. En el caso de este TFG, se han utilizado los cursos 22-23, 23-24 y 24-25. Así, se ha definido un código numérico para cada categoría de día, diferenciando el día de la semana, los días lectivos con clases, sin clases y festivos (ver figura 9):
  - **Códigos del 1 al 7.** Días de la semana respectivamente de lunes (=1) a domingo (=7).
  - **Código 8.** Días festivos, dónde la universidad estuvo completamente cerrada.
  - **Código 9.** Días lectivos sin clases y sin evaluación. Universidad abierta, pero sin clases ni evaluación.

	A	B
1	Time	calendar_categories
2	03/06/2024 07:00:00	9
3	03/06/2024 07:10:00	9
4	03/06/2024 07:20:00	9
5	03/06/2024 07:30:00	9
6	03/06/2024 07:40:00	9
7	03/06/2024 07:50:00	9
8	03/06/2024 08:00:00	9
9	03/06/2024 08:10:00	9
10	03/06/2024 08:20:00	9
11	03/06/2024 08:30:00	9
12	03/06/2024 08:40:00	9
13	03/06/2024 08:50:00	9
14	03/06/2024 09:00:00	9

Figura 9. Ejemplo de datos calendar_categories

- **`humidity, light y pressure.`** La variable `humidity` mide la humedad relativa del aire, la proporción de vapor de agua respecto al máximo que el aire puede contener a esa temperatura, la unidad es un porcentaje. `Light` mide el nivel de luminancia, la cantidad de luz que incide en el sensor, en lux. Finalmente, `pressure` se refiere a la presión atmosférica, la fuerza que ejerce la columna de aire sobre el sensor, y se mide en hectopascales. Estas variables permiten ajustes más avanzados, aunque tengan una utilidad o importancia menor que otras. Ver un ejemplo en la figura 10.

	A	B	C	D
1	Time	pressure	humidity	light
2	02/06/2024 0:06	994.8	37.5	0.0
3	02/06/2024 0:16	994.7	37.5	0.0
4	02/06/2024 0:26	994.8	37.5	0.0
5	02/06/2024 0:36	994.8	37.5	0.0
6	02/06/2024 0:46	994.9	37.5	0.0
7	02/06/2024 0:56	994.9	37.5	0.0
8	02/06/2024 1:06	994.8	37.5	0.0
9	02/06/2024 1:16	994.8	37.5	0.0
10	02/06/2024 1:26	994.7	37.5	0.0
11	02/06/2024 1:36	994.7	37.5	0.0
12	02/06/2024 1:46	994.7	38.0	0.0
13	02/06/2024 1:56	994.7	38.0	0.0
14	02/06/2024 2:06	994.6	38.0	0.0
15	02/06/2024 2:16	994.5	38.0	0.0
16	02/06/2024 2:26	994.5	38.0	0.0
17	02/06/2024 2:36	994.5	38.0	0.0
18	02/06/2024 2:46	994.6	38.0	0.0
19	02/06/2024 2:56	994.6	38.0	0.0
20	02/06/2024 3:06	994.7	38.0	0.0
21	02/06/2024 3:16	994.8	38.0	0.0
22	02/06/2024 3:26	995.0	38.0	0.0
23	02/06/2024 3:36	994.9	38.0	0.0
24	02/06/2024 3:46	995.0	38.0	0.0
25	02/06/2024 3:56	995.1	38.0	0.0
26	02/06/2024 4:06	995.3	38.0	0.0

Figura 10. Ejemplo de datos pressure, humidity y light

Dataset. Finalmente, después de la selección de las variables y la nomenclatura, así es como quedaría un Excel con todas las variables con un intervalo de 10 minutos unificado (ver figura 11).

	A	B	C	D	E	F	G	H	I	J	K	L
1	Time	co2	tmp_indoor	tmp_meteo	tmp_aemet.max	tmp_aemet.min	occ_boolean	occ_people	humidity	light	pressure	hvac_boolean
2	03/06/2024 08:00:00	703	19,8	19,1	28,9	14,5	0	0	40,5	0	991,1	0
3	03/06/2024 08:10:00	703	19,8	19,4	28,9	14,5	0	0	40,5	0	991,1	0
4	03/06/2024 08:20:00	699	19,8	19,7	28,9	14,5	0	0	40,5	0	991	0
5	03/06/2024 08:30:00	702	19,9	20,5	28,9	14,5	0	0	40,5	0	990,9	0
6	03/06/2024 08:40:00	703	20	20,1	28,9	14,5	0	0	40,5	0	990,8	0
7	03/06/2024 08:50:00	696	20,1	20,4	28,9	14,5	0	0	40,5	0	990,8	0
8	03/06/2024 09:00:00	690	20,2	20,3	28,9	14,5	0	0	40	0	990,8	0
9	03/06/2024 09:10:00	679	20,2	20,3	28,9	14,5	0	0	40	0	990,8	0
10	03/06/2024 09:20:00	683	20,3	21	28,9	14,5	0	0	40	2	990,6	0
11	03/06/2024 09:30:00	705	20,4	21,5	28,9	14,5	0	0	40	1	990,6	0
12	03/06/2024 09:40:00	710	20,4	21,3	28,9	14,5	0	0	40,5	1	990,6	0
13	03/06/2024 09:50:00	694	20,5	22,6	28,9	14,5	0	0	40,5	1	990,5	0
14	03/06/2024 10:00:00	679	20,6	23,1	28,9	14,5	0	0	40,5	1	990,4	0
15	03/06/2024 10:10:00	781	20,8	23	28,9	14,5	0	0	41	1	990,2	0
16	03/06/2024 10:20:00	915	21	23,1	28,9	14,5	0	0	42	1	990,2	0
17	03/06/2024 10:30:00	977	21,3	23,8	28,9	14,5	0	0	42	1	990	0
18	03/06/2024 10:40:00	1067	21,5	24,6	28,9	14,5	0	0	42,5	1	989,8	0
19	03/06/2024 10:50:00	1147	21,6	24,2	28,9	14,5	0	0	43,5	1	989,6	0
20	03/06/2024 11:00:00	1231	21,8	24,4	28,9	14,5	0	0	43,5	1	989,5	0

Figura 11. Ejemplo de un dataset con todas las variables

3 Problemática en el acondicionamiento de datos

3.1 Acondicionamiento de datos

La generación automática de *datasets* a partir de datos de un entorno *smart campus* presenta una serie de desafíos técnicos y conceptuales que deben entenderse con detalle antes de plantear cualquier solución. Una vez obtenidas todas las variables de interés, hay que homogeneizar estas variables para poder unificarlas en un único archivo final. Para ello, se ha creado en un *script* de Python, llamado `procesar_datos.py`, que es un *pipeline* modular que realiza las siguientes funciones:

- Ingesta de datos de múltiples sensores.
- Limpieza y alineación automática de índices y NaN.
- Normalización semántica de variables.
- Enriquecimiento con datos externos de AEMET, ocupación, calefacción.
- Exportación ordenada en el formato deseado.

Un esquema secuencial de funcionamiento se muestra en la figura 12. Gracias a este pipeline, se consiguen solucionar las problemáticas que han ido surgiendo y que se detallan a continuación:

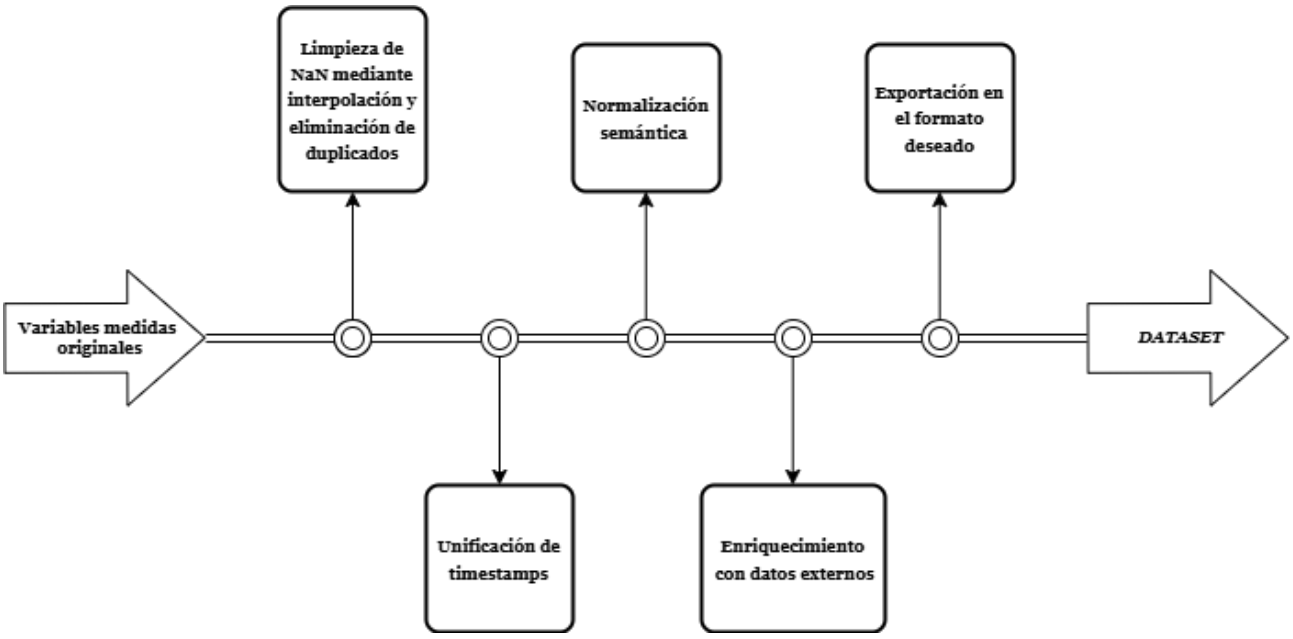


Figura 12. Ejemplo del pipeline seguido en el acondicionamiento de datos

3.1.1 Valores ausentes y huecos temporales (NaN)

Los sensores desplegados en el campus (medidores de CO_2, temperatura, humedad, contadores de aforo, luminancia, entre otros) no garantizan la entrega continua de información. En muchas ocasiones, debido a cortes de conexión, reinicios de dispositivos o errores de transmisión, aparecen secuencias de lecturas vacías (*NaN*) de duración variable, (ver figura 13). Estos vacíos comprometen la integridad de las series temporales y dificultan el posterior análisis estadístico. Hay dos tipos de vacíos, interrupciones breves, de minutos u horas, generan discontinuidades que impiden un normal desarrollo. En estos casos, se ha decidido realizar una interpolación lineal si hay un máximo de 5 datos consecutivos entre datos, lo que sería 1 hora entre datos. Hay otro tipo de vacíos que son más complicados de solucionar, que son las ausencias prolongadas, que pueden durar días completos, obligan a decidir si se descarta el intervalo por falta de fiabilidad o se interpolan valores basados en datos adyacentes. En nuestro caso se ha mantenido el mismo valor para que no falten datos en el fichero final de exportación, al final no se pueden inventar datos que faltan, pero se pueden rellenar para que al menos no estén vacíos.

	A	B
1	Time	tmp_indoor
2	05/07/2023 17:00:00	21,85
3	05/07/2023 17:10:00	21,85
4	05/07/2023 17:20:00	21,9
5	05/07/2023 17:30:00	22,3
6	05/07/2023 17:40:00	22,35
7	05/07/2023 18:00:00	NaN
8	05/07/2023 18:10:00	NaN
9	05/07/2023 18:20:00	NaN
10	05/07/2023 18:30:00	23,1
11	05/07/2023 18:40:00	23,15
12	05/07/2023 18:50:00	23,25
13	05/07/2023 19:00:00	23,3

(a) Fichero ejemplo con NaN

	A	B
1	Time	tmp_indoor
2	05/07/2023 17:00:00	21,85
3	05/07/2023 17:10:00	21,85
4	05/07/2023 17:20:00	21,9
5	05/07/2023 17:30:00	22,3
6	05/07/2023 17:40:00	22,35
7	05/07/2023 18:00:00	22,7
8	05/07/2023 18:10:00	22,9
9	05/07/2023 18:20:00	22,95
10	05/07/2023 18:30:00	23,1
11	05/07/2023 18:40:00	23,15
12	05/07/2023 18:50:00	23,25
13	05/07/2023 19:00:00	23,3

(b) Fichero ejemplo tras interpolación

Figura 13. Proceso interpolación ejemplo

Se ha intentado ayudar al usuario añadiendo en la carpeta final de exportación (como se puede ver en la Figura 14) dos ficheros de texto, que se explicarán más adelante en el capítulo 5. Como anticipo, indicar que el fichero *quality* incluye el porcentaje de valores reales de las variables para el periodo de tiempo seleccionado, teniendo en cuenta solo los datos reales, sin tener en cuenta *NaN* ni duplicados. Y el fichero *days* indica, para cada día dentro del rango seleccionado de fechas, el porcentaje de datos que hay, ayudando así a seleccionar el mejor intervalo de tiempo.

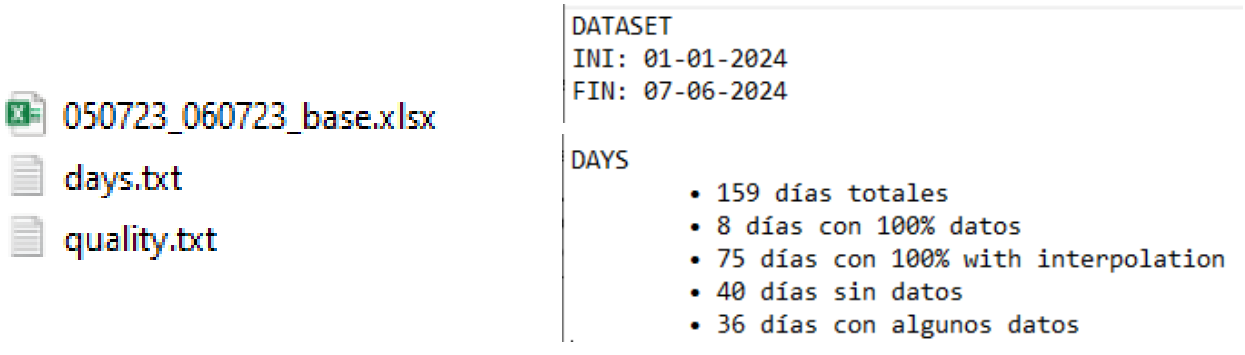


Figura 14. Archivos dentro de la carpeta final de exportación

3.1.2 Desalineación temporal de *timestamps*

Al tener datos de múltiples sensores, además de datos provenientes de fuentes externas, se produce una desalineación temporal entre todos los *timestamps* de las diferentes variables que hay que soluciona y homogeneizar en un único *timestamp* global para poder extraer todas las variables en un archivo final. Cada tipo de sensor opera con su propia cadencia de muestreo:

- Sensores ambientales (temperatura, CO_2, humedad) envían datos cada 10 minutos aproximadamente, pero no todos los sensores los envían en los mismos minutos, ni a una frecuencia exacta. Por ejemplo, un dato de temperatura tomado 02/06/2024 0:07 se redondea a 02/06/2024 0:00. Con esto se consigue que haya 6 datos a la hora, si se escoge la interpolación cada 10 minutos.
- Estación meteorológica, en este caso envía los datos cada 5 minutos así que lo que se hace es hacer una media entre los dos datos y poner uno cada 10 minutos si es el intervalo escogido.
- APIs externas, AEMET devuelve un dato al día, el cual hay que expandir durante todo el día según el intervalo de interpolación seleccionado.

La solución es unificar las frecuencias y crear un rango de tiempo común con el intervalo de interpolación deseado entre 1 y 60, todos los orígenes de datos quedan obligados a ese calendario. Además, se eliminan *timestamps* duplicados. En las siguientes figuras 15, 16 y 17, se puede ver un ejemplo de cómo se unifican en un mismo *timestamp* variables que originalmente han sido recibidas en tiempos distintos y con frecuencias distintas, las variables `co2` y `tmp_indoor` se reciben cada 10 minutos y `tmp_meteo` cada 5.

	A	B	C
1	Time	co2	tmp_indoor
2	02/06/2024 0:07	570	18,55
3	02/06/2024 0:17	563	18,5
4	02/06/2024 0:27	569	18,45
5	02/06/2024 0:37	564	18,45
6	02/06/2024 0:47	559	18,4
7	02/06/2024 0:57	554,5	18,375
8	02/06/2024 1:07	550	18,35
9	02/06/2024 1:17	554	18,3
10	02/06/2024 1:27	549	18,25
11	02/06/2024 1:37	555	18,225

(a) raw tmp_meteo

	A	B
1	Time	tmp_meteo
2	02/06/2024 0:04	16.3
3	02/06/2024 0:09	16.3
4	02/06/2024 0:14	16.2
5	02/06/2024 0:19	16.1
6	02/06/2024 0:24	16.1
7	02/06/2024 0:30	16.1
8	02/06/2024 0:35	16.1
9	02/06/2024 0:40	15.9
10	02/06/2024 0:45	15.9
11	02/06/2024 0:50	15.8
12	02/06/2024 0:55	15.7
13	02/06/2024 1:00	15.7

(b) raw co2 y tmp_indoor

	A	B	C	D
1	Time	co2	tmp_indoor	tmp_meteo
2	02/06/2024 00:00:00	570	18,55	16,3
3	02/06/2024 00:10:00	563	18,5	16,3
4	02/06/2024 00:20:00	569	18,45	16,1
5	02/06/2024 00:30:00	564	18,45	16,1
6	02/06/2024 00:40:00	559	18,4	16,1
7	02/06/2024 00:50:00	554,5	18,375	15,8
8	02/06/2024 01:00:00	550	18,35	15,7
9	02/06/2024 01:10:00	554	18,3	15,6
10	02/06/2024 01:20:00	549	18,25	15,6
11	02/06/2024 01:30:00	555	18,225	15,4
12	02/06/2024 01:40:00	561	18,2	15,3
13	02/06/2024 01:50:00	542	18,15	15,3

(c) dataset con timestamp unificado

Figura 15. Ejemplo de datos alineados temporalmente

3.1.3 Heterogeneidad semántica y de formatos:

Los fabricantes de dispositivos IoT no emplean un estándar unificado para nombrar parámetros ni para indicar unidades de medida. Entre las lecturas recogidas se observan variantes de cada variable. Este problema tiene una fácil solución, y es crear un diccionario (se puede ver en la figura 16), en el que se cambia el nombre asignado por los sensores a un nombre elegido según un formato estipulado en el capítulo 2. Tras la unificación semántica, se apilan en orden (`base`, `ext` y `adv`) según tus listas `base_keys_app`, `ext_keys_app` y `adv_keys_app`. Con ello las variables quedan siempre en la misma disposición

```
# Mapeo de nombres "crudos" a los nombres que aparecen en la interfaz
raw_to_app = {
    "temperature":       "tmp_indoor",
    "lightLevel":        "light",
    "atmosphericPressure":"pressure",
    "temperatureExterior": "tmp_meteo",
    # Suponiendo que el resto de sensores (co2, humidity, etc.)
    # ya se llaman igual en raw y en la app. Si hubiera más casos, añadelos aquí.
}
```

Figura 16. Ejemplo de diccionario de datos

3.1.4 Detección y tratamiento de anomalías

Además de los valores ausentes, los *datasets* incluyen lecturas erróneas o atípicas: picos de CO_2 superiores a 5000 ppm, temperaturas inferiores a 0°C en verano o humedades del 0%. Estas anomalías pueden deberse a fallos de calibración o a situaciones reales transitorias (ventanas abiertas, cambios de climatización). Para solucionar esta problemática, se utiliza un reindexado *nearest* que atenúa picos aislados al suavizar con la lectura más próxima. Este procesado según cada caso incluye:

- **`co2`**. Se interpola linealmente y se eliminan valores fuera de rango (400 – 1200 ppm) para asegurar la precisión y mantener la calidad de los datos.
- **`humidity`.** Se interpola linealmente para asegurar la precisión y no se elimina ningún valor ya que es un porcentaje, y así se mantiene la calidad de los datos.
- **`pressure`.** Se interpola linealmente y se eliminan valores fuera de rango (400 – 1200 hPa) para asegurar la precisión y mantener la calidad de los datos.
- **`light`.** Se interpola linealmente y se eliminan valores superiores a 40000 lux, para asegurar la precisión y mantener la calidad de los datos.
- **`tmp_indoor`, `tmp_meteo`, `tmp_aemet.max`, `tmp_aemet.min`**. Se rellenan valores faltantes de forma lineal y se eliminan valores fuera de rango (-10ºC – 40ºC) para asegurar la precisión y mantener la calidad de los datos.
- **`hvac_boolean`, `occ_boolean`**. Se rellenan los valores faltantes con el valor posterior para mantener la consistencia temporal.
- **`occ_people`.** Esta variable se crea a partir de la `occ_boolean`, una vez que se ha procesado.

En conclusión, gracias a la separación clara en pasos, cada problemática encuentra su contrapartida en un bloque de código dedicado, lo que hace la solución mantenible y ampliable.

4 Aplicativo APP Python

La aplicación desarrollada tiene como objetivo facilitar la consulta, procesamiento y exportación de una forma automatizada de los datos recogidos por sensores IoT instalados en diferentes aulas universitarias. Está pensada para que usuarios no expertos en programación puedan extraer datos limpios, completos y listos para análisis con tan solo unos clics. Antes de empezar a programar con Python, hay que crear un diseño que ha tenido varias fases antes de llegar a la aplicación final.

4.1 Metodología de trabajo

El desarrollo de este TFG ha seguido un enfoque iterativo y orientado a objetivos prácticos, partiendo de la necesidad real de disponer de una herramienta que automatizara por completo la generación de *datasets* a partir de sensores IoT en entornos universitarios. En la fase inicial se realizó un análisis de requisitos a partir de proyectos previos en Matlab, identificando los puntos débiles de las soluciones existentes (procesos manuales de limpieza, intervalos fijos y falta de modularidad). A continuación, se elaboró un plan de trabajo dividido en hitos semanales:

- **Revisión de soluciones anteriores:**
  - Identificación de los *scripts* y *pipelines* de Matlab utilizados en cursos anteriores.
  - Detección de cuellos de botella (poca flexibilidad en la definición de intervalos, dificultad para incorporar nuevas variables).
- **Definición de arquitectura modular:**
  - Especificación de las funciones principales (ingesta, limpieza, enriquecimiento, exportación).
  - Diseño de un esquema de *keys (base, external, advanced*) que permitiera escalar a futuro.
- **Implementación y validación de cada módulo:**
  - Desarrollo incremental en Python de cada bloque de procesamiento, con pruebas unitarias y validación estadística de resultados.
  - Comparativa de salidas contra los casos de prueba originales en Matlab para garantizar consistencia.
- **Integración y pruebas de usuario**
  - Creación de la interfaz PyQt5 y su vinculación con el pipeline de datos.
  - Pruebas funcionales para pulir la usabilidad.

Gracias a esta metodología, se ha conseguido una herramienta robusta, flexible y mantenible, que no solo automatiza tareas repetitivas, sino que sienta las bases para futuras extensiones como nuevos sensores, formatos de exportación o integraciones con servicios en la nube.

4.2 Diseño del aplicativo

En un primer momento, el prototipo se desarrolló en Matlab, siguiendo la línea de trabajo de proyectos anteriores liderados desde entornos académicos. El diseño anterior (ver figura 18), presentaba varias funcionalidades que, con el *feedback* de compañeros anteriores, se decidió que no tenían mucha utilidad, incluso había varias que se solapaban. Al final, cada persona había añadido lo que necesitaba a la aplicación, siendo así poco homogéneo, y por eso en versiones posteriores se decidieron eliminar. El primer paso fue realizar un diseño esquemático de la interfaz gráfica, se decidió apostar por un sistema de cajas que facilita la creación de diseños coherentes y adaptables: cada caja actúa como bloque reutilizable, lo que acelera el desarrollo y evita estilos dispersos, como se puede apreciar en la figura 17.

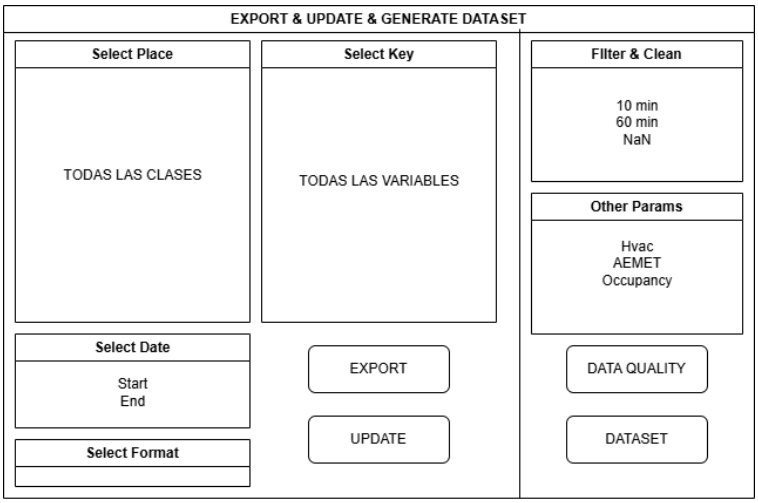


Figura 17. Diseño esquemático de la primera versión IoB APP Matlab

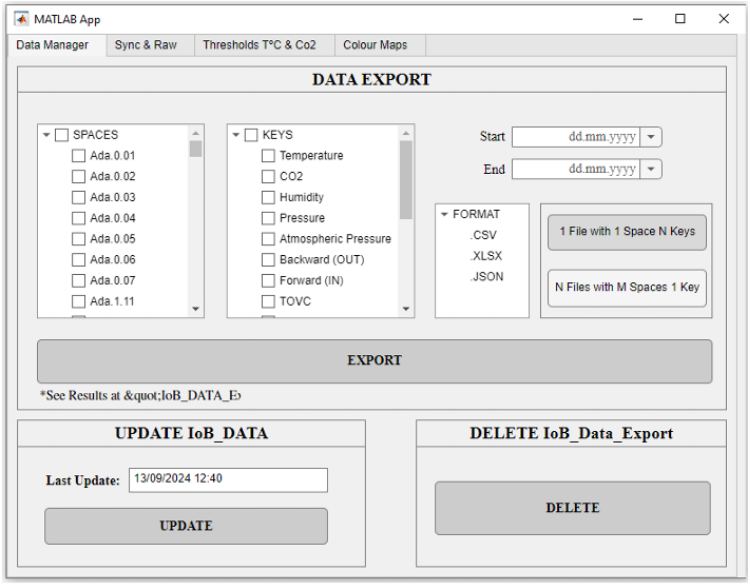


Figura 18. IoB APP Matlab

Como se puede ver en la figura 17, desde el primer momento se decidió añadir la selección de un intervalo unificado, además de añadir las variables provenientes de fuera de los sensores. También se iban reutilizando ideas que ya estaban diseñadas en la app Matlab, como la selección de fechas mediante un calendario, para hacerlo más intuitivo.

Posteriormente se diseñó un segundo prototipo en el entorno Matlab (ver figura 19), qué seguía en la línea estética de la primera versión, pero añadiendo nuevas funcionalidades, siguiendo el dibujo esquemático.

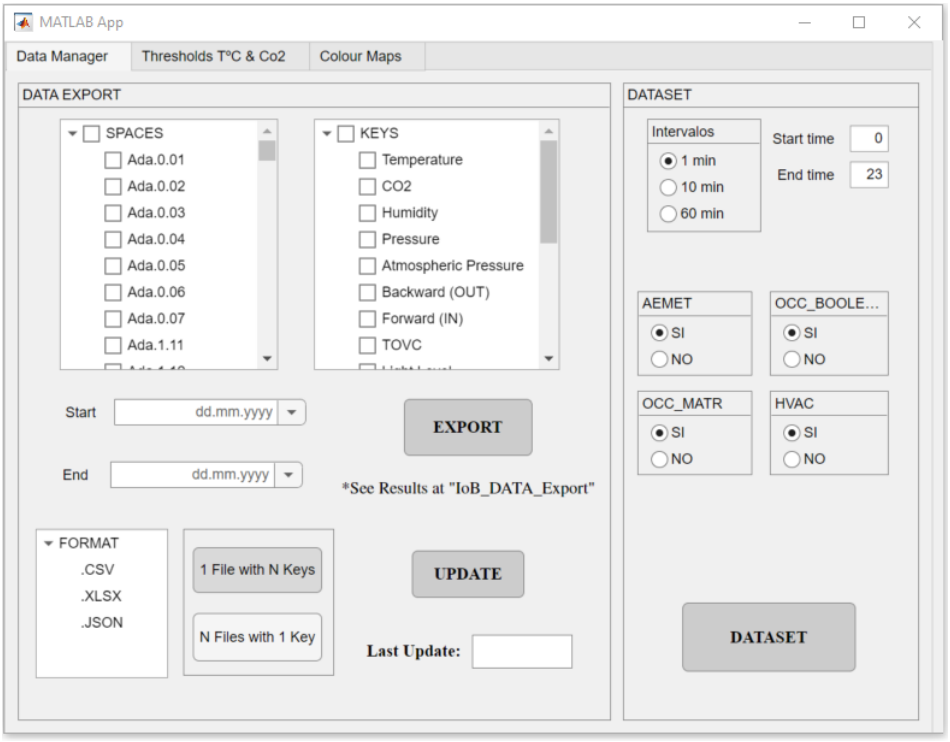


Figura 19. IoB APP Matlab segunda versión

Matlab aportaba varias ventajas como la rapidez para prototipar funciones de limpieza y visualización, pero mostró limitaciones crecientes:

- **Rigidez en la GUI**: la generación de interfaces de usuario era más limitada y menos modular, Matlab propone poca variación en el *frontend*, limitando así la parte visual.
- **Lenguaje de programación propio**: toda la parte del *backend* se tenía que desarrollar con un lenguaje único, no tan global como Python.
- **Disponibilidad de bibliotecas**: aunque Matlab tiene *toolboxes* para muchas áreas, Python dispone de un repositorio unificado (PyPI) con decenas de miles de paquetes: *pandas*, *numpy*, *scikit-learn*, *tensorflow*, *requests*, *plotly*, entre otros.

Por estas razones, se decidió rediseñar la aplicación desde cero en Python, creando en primer lugar un esquema (ver figura 20), siguiendo con un diseño visual parecido al que ya había en Matlab, pero aprovechando las ventajas que ofrece este lenguaje de programación.

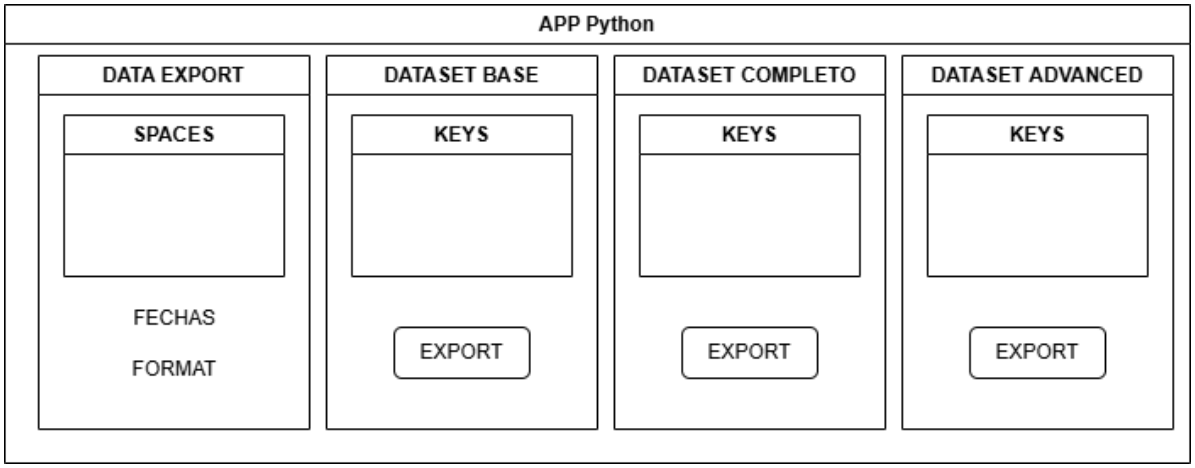


Figura 20. Diseño esquemático aplicativo Python

Las principales ventajas que presenta Python respecto a Matlab son las siguientes:

- **Ecosistema abierto y específico.** Usamos pandas para todo el procesamiento de series temporales y *requests* para llamar a la API de AEMET, algo que en Matlab implicaría *toolboxes* adicionales. La GUI con PyQt5 permitió diseñar rápidamente la interfaz sin tener que adaptar ventanas predefinidas de Matlab.
- **Código modular y fácil de ampliar.** Todo el pipeline de datos vive en procesar_datos.py, mientras que la interfaz está en MiApp (PyQt5), de forma que añadir un nuevo sensor solo requiere actualizar el diccionario y no tocar la vista. Los mapeos abstraen los nombres crudos de los sensores, garantizando que, al crecer el número de variables, la interfaz no sufra cambios drásticos.
- **Interfaz fluida y escalable.**
- **Mejor experiencia de usuario.** La interfaz visual tiene un aspecto coherente y moderno. Además, las validaciones en cada paso (fechas, selección de *keys*, carpeta de salida) evitan errores antes de que arranque la exportación de datos

4.3 *Frontend*

El *frontend* es la parte de la aplicación que ve y usa el usuario final. Incluye todo lo que se muestra en pantalla: botones, menús, formularios, gráficos, etc. El diseño de esta interfaz prioriza la claridad visual y la interacción intuitiva, estructurando las opciones de forma lógica para guiar al usuario en el proceso. Este *frontend* está construido en Python, como se puede ver en la figura 21, íntegramente con PyQt5, aprovechando su modelo de widgets para separar claramente la lógica de interfaz de las demás lógicas. Cada componente (selector de aulas, intervalo, formato, fechas/hora y listas de *keys*) se encapsula en su propio QVBoxLayout o QHBoxLayout, y los diálogos modales (como SelectorAulas) están implementados como subclases de QDialog. Además, el procesamiento pesado de exportación se delega a un hilo de trabajo (ExportWorker, derivado de QThread) que emite señales de progress, finished y error; de este modo la interfaz permanece receptiva y actualiza dinámicamente la QProgressDialog y la QProgressBar.

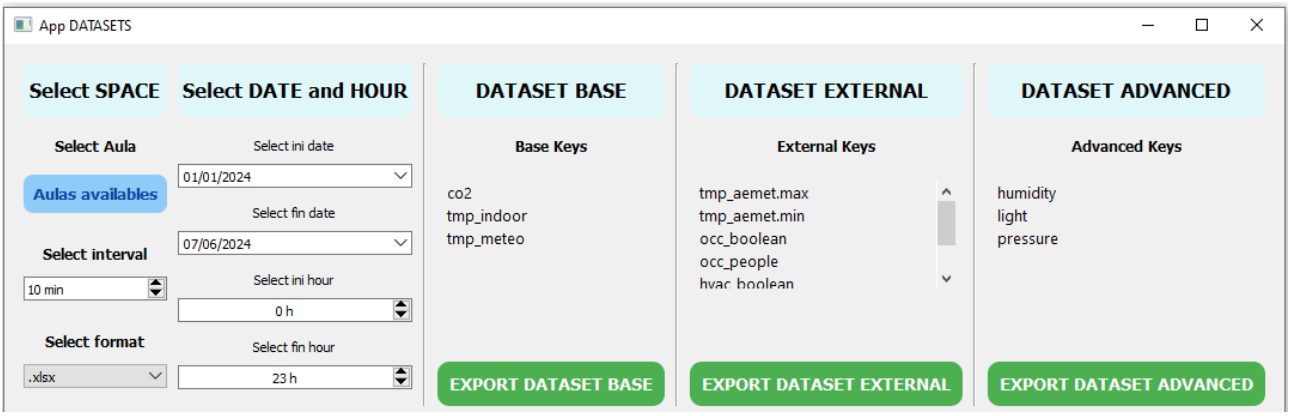


Figura 21. Frontend aplicativo Python

En primer lugar, puede verse que la interfaz se separa en 5 columnas para conseguir mantener la proporción y que quede cuadriculada. A continuación, se explica la funcionalidad de cada parte.

- **Selección de aulas:** Esta parte se ha realizado con un desplegable de espacios (ver figura 22), al tener una multitud de aulas diferentes. En este ejemplo se muestran aulas del edificio Ada Byron y del Betancourt, que son los edificios donde había mayor cantidad de sensores y de datos. Este paso define los directorios de donde se leerán los datos. Además, ya que en algunas aulas hay más sensores que en otras, al seleccionar un aula, sale una pantalla en la que se muestra las variables disponibles para esa aula, como muestra la figura 23.

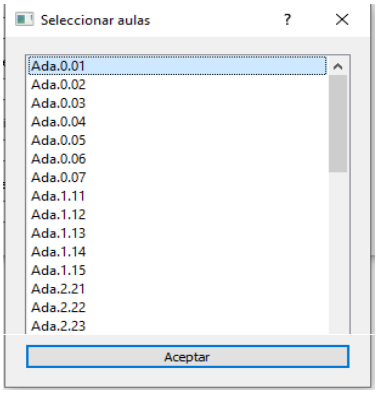


Figura 22. Interfaz selección aulas

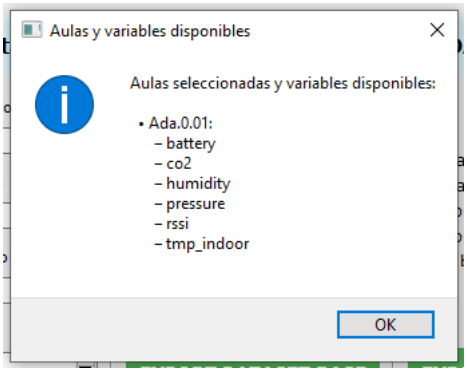


Figura 23. Ejemplo de variables disponibles en un aula

- **Selección de intervalo:** Escuchando el *feedback* de otros compañeros que habían trabajado con la app Matlab anterior, decían que una funcionalidad que se podría haber implementado es poder obtener los datos finales en diferentes intervalos, no solo cada 10 minutos, ya que hay estudios para los que es mejor tener datos cada minuto o un dato cada hora, por eso se introdujo el siguiente menú, con el que se puede elegir cualquier intervalo entre 0 y 60 minutos, aunque para obtener los datos más precisos es mejor elegir cada 10 minutos, como muestra la 24.

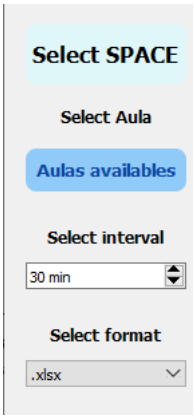


Figura 24. Interfaz selección intervalo

- **Selección de formato:** La aplicación permite seleccionar un formato de exportación (*.xlsx*, *.csv* y *.json*) para dar mayor versatilidad. Cada formato es útil para una función distinta: *xlsx* es el más usado e intuitivo, *csv* es más ligero y rápido de procesar, mientras que *json* es ideal para integraciones con APIs o aplicaciones web, como muestra la 25.

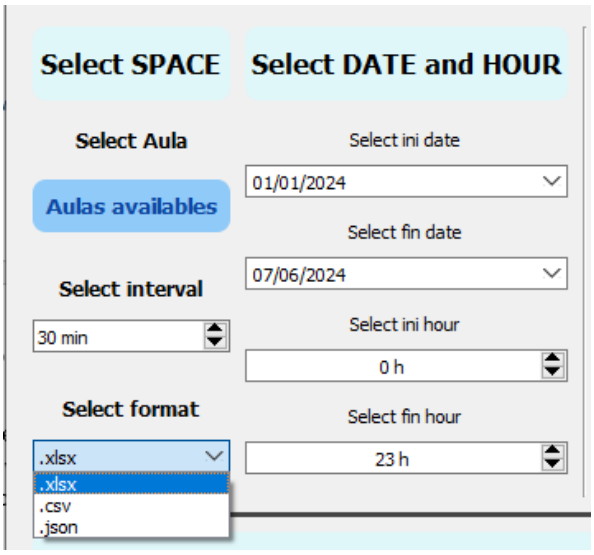


Figura 25. Interfaz selección formato

- **Selección de fechas:** Está compuesto de un selector de fecha de inicio y otro de fecha final, ver figura 26, permite al usuario filtrar por un rango de fechas con una interfaz intuitiva, facilita el posterior análisis. Si la fecha final es anterior a la fecha inicial, saca un aviso por pantalla antes de exportar, como muestra la figura 27.

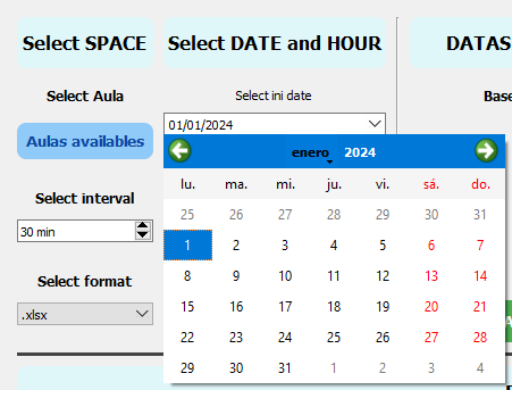


Figura 26. Interfaz selección fechas

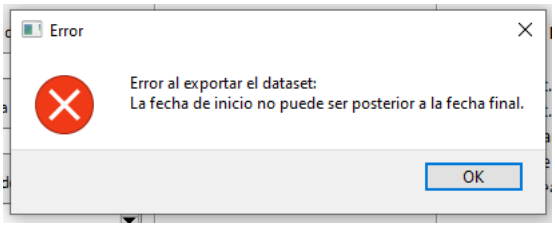


Figura 27. Aviso error exportación

- **Selección de horas:** Hay un selector de hora de inicio y hora final, para poder acotar aún más los datos, permite realizar un análisis más preciso, ya que se puede seleccionar por ejemplo solo las horas en las que está abierto un edificio, como el Ada Byron, y eliminar así las horas de noche que son menos importantes a la hora de realizar un análisis, como muestra la figura 28.

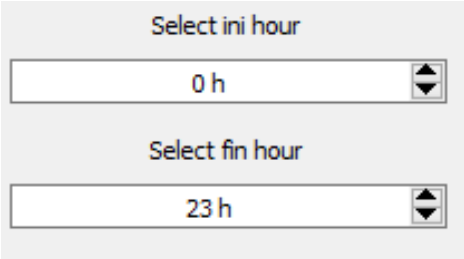


Figura 28. Interfaz selección hora inicio y final

Generación de *datasets*. Las variables se organizan en tres grandes grupos*: base keys, external keys y advanced keys*. Esta clasificación tiene como objetivo estructurar el tratamiento y visualización de los datos según su origen, relevancia y complejidad.

- ***Dataset Base*** (ver figura 29). Contiene un *dataset* con las *keys base*, las variables más importantes, las principales, que se consideran mínimas y fundamentales para cualquier análisis en el contexto de calidad del aire o confort ambiental.
- ***Dataset External*** (ver figura 30). Contiene las *keys externas*, que son variables contextuales o complementarias que no provienen de sensores físicos del aula sino de fuentes externas o cálculos adicionales. Complementan directamente las *keys base*.
- ***Dataset Advanced*** (ver figura 31). Contiene las variables secundarias o técnicas que complementan los análisis más detallados o avanzados, son útiles, pero no esenciales.

	A	B	C	D
1	Time	co2	tmp_indoor	tmp_meteo
2	03/06/2024 00:00:00	463	17,2	16,6
3	03/06/2024 00:10:00	446	17,2	16,6
4	03/06/2024 00:20:00	465	17,15	16,5
5	03/06/2024 00:30:00	456	17,15	16,5
6	03/06/2024 00:40:00	462	17,15	16,3
7	03/06/2024 00:50:00	468	17,15	16,2
8	03/06/2024 01:00:00	460	17,1	16,1
9	03/06/2024 01:10:00	453	17,1	16
10	03/06/2024 01:20:00	456	17,05	15,9
11	03/06/2024 01:30:00	465	17,05	15,8
12	03/06/2024 01:40:00	460	17,05	15,8
13	03/06/2024 01:50:00	467	17,05	15,7
14	03/06/2024 02:00:00	459	17	15,6

Figura 29. Ejemplo dataset base

	A	B	C	D	E	F	G
1	Time	tmp_aemet.max	tmp_aemet.min	occ_boolean	occ_people	hvac_boolean	calendar_categories
2	03/06/2024 08:00:00	28,9	14,5	0	0	0	9
3	03/06/2024 08:10:00	28,9	14,5	0	0	0	9
4	03/06/2024 08:20:00	28,9	14,5	0	0	0	9
5	03/06/2024 08:30:00	28,9	14,5	0	0	0	9
6	03/06/2024 08:40:00	28,9	14,5	0	0	0	9
7	03/06/2024 08:50:00	28,9	14,5	0	0	0	9
8	03/06/2024 09:00:00	28,9	14,5	0	0	0	9
9	03/06/2024 09:10:00	28,9	14,5	0	0	0	9
10	03/06/2024 09:20:00	28,9	14,5	0	0	0	9
11	03/06/2024 09:30:00	28,9	14,5	0	0	0	9
12	03/06/2024 09:40:00	28,9	14,5	0	0	0	9
13	03/06/2024 09:50:00	28,9	14,5	0	0	0	9
14	03/06/2024 10:00:00	28,9	14,5	0	0	0	9
15	03/06/2024 10:10:00	28,9	14,5	0	0	0	9
16	03/06/2024 10:20:00	28,9	14,5	0	0	0	9
17	03/06/2024 10:30:00	28,9	14,5	0	0	0	9
18	03/06/2024 10:40:00	28,9	14,5	0	0	0	9
19	03/06/2024 10:50:00	28,9	14,5	0	0	0	9
20	03/06/2024 11:00:00	28,9	14,5	0	0	0	9

Figura 30. Ejemplo dataset external

	A	B	C	D
1	Time	humidity	light	pressure
2	03/06/2024 00:10:00	39	0	993,7
3	03/06/2024 00:30:00	39	0	993,8
4	03/06/2024 00:40:00	39	0	993,6
5	03/06/2024 00:50:00	39	0	993,5
6	03/06/2024 01:00:00	39	0	993,5
7	03/06/2024 01:10:00	39	0	993,5
8	03/06/2024 01:20:00	39	0	993,5
9	03/06/2024 01:30:00	39	0	993,4
10	03/06/2024 01:40:00	39	0	993,1
11	03/06/2024 01:50:00	39	0	993,1
12	03/06/2024 02:00:00	39	0	993,2
13	03/06/2024 02:10:00	39	0	993,1
14	03/06/2024 02:20:00	39	0	993
15	03/06/2024 02:30:00	39	0	992,9
16	03/06/2024 02:40:00	39	0	992,6
17	03/06/2024 02:50:00	39	0	992,3
18	03/06/2024 03:00:00	39	0	992,5

Figura 31. Ejemplo dataset advanced

Si se pulsa el botón *export dataset base*, se exportan solo las variables seleccionadas dentro de las variables base. Si se pulsa el botón *export dataset external,* se exportan las variables seleccionadas de las variables base y las externas. Finalmente, si se *presiona export dataset advanced*, se exportan las variables seleccionadas de todas las *keys*. Se puede ver este interfaz en la figura 32. Además, a la hora de exportar los datos, se puede seleccionar el directorio que se quiera, para tener más libertad y control sobre los datos, ya no hace falta una carpeta configurada de antemano, como se puede ver en la figura 33(a). El fichero generado se nombra concatenando fecha de inicio, fecha de final y el tipo de *dataset* como: *ddmmyy_ddmmyy_base/external/advanced*, ver Figura 33(b).

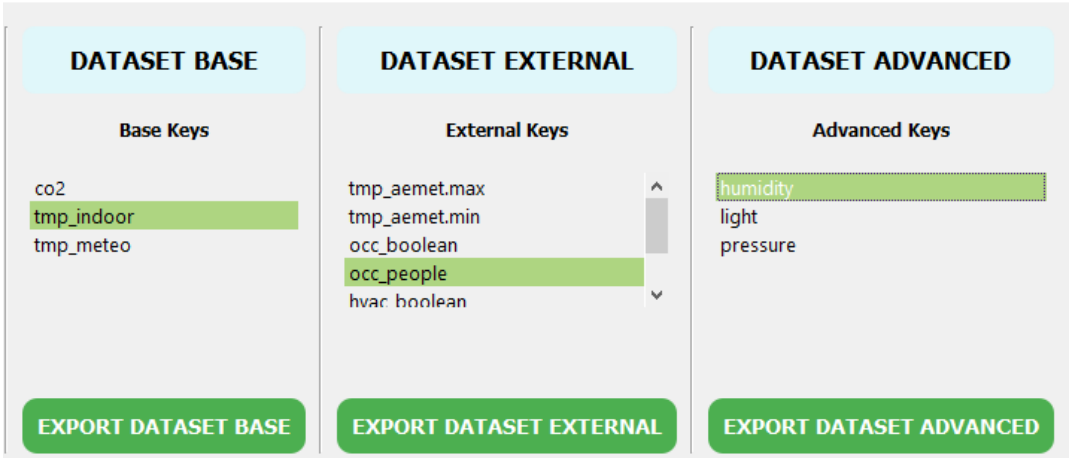
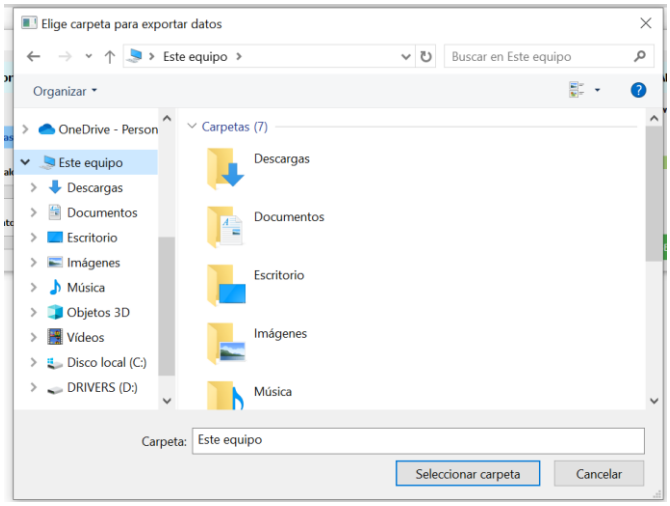
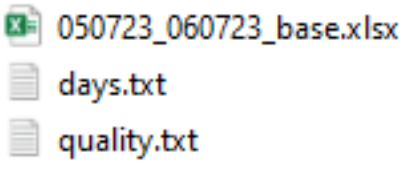


Figura 32. Interfaz selección keys





(a) Interfaz seleccionar directorio de exportación *(b) Archivos dentro del directorio de exportación*

Figura 33. Interfaz directorio de exportación

Además, se incluye una barra de progreso una vez se están exportando los datos. Esto aporta una mayor ayuda visual, a pesar de que los datos se exportan en menos de 5 segundos (ver figura 34). Y también saca por pantalla los tres intervalos de tiempo dentro del rango seleccionado para los que hay un mejor porcentaje de datos, como se ve en la figura 35. Esta información también se incluye en el fichero *days.txt*, como se detalla en el capítulo 5. Finalmente, cuando ya se han exportado los datos correctamente, aparece un mensaje de confirmación por pantalla, como se ve en la figura 36

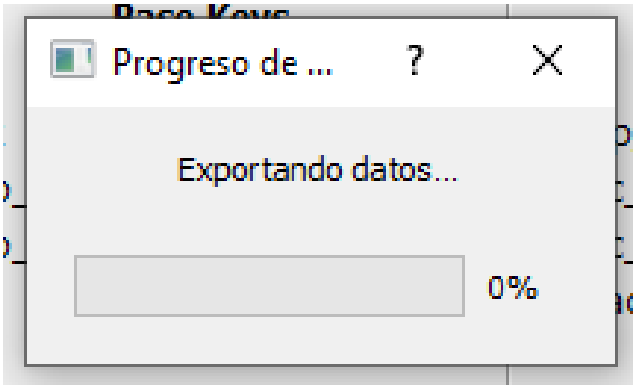


Figura 34. Barra de progreso de exportación de datos

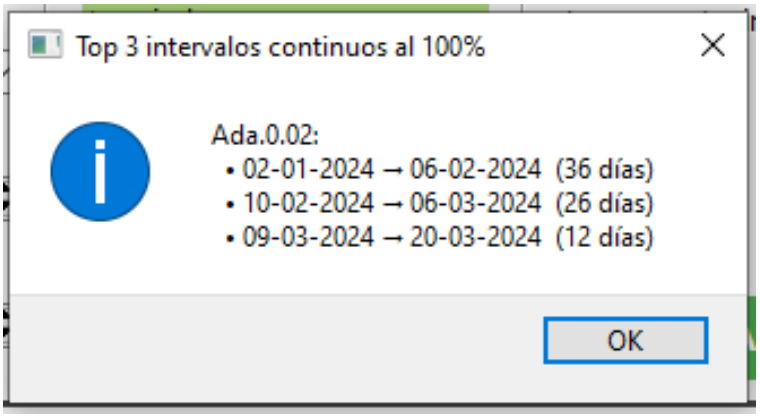


Figura 35. Ejemplo tres mejores intervalos de datos

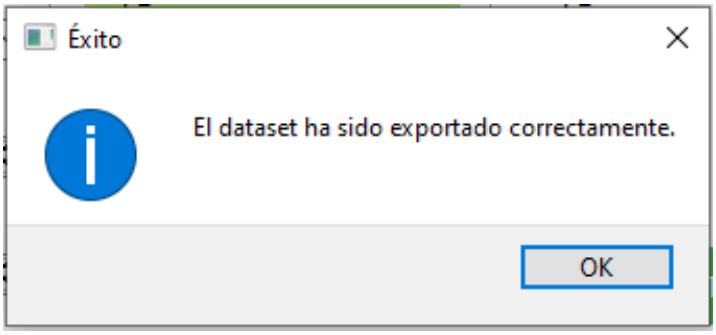


Figura 36. Mensaje de exportación correcta

Gracias al *feedback* recibido de otros compañeros asociados a proyectos en este entorno, se ha implementado una funcionalidad extra, llamada *dataset aggregated* (ver figura 37). Se utiliza para poder añadir variables extras a los *datasets* finales: por ejemplo, al generar modelos de predicción, se necesitan variables como la temperatura o el CO_2 una hora antes o después. Con esta funcionalidad, puedes juntar dos excels con la misma longitud de filas en uno solo, quedando como resultado final un Excel con una columna de tiempos y el resto de las variables, se pueden concatenar tantas filas de variables como el usuario quiera.

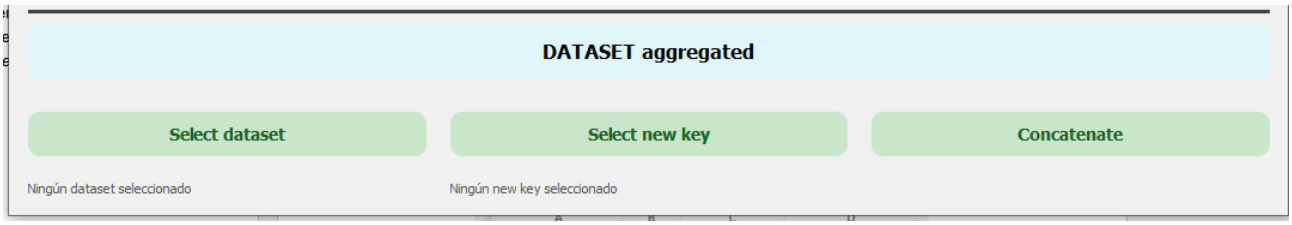


Figura 37. Interfaz DATASET aggregated

El funcionamiento es el siguiente. En primer lugar, cuando se pulsa el botón *seleccionar dataset,* se abre una ventana para escoger un Excel de cualquier directorio, con total libertad. Una vez se selecciona, sale por pantalla, debajo del botón, para que no haya errores, como se ve en la figura 38. Después, pulsando el botón *selecciona new key*, se puede seleccionar otro Excel para finalmente concatenar amabas Excel. Tiene el mismo funcionamiento que el botón *seleccionar dataset*, como se ve en la figura 39.

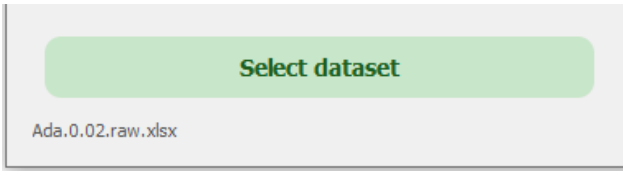


Figura 38. Ejemplo dataset seleccionado

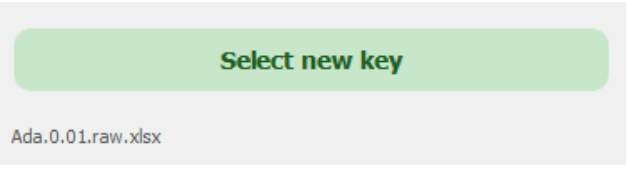


Figura 39. Ejemplo new key seleccionada

Para finalizar, ya solo quedaría seleccionar el botón *concatenate*, con el que se unen ambos Excels en uno. Si no tienen el mismo número de filas y de *timestamps*, salta un aviso por pantalla (véase en la figura 40). Esto es así porque esta funcionalidad se ha diseñado principalmente para que: primero, se cree un *dataset*; y después (utilizando un modelo predictivo o algoritmo), se cree una nueva variable que no se puede seleccionar directamente en ninguna de las *keys*; así, se puede concatenar y tener todo en un mismo archivo.

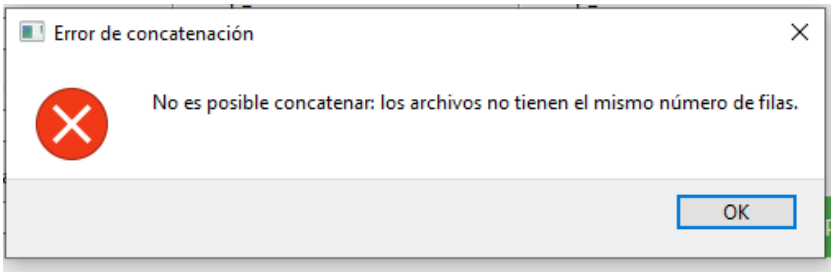


Figura 40. Aviso error al concatenar

A continuación, se ha detallado esta explicación con un ejemplo (ver figura 43), siendo el *dataset* principal el visto en la figura 41. Se le quiere añadir una variable alternativa llamada *tmp_indor_after1h,* véase en la figura 42. Para ello se utiliza esta funcionalidad, y quedan todas las variables juntas y alineadas en un único archivo final, como se ve en la figura 43.

	A	B	C
1	Time	co2	tmp_indoor
2	01/01/2024 20:10:00	439	12,95
3	01/01/2024 20:20:00	455	13
4	01/01/2024 20:30:00	443	12,95
5	01/01/2024 20:40:00	447	12,9
6	01/01/2024 20:50:00	444	12,9
7	01/01/2024 21:00:00	448	12,85
8	01/01/2024 21:10:00	456	12,85
9	01/01/2024 21:20:00	452	12,8
10	01/01/2024 21:30:00	440	12,8
11	01/01/2024 21:40:00	432	12,75
12	01/01/2024 21:50:00	438	12,75

Figura 41. Ejemplo dataset

	A	B
1	Time	tmp_indoor_after1h
2	01/01/2024 20:10:00	13
3	01/01/2024 20:20:00	12,95
4	01/01/2024 20:30:00	12,85
5	01/01/2024 20:40:00	12,85
6	01/01/2024 20:50:00	12,8
7	01/01/2024 21:00:00	12,8
8	01/01/2024 21:10:00	12,8
9	01/01/2024 21:20:00	12,85
10	01/01/2024 21:30:00	12,85
11	01/01/2024 21:40:00	12,9
12	01/01/2024 21:50:00	12,9

Figura 42. Ejemplo new key

	A	B	C	D
1	Time	co2	tmp_indoor	tmp_indoor_after1h
2	01/01/2024 20:10:00	439	12,95	13
3	01/01/2024 20:20:00	455	13	12,95
4	01/01/2024 20:30:00	443	12,95	12,85
5	01/01/2024 20:40:00	447	12,9	12,85
6	01/01/2024 20:50:00	444	12,9	12,8
7	01/01/2024 21:00:00	448	12,85	12,8
8	01/01/2024 21:10:00	456	12,85	12,8
9	01/01/2024 21:20:00	452	12,8	12,85
10	01/01/2024 21:30:00	440	12,8	12,85
11	01/01/2024 21:40:00	432	12,75	12,9
12	01/01/2024 21:50:00	438	12,75	12,9

Figura 43. Ejemplo variables concatenadas

4.4 *Backend*

Una vez presentado el *frontend* a continuación se detalla el *backend*. El *backend* de esta aplicación está diseñado para gestionar procesar y exportar datos ambientales de forma precisa, flexible y eficiente. Funciona como un motor de procesamiento que actúa bajo demanda cuando el usuario realiza una acción desde la interfaz (*frontend*). A continuación, se describe cómo se estructura el *backend*. Para que el *backend* funcione correctamente, primero hay que pasarle a la aplicación dos carpetas (IoB_DATA e IoB_DATA_External) y un archivo Excel (*IoB_spaces_devices.xlsx*), donde tienen que estar ubicados los siguientes ficheros, como se muestra en la figura 44. Además, en este directorio también se incluye el aplicativo Python, `app.py`

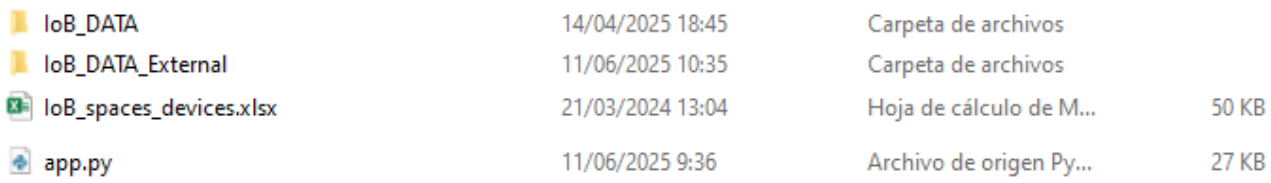


Figura 44. Archivos necesarios para el funcionamiento del aplicativo

- **IoB_DATA.** En esta carpeta, como ya se ha mencionado antes, están ubicados todos los archivos de los sensores, indispensables para poder extraer las variables.
- **IoB_DATA_External.** En esta carpeta están ubicados los archivos Excel referentes a los datos externos, como son (ver figura 45): ocupación, ocupación matriculados, o datos de temperaturas de AEMET, entre otros. Todos estos archivos tienen que irse actualizando según transcurran los períodos de análisis.

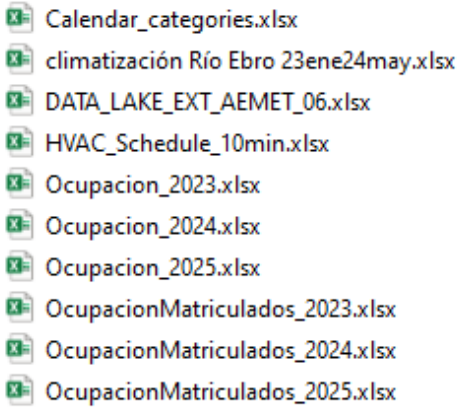


Figura 45. Archivos necesarios dentro de IoB_DATA_External

Una vez están los archivos necesarios en su ubicación, el usuario introduce todos los parámetros:

- **Entrada de parámetros del usuario.** Desde la interfaz, el *backend* recibe:
  - El nombre del aula (por ejemplo, Ada.0.01)
  - El rango de fechas y horas
  - Las variables a incluir (CO_2, temperatura, ocupación…)
  - El formato de exportación deseado (Excel, CSV o JSON)
  - La frecuencia de interpolación (cada 10 min, por ejemplo)

  Estos parámetros determinan qué datos buscar y cómo tratarlos.
- **Búsqueda y lectura de datos.** Después, el *backend* localiza automáticamente los archivos correspondientes a los sensores instalados en esa aula, gracias a un archivo maestro de configuración (*IoB_spaces_devices.xlsx*) que asocia:
  - Cada aula → con sus sensores físicos
  - Cada sensor → con su tipo y ubicación en el sistema de carpetas

  Los datos pueden venir de:
  - Sensores físicos (CO_2, temperatura, humedad, etc.)
  - Fuentes externas (AEMET, calefacción, etc.)
  - Ficheros CSV con nombres estandarizados (*.raw.csv)
- **Procesamiento y limpieza.** Una vez obtenidos los datos:
  - Se filtran según el rango de fechas y horas solicitado
  - Se sincronizan en un índice temporal común con la frecuencia indicada
  - Se aplica interpolación temporal lineal para rellenar huecos
  - Se eliminan duplicados y se normalizan los nombres de las columnas
  - Se realiza un reordenamiento para agrupar las variables según su tipo (base, externas, avanzadas)

  Si alguna variable no está disponible en el rango solicitado, el sistema lo indica y continúa sin ella.
- **Cálculo de completitud.** Antes de exportar los datos, el *backend* analiza qué porcentaje del total de registros tiene valores reales (no interpolados) para las variables seleccionadas. Esto permite:
  - Medir la **calidad de los datos**
  - Informar al usuario de si ese rango es adecuado para el análisis
- **Exportación.** Finalmente, los datos procesados se exportan al formato elegido por el usuario:
  - **Excel**: para análisis manual y visualización
  - **CSV**: para procesamiento en otras herramientas
  - **JSON**: para integraciones o usos web

Finalmente, el archivo generado se guarda en una carpeta con nombre automático que incluye la fecha y hora de la exportación. Además, de incluir:

- Un Excel con los datos alineados
- Un archivo .txt con la completitud general y por variable
- Un archivo .txt con los días dentro del rango seleccionado y su porcentaje de completitud.

Para que el *backend* sea capaz de realizar todas estas funcionalidades, se ha programado el script `procesar_datos.py` que incluye las siguientes funciones:

- **`get_space_telemetries()`**
  Obtiene todos los sensores físicos asociados a cada aula (espacio) consultando el archivo maestro *IoB_spaces_devices.xlsx*. Para cada sensor, busca en qué subcarpetas del sistema se encuentra su fichero **.raw.mat* y devuelve una lista de tuplas con (clave del sensor, tipo, identificador).

- **`sensor_tt()`**
  Carga y prepara los datos de los sensores seleccionados (físicos y externos) para un aula concreta dentro del rango temporal definido.
  - **AEMET**: extrae máximas y mínimas diarias de *DATA_LAKE_EXT_AEMET_06.xlsx.*
  - **HVAC**: importa y depura la hoja de horario de *HVAC_Schedule_10min.xlsx*, elimina duplicados y ajusta el índice.
  - **Meteorología exterior**: lee el archivo *csv* de la carpeta *IoB_DATA_External*, filtra y renombra.
  - **Ocupación** (`occ_boolean`) y **aforo** (`occ_people`): recorre los Excel de ocupación por año, filtra por rango, determina la columna de aula, elimina duplicados.
  - **Sensores físicos**: para cada tipo de sensor y cada ID encontrado, carga su **.raw.csv*, filtra por el rango, renombra la columna y elimina duplicados.

- **`round_to_nearest_minutes()`**
  Función auxiliar que redondea cada *timestamp* al múltiplo de base minutos más cercano por defecto hacia abajo.

- **`añadir_calendar_categories()`**
  Fusiona el *DataFrame* final con el calendario académico (*calendar_categories.xlsx*), añadiendo una columna *calendar_categories* alineada con el índice temporal.

- **`exportTT()`**
  Se encarga de exportar un *DataFrame* al formato deseado (*csv*, *xlsx* o *json*):
  - Crea carpetas si no existen.
  - Formatea el índice como cadena DD/MM/YYYY HH:MM:SS.
  - En JSON convierte nombres de columnas y formatea números con un decimal.

- **`export_data()`**
  Función principal que orquesta la recopilación, sincronización y exportación de los datos:
  - **Parseo de fechas y horas**: valida rangos, ajusta la fecha/hora de inicio al siguiente múltiplo de interpolación.
  - **Generación del índice común** (*sync_interval*) con frecuencia *interpolacion minutos*. Para cada aula llama a `sensor_tt` y separa los *DataFrames* según su origen.
  - **Reindexado temporal**: usa la función interna **`reindex_df()`** para ajustar cada serie al índice maestro, redondear *timestamps*, eliminar duplicados y rellenar desde el vecino más cercano con tolerancia de medio intervalo.
  - **Expansión de AEMET**: convierte los datos diarios en series a la frecuencia de `sync_interval`.
  - **Concatenación** de todas las fuentes, y si se solicita, llamada a `añadir_calendar_categories.`
  - **Cálculo de completitud**:
    - Porcentaje global y por variable (excluyendo `hvac_boolean`).
    - Clasificación de cada día en “100%”, “0%”, “100% with interpolation” o “some_data” en función del umbral de NaN (10%).
    - Generación de dos archivos de informe en la carpeta de exportación del aula:
      - **quality.txt**: resumen de fechas, completitud global y por variable, total de días y conteo por categoría.
      - **days.txt**: línea por día con su estado.
  - **Interpolación y limpieza final**: elimina filas totalmente vacías (sin HVAC), rellena numéricos por interpolación “time”, limpia `hvac_boolean` a 0 donde falte, renombra columnas de “raw” a “app” según *raw_to_app,* y reordena en base a *base_keys_app, ext_keys_app, adv_keys_app.*
  - **Exportación definitiva**: invoca `exportTT` para generar el archivo final en la carpeta de salida.

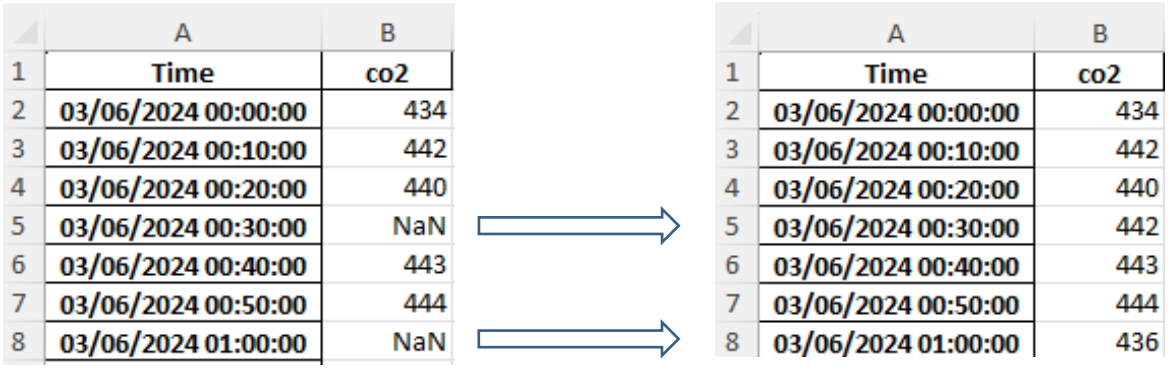
Con este conjunto de funciones, el *backend* puede localizar, cargar, sincronizar, interpolar, analizar calidad y exportar en distintos formatos todos los datos de sensores y fuentes externas para las aulas configuradas.

5 Validación del aplicativo

En este capítulo se presentan el conjunto de pruebas y comprobaciones que garantizan la fiabilidad y la integridad de datos procesados por el aplicativo. Se abordan las casuísticas de datos perdidos o inconsistentes, huecos puntuales, horas completas sin lecturas, duplicados de *timestamp* o desalineaciones temporales, etc. Se detalla de forma concisa cómo cada una es detectada, normalizada y, en la medida de lo posible, corregida o interpolada automáticamente. Además, se describe el mecanismo de generación de informes de calidad (*quality.txt y days.txt*), que permite cuantificar la completitud global y diaria de las series antes de su exportación.

5.1 Casuísticas

- **Huecos puntuales dentro de una hora**. Por ejemplo, faltan lecturas de CO_2 durante 5 minutos dentro de un intervalo de 60 min. Esos huecos pequeños se rellenan automáticamente mediante interpolación lineal entre el dato anterior y el siguiente, ver figura 46.



	A	B
1	Time	co2
2	03/06/2024 00:00:00	434
3	03/06/2024 00:10:00	442
4	03/06/2024 00:20:00	440
5	03/06/2024 00:30:00	NaN
6	03/06/2024 00:40:00	443
7	03/06/2024 00:50:00	444
8	03/06/2024 01:00:00	NaN

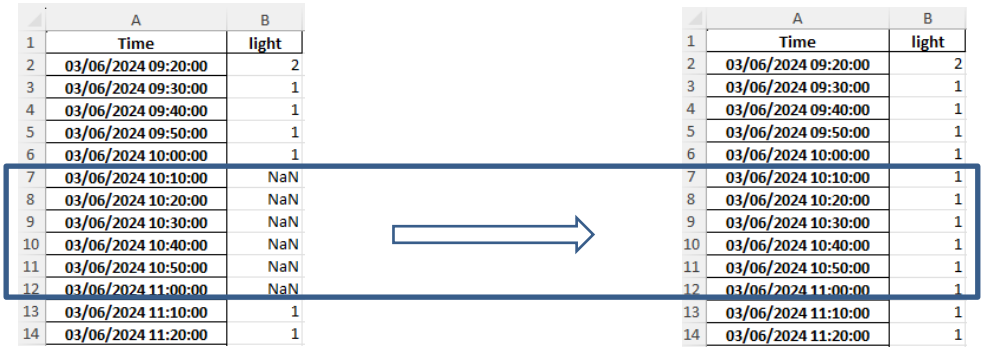
(a) Antes del procesado

	A	B
1	Time	co2
2	03/06/2024 00:00:00	434
3	03/06/2024 00:10:00	442
4	03/06/2024 00:20:00	440
5	03/06/2024 00:30:00	442
6	03/06/2024 00:40:00	443
7	03/06/2024 00:50:00	444
8	03/06/2024 01:00:00	436

(b) Después del procesado

Figura 46. Ejemplo NaN dentro de una hora

- **Falta de datos de un sensor durante toda una hora.** Por ejemplo, el sensor de luminosidad dejó de enviar datos entre las 10:00 y las 11:00, en ese caso, la serie de luminosidad queda con *NaN* en esos seis puntos. Se eliminan y se pone el último dato recibido, con esto no se consigue fiabilidad, pero sirve para que no haya problemas a la hora de exportar. Con el uso de *days.txt* y de los intervalos con mejores datos, se puede evitar el uso de días, en los que faltan datos durante horas. Como se puede ver en la figura 47.



	A	B
1	Time	light
2	03/06/2024 09:20:00	2
3	03/06/2024 09:30:00	1
4	03/06/2024 09:40:00	1
5	03/06/2024 09:50:00	1
6	03/06/2024 10:00:00	1
7	03/06/2024 10:10:00	NaN
8	03/06/2024 10:20:00	NaN
9	03/06/2024 10:30:00	NaN
10	03/06/2024 10:40:00	NaN
11	03/06/2024 10:50:00	NaN
12	03/06/2024 11:00:00	NaN
13	03/06/2024 11:10:00	1
14	03/06/2024 11:20:00	1

(a) Antes del procesado

	A	B
1	Time	light
2	03/06/2024 09:20:00	2
3	03/06/2024 09:30:00	1
4	03/06/2024 09:40:00	1
5	03/06/2024 09:50:00	1
6	03/06/2024 10:00:00	1
7	03/06/2024 10:10:00	1
8	03/06/2024 10:20:00	1
9	03/06/2024 10:30:00	1
10	03/06/2024 10:40:00	1
11	03/06/2024 10:50:00	1
12	03/06/2024 11:00:00	1
13	03/06/2024 11:10:00	1
14	03/06/2024 11:20:00	1

(b) Después del procesado

Figura 47. Ejemplo NaN una hora seguida

- **Día entero sin ninguna lectura.** Por ejemplo, el sensor de temperatura exterior no funcionó el día 05-06-2025. Ese día se refleja en days.txt (“05-06-2025: 0%”) y en el resumen de *quality.txt*. Aunque no hay datos que rellenar, el *script* continúa con el resto de los días sin interrumpir la exportación del resto de aulas, véase en la figura 48.

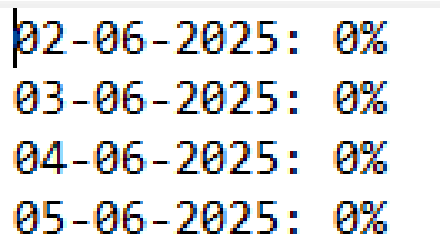


Figura 48. Ejemplo days.txt 0%

- **Valores nulos o no numéricos en columnas avanzadas/externas.** Por ejemplo, faltan columnas de `calendar_categories` porque no había registro. Se añade `calendar_categories` intenta fusionar, y si falla, simplemente deja la tabla tal cual. En JSON y CSV, cualquier campo no numérico se convierte a *string*, evitando errores, como se puede ver en la figura 49.

	A	B	C	D	E
1	Time	occ_boolean	occ_people	hvac_boolean	calendar_categories
2	02/06/2025 00:00:00	0	0	0	
3	02/06/2025 00:10:00	0	0	0	
4	02/06/2025 00:20:00	0	0	0	
5	02/06/2025 00:30:00	0	0	0	
6	02/06/2025 00:40:00	0	0	0	
7	02/06/2025 00:50:00	0	0	0	
8	02/06/2025 01:00:00	0	0	0	
9	02/06/2025 01:10:00	0	0	0	
10	02/06/2025 01:20:00	0	0	0	
11	02/06/2025 01:30:00	0	0	0	
12	02/06/2025 01:40:00	0	0	0	
13	02/06/2025 01:50:00	0	0	0	
14	02/06/2025 02:00:00	0	0	0	

Figura 49. Ejemplo columna datos vacía

5.2 Ficheros de validación

En esta sección se explica en detalle el funcionamiento de *quality.txt y days.txt*. Ambos ficheros junto con los tres mejores intervalos de tiempo (ver figura 37), sirven de retroalimentación para el usuario y para que sea capaz de poder trabajar con un mejor conjunto de datos.

5.2.1 Quality.txt

El formato de este fichero de texto es el que se ve en la figura 50. En primer lugar, aparece la fecha inicial y final de la exportación, para tener un marco temporal. Luego aparece la completitud global, que ya se ha explicado su obtención en el capítulo 2, además de la completitud por variable. Finalmente, en la última sección *days,* se nos muestra los días totales dentro del rango de fechas seleccionado, los días a los que no les falta ningún dato, los días llamados *100% with interpolation,* que son los días que tienen un 90% de datos, que, mediante interpolación, se puede considerar que son días casi completos, luego los días con algunos datos, que son los que tienen datos, pero menos de un 90% y finalmente los días que no tienen datos.

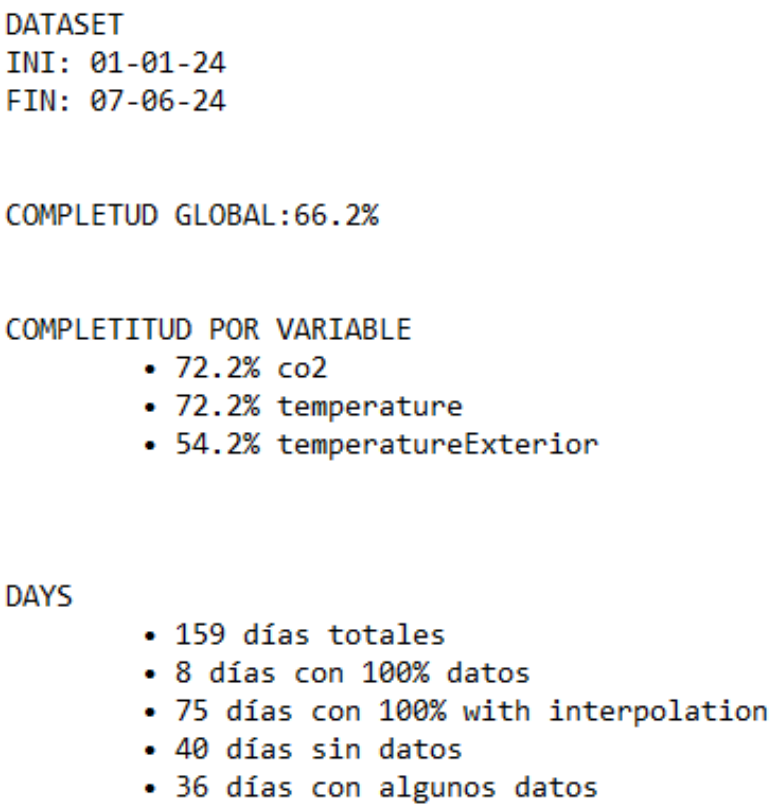


Figura 50. Ejemplo fichero quality.txt

5.2.2 Days.txt

En este fichero se detalla día a día, la completitud de los datos (ver figura 51). Como se ha explicado en el apartado 5.2.1, se dividen los días en cuatro rangos: los días que tienen un 100% de datos, los que tienen más de un 90% (y se llaman *100% with interpolation*), los días en los que hay datos (pero menos de un 90%), y los días para los que no hay ningún dato. Cuando se seleccionan los mejores intervalos, solo se tienen en cuenta días con el 100% o al menos con un 90% de datos.

```
01-01-2024: some_data
02-01-2024: 100% with interpolation
03-01-2024: 100% with interpolation
04-01-2024: 100%
05-01-2024: 100% with interpolation
06-01-2024: 100% with interpolation
07-01-2024: 100% with interpolation
08-01-2024: 100% with interpolation
09-01-2024: 100%
10-01-2024: 100% with interpolation
11-01-2024: 100% with interpolation
12-01-2024: 100% with interpolation
13-01-2024: 100% with interpolation
14-01-2024: 100% with interpolation
15-01-2024: 100% with interpolation
16-01-2024: 100% with interpolation
17-01-2024: 100% with interpolation
18-01-2024: 100% with interpolation
19-01-2024: 100% with interpolation
20-01-2024: 100% with interpolation
21-01-2024: 100% with interpolation
22-01-2024: 100% with interpolation
23-01-2024: 100% with interpolation
24-01-2024: 100% with interpolation
25-01-2024: 100% with interpolation
26-01-2024: 100% with interpolation
27-01-2024: 100% with interpolation
```

Figura 51. Ejemplo contenido days.txt

6 Conclusiones y líneas futuras

6.1 Conclusiones

A lo largo del desarrollo de este TFG se han aprendido varias lecciones clave que orientarán mejoras y buenas prácticas en futuros proyectos de integración de datos IoT en entornos de *smart campus*:

- **Modularidad y claridad del código.** Encapsular cada etapa del flujo (ingesta, limpieza, sincronización, exportación) en funciones independientes facilita enormemente la detección de errores y la incorporación de nuevas fuentes de datos. Además de mantener interfaces bien definidas entre módulos reduce el riesgo de efectos colaterales al modificar o ampliar funcionalidades.
- **Gestión rigurosa de la temporalidad.** Las desalineaciones de *timestamps* y los formatos heterogéneos suponen la mayor fuente de inconsistencias. Diseñar un índice maestro y aplicar redondeos consistentes garantiza que todos los sensores queden alineados.
- **Detección y tratamiento de datos faltantes.** Definir umbrales claros (por ejemplo, 10 % de huecos diarios) permite distinguir entre situaciones de interpolación segura y días con cobertura insuficiente. Generar informes automáticos (*quality.txt y days.txt*) aporta transparencia sobre la calidad del *dataset*. Así se consigue realimentación al usuario y facilita la usabilidad del aplicativo.
- **Importancia de los *reports* de calidad.** El simple hecho de contar y categorizar días completos, parciales o vacíos ayuda a priorizar tareas de mantenimiento de sensores o revisión de pipelines.
- **Diseño centrado en el usuario.** La interfaz en PyQt5, con validaciones en tiempo real y *feedback* continuo, ayuda a reducir el número de errores en la selección de parámetros y mejorar la facilidad a la hora de usar la app por cualquier tipo de usuario.
- **Escalabilidad y adaptabilidad.** El uso de diccionarios de mapeo semántico (*raw_to_app*) y de configuraciones parametrizables (lista de aulas, rutas de archivos, formatos de exportación) permite incorporar nuevos sensores con un mínimo de cambios.

En conclusión, se ha avanzado con un aplicativo sólido y robusto que, además, proporciona un marco de trabajo para futuros despliegues y ampliaciones en el ámbito del *smart campus*. A partir de los resultados obtenidos y las lecciones aprendidas, se describen algunas de las líneas futuras.

6.2 Líneas futuras

A partir de los resultados obtenidos y las conclusiones previas, se identifican varias líneas de trabajo que pueden potenciar la herramienta para la gestión de datos IoT en entornos de *smart campus*:

- **Integración de procesamiento en tiempo real.** Incorporar un módulo de ingesta en *streaming* que permita procesar lecturas al instante y alimentar *dashboards*. Este enfoque reduciría la latencia entre la adquisición y el análisis, facilitando la detección inmediata de anomalías.
- **Ampliación de fuentes de datos y sensores.** El soporte para nuevos dispositivos, como medidores de energía eléctrica y de agua o sensores avanzados de calidad del aire y ruido, enriquecería el ecosistema de datos y abriría la puerta a análisis más completos.
- ***Dashboard* y visualizaciones interactivas**. Un panel web que ofreciese visualizaciones en tiempo real de las series temporales, mapas de calor y alertas configurables por umbral. Incluir gráficas de completitud diaria y mensual facilitaría al usuario explorar patrones de datos faltantes, comparando variables y detectando tendencias a largo plazo. La posibilidad de filtrar por aula, rango horario o tipo de sensor aumentaría la usabilidad y la toma de decisiones.
- **Análisis avanzado y modelos predictivos.** Desarrollar algoritmos de detección de anomalías basados en técnicas no supervisadas para automatizar la identificación de lecturas atípicas antes de la exportación. Paralelamente, entrenar modelos de predicción de temperatura interior o niveles de CO_2 para anticipar necesidades de climatización y ocupación, optimizando el consumo energético y mejorando el confort de los usuarios.
- **Automatización de actualizaciones y alertas.** Programar tareas periódicas para generar automáticamente los *datasets* diarios y enviar notificaciones de completitud o incidencias facilitaría la monitorización continua sin intervención manual. La implementación de un mecanismo de umbral configurable, que alerte cuando la cobertura de datos baje de un porcentaje, reforzaría la fiabilidad y acortaría los tiempos de respuesta ante fallos de sensores.

.

Bibliografía

[1] European Commission. Energy performance of buildings directive (2021) https://energy.ec.europa.eu/topics/energy-efficiency/energy-efficient-buildings/energyperformance-buildings-directive_en

[2] European Commission. Indoor air pollution (2003) https://ec.europa.eu/commission/presscorner/detail/en/IP_03_1278

[3] United Nations. The Paris Agreement (2015) https://www.un.org/en/climatechange/paris-agreement

[4] J. L. Olivera Pinies (2024). Diseño, desarrollo e implementación de un entorno de análisis multivariable para el estudio de la eficiencia energética y la calidad ambiental en un smart campus (TFM). Universidad de Zaragoza.

[5] Agencia Estatal de Meteorología AEMET: https://www.aemet.es/es/

[6] Servicio de mantenimiento de la Universidad de Zaragoza https://oficinaverde.unizar.es/

[7] Gobierno de España. Horizonte Europa. Misiones. https://www.horizonteeuropa.es/misiones.

[8] Indoor Ambiance Monitoring Sensor — AM300 Series User Guide, MILESIGHT (2025) https://resource.milesight-iot.com/milesight/document/am300-series-user-guide-en.pdf

[9] Aranet4 HOME and Aranet4 PRO, ARANET (2025) https://naltic.com/wp content/uploads/2023/04/Aranet4_User_Manual_v24_WEB.pdf

[10] Outdoor Environment Monitoring Sensor — EM500 Series User Guide, MILESIGHT (2025) https://resource.milesight-iot.com/milesight/document/em500-series-user-guide-en.pdf

[11] Khomp Meteo Estación Meteorológica, Khomp (2025) https://www.khomp.com/wp-content/uploads/2020/08/Extens%C3%A3o_de_Clima_-_PT-v2.pdf

[12] Siemens QPA1004. Room air quality sensor CO2 (2025) https://hit.sbt.siemens.com/RWD/app.aspx?rc=Baltics&lang=en&module=Catalog&action=ShowProduct&key=S55720-S453

[13] AI Workplace Occupancy Sensor VS121 User Guide, MILESIGHT (2025) https://resource.milesight-iot.com/milesight/document/vs121-user-guide-en.pdf

[14] Xovis Technical Documentation PC-Series User Manual, XOVIS (2025) https://downloads.adpgauselmann.de/Andere%20Produkte/PLAYSAFE/Xovis/20180713_PC_Series_User_Manual.pdf

[15] Sistema de Información Geográfica de la Universidad de Zaragoza (SIGEUZ) https://sigeuz.unizar.es/

[16] Universidad de Zaragoza. Oficina Verde. Horarios de encendido de instalaciones generales frío y calor. https://oficinaverde.unizar.es/horarios-de-encendido-de-las-instalaciones-generales-de-fr%C3%ADo-y-calor

Anexo I. APP Python (app.py)

miApp (Clase principal de la interfaz)

La clase principal MiApp crea la interfaz gráfica de usuario para seleccionar aulas, intervalos, fechas, formato y variables, y permite exportar datasets en distintos niveles de complejidad.

Aquí se encuentra un ejemplo de como se ha creado el menú de las aulas disponibles:

```
52
53  class MiApp(QWidg
54      def __init__(
55          super().__
56          self.setW
57          self.setG
58
59          layout_pr
60          self.lista
61
62          titulos =
63
64          botones =
65          lista_tit
66
67          lista_ite
68              [],
69              [],
70              ["co2
71              ["tmp
72              ["hum
73          ]
74          self.data
75          self.new_
76          # --- Gen
77          aulas_dis
78
79          # 1) Ada.
80          aulas_dis
81          # 2) Ada.
82          aulas_dis
83          # 3) Ada.2.21 a Ada.2.25
84          aulas_disponibles += [f"Ada.2.{i}" for i in range(21, 26)]
85          # 4) Ada.0.CN
86          aulas_disponibles.append("Ada.0.CN")
87          # 5) Bet.0.01 a Bet.0.04
88          aulas_disponibles += [f"Bet.0.{i:02d}" for i in range(1, 5)]
89          # 6) Bet.1.01 a Bet.1.09
90          aulas_disponibles += [f"Bet.1.{i:02d}" for i in range(1, 10)]
91          # 7) Bet.2.01 a Bet.2.21, excepto Bet.2.17
92          for i in range(1, 22):
93              if i == 17:
94                  continue
95              aulas_disponibles.append(f"Bet.2.{i:02d}")
96          # 8) Bet.3.01 a Bet.3.09
97          aulas_disponibles += [f"Bet.3.{i:02d}" for i in range(1, 10)]
```

```
# Columna 0: botón Seleccionar aulas + combo intervalos + combo formatos
if i == 0:
    btn_aulas = QPushButton("Aulas availables")
    btn_aulas.setStyleSheet("""
        QPushButton {
            border-radius: 8px;
            font-weight: bold;
            font-size: 13px;
            padding: 8px;
            background-color: #90caf9;
            color: #0d47a1;
        }
        QPushButton:hover {
            background-color: #64b5f6;
        }
        QPushButton:pressed {
            background-color: #42a5f5;
        }
    """)
    btn_aulas.clicked.connect(lambda: self.abrir_selector_aulas(aulas_disponibles))

    btn_layout = QHBoxLayout()
    btn_layout.setAlignment(Qt.AlignCenter)
    btn_layout.addWidget(btn_aulas)
    columna.addLayout(btn_layout)
    columna.addSpacing(15)
```

```
# Ext Keys (columna 3)
```

seleccionar_directorio_salida()

Método para seleccionar un directorio de salida mediante QFileDialog.

- **Entrada**: ninguna.
- **Salida**: ruta del directorio seleccionada, almacenada en self.output_dir y mostrada en una etiqueta.

```
500     def seleccionar_directorio_salida(self):
501         ruta = QFileDialog.getExistingDirectory(
502             self,
503             "Elige carpeta para exportar datos",
504             os.getcwd(),
505             QFileDialog.ShowDirsOnly | QFileDialog.DontResolveSymlinks
506         )
507         if ruta:
508             self.output_dir = ruta
509             self.label_output_dir.setText(f"Carpeta salida: {ruta}")
510         else:
511             # Si el usuario cancela, no hacemos nada
512             return
```

abrir_selector_aulas(aulas)

Abre el diálogo para seleccionar aulas y muestra un resumen con las variables disponibles por aula.

- **Entrada**: lista de aulas.
- **Salida**: actualiza self.aulas_seleccionadas.

```
514     def abrir_selector_aulas(self, aulas):
515         dialogo = SelectorAulas(aulas)
516         if dialogo.exec_():
517             self.aulas_seleccionadas = dialogo.obtener_seleccion()
518 
519             if not self.aulas_seleccionadas:
520                 QMessageBox.information(
521                     self,
522                     "Aulas seleccionadas",
523                     "Ninguna aula seleccionada."
524                 )
525                 return
526 
527             # Construimos el texto que mostraremos
528             mensaje = "Aulas seleccionadas y variables disponibles:\n"
529             for aula in self.aulas_seleccionadas:
530                 # get_space_telemetries devuelve listas [raw_key, sensor_type, sensor_name]
531                 lst = get_space_telemetries([aula])
532                 # Extraer solo raw_key de cada entrada
533                 raw_keys = [entrada[0] for entrada in lst]
534                 # Traducir raw_key a app_key (o quedarnos con raw_key si no hay mapeo)
535                 app_keys = sorted({ raw_to_app.get(rk, rk) for rk in raw_keys })
536 
537                 if app_keys:
538                     mensaje += f"\n  • {aula}:\n"
539                     for clave_app in app_keys:
540                         mensaje += f"      - {clave_app}\n"
541                 else:
542                     mensaje += f"\n  • {aula}: (no se encontraron sensores)\n"
543 
544             QMessageBox.information(
545                 self,
546                 "Aulas y variables disponibles",
547                 mensaje
548             )
```

mostrar_mensaje(columna)

Función principal de exportación. Ejecutada al pulsar cualquiera de los tres botones de exportación (base, completo, avanzado).

- **Entrada**: número de columna.
- Recoge selección de claves, fechas, horas, formato e intervalo.
- Traduce claves de app a claves crudas.
- Solicita carpeta de salida.
- Inicia exportación en segundo plano con ExportWorker y muestra QProgressDialog.

```
550     def mostrar_mensaje(self, columna):
551         # 1) Validar que haya al menos un aula seleccionada
552         if not self.aulas_seleccionadas:
553             QMessageBox.warning(self, "Advertencia", "No has seleccionado ninguna aula.")
554             return
555 
556         # 2) Recoger parámetros de fecha, hora, formato e intervalo
557         fecha_inicio = self.fecha_inicio.date().toString("dd/MM/yyyy")
558         fecha_fin    = self.fecha_fin.date().toString("dd/MM/yyyy")
559         hora_inicio  = f"{self.hora_inicio.value():02}:00"
560         hora_fin     = f"{self.hora_fin.value():02}:00"
561         formato      = self.combo_formatos.currentText().replace(".", "")
562         intervalo    = intervalo = self.intervalo_spinbox.value()
563 
564         # 3) Construcción de 'claves_app' según el botón pulsado
565         if columna == 2:
566             seleccionados_base = self.listas[2].selectedItems() if self.listas[2] else []
567             claves_app = [item.text() for item in seleccionados_base]
568             if not claves_app:
569                 QMessageBox.warning(self, "Advertencia", "No has seleccionado ninguna clave en 'Base Keys'.")
570                 return
571             titulo_msj    = "Dataset Base"
572             mensaje_exito = "El Dataset Base ha sido exportado correctamente."
573             export_type = "base"
574 
```

exportWorker (QThread)

Hilo en segundo plano que ejecuta la exportación para no bloquear la interfaz.

- **Entradas**:
  - spaces: aulas seleccionadas.
  - keys: claves crudas a exportar.
  - i_date, f_date: fecha de inicio y fin.
  - hora_inicio, hora_fin: horas dentro del día.
  - fmt: formato del archivo.
  - interpolacion: intervalo de muestreo.
  - output_base_path: directorio destino.
- **Salida**: ejecuta export_data() y emite las señales finished, error, progress.

```python
class ExportWorker(QThread):
    # Señales:
    progress = pyqtSignal(int)      # valor 0-100
    finished = pyqtSignal()         # al terminar sin errores
    error = pyqtSignal(str)         # mensaje de error en caso de excepción

    def __init__(self, spaces, keys, i_date, f_date, fmt, interpolacion, hora_inicio, hora_fin, output_base_path, export_type):
        super().__init__()
        self.spaces = spaces
        self.keys = keys
        self.i_date = i_date
        self.f_date = f_date
        self.fmt = fmt
        self.interpolacion = interpolacion
        self.hora_inicio = hora_inicio
        self.hora_fin = hora_fin
        self.output_base_path = output_base_path
        self.export_type      = export_type
```

on_progress(porcentaje)

Actualiza la barra de progreso visible mediante la señal progress.

- Entrada: porcentaje (0-100).

```python
def on_progress(self, porcentaje):
    if self.progress_dialog:
        self.progress_dialog.setValue(porcentaje)
```

on_finished()

Finaliza la barra de progreso y muestra mensaje de éxito.

```python
def on_finished(self):
    if self.progress_dialog:
        self.progress_dialog.setValue(100)
        self.progress_dialog.close()
        self.progress_dialog = None
    self.show_best_intervals()
    QMessageBox.information(self, "Éxito", "El dataset ha sido exportado correctamente.")
```

on_error(mensaje_error)

Cierra la barra de progreso y muestra un mensaje de error.

Componentes visuales adicionales

- QLabel, QComboBox, QDateEdit y QSpinBox para la selección de parámetros.
- Estilos CSS en linea para mejorar la apariencia.

Organización de datos

- Secciones separadas por columnas: espacios, fecha/hora, base, completo, avanzado.
- Claves separadas por tipo: base_keys_app, ext_keys_app, adv_keys_app.
- Traducción de nombres de sensores: raw_to_app, app_to_raw.

Este script permite al usuario seleccionar los parámetros necesarios para la exportación de datasets con datos sensorizados de aulas y generar ficheros estructurados en formato Excel, CSV o JSON.

```python
def on_error(self, mensaje_error):
    if self.progress_dialog:
        self.progress_dialog.close()
        self.progress_dialog = None
    QMessageBox.critical(self, "Error", f"Error al exportar el dataset:\n{mensaje_error}")
    # ---- Campos para los paths seleccionados ----
```

Anexo II. *APP Python (procesar_datos.py)*

A continuación, se describen las funciones desarrolladas en el script principal de exportación de datos, procesar_datos.py, siguiendo un formato funcional y estandarizado.

exportTT

La función se encarga de exportar un DataFrame con datos temporales al disco en el formato indicado por el usuario.

Parámetros de entrada:

- **df:** DataFrame que contiene los datos a exportar.
- **fmt**: Formato deseado de exportación ('csv', 'xlsx' o 'json').
- **path**: Ruta completa de salida donde se guardará el archivo.

Funcionamiento:

- Crea las carpetas necesarias.
- Convierte el índice a formato de fecha legible.
- Exporta en el formato especificado utilizando los métodos de Pandas y JSON.

```python
def exportTT(df, fmt, path):

    os.makedirs(os.path.dirname(path), exist_ok=True)

    print(f"Intentando exportar archivo en formato {fmt} a: {path}")

    df.index = df.index.strftime('%d/%m/%Y %H:%M:%S')

    if fmt == 'csv':
        print(f"Exportando CSV a {path}...")
        df.to_csv(path, index_label='Time')

    elif fmt == 'xlsx':
        try:
            print(f"Exportando Excel a {path}...")
            df.to_excel(path, index_label='Time', engine='openpyxl')
            print(f"Archivo Excel exportado correctamente a {path}.")
        except Exception as e:
            print(f"Error al exportar a Excel: {e}")

    elif fmt == 'json':
        json_data = {'Time': df.index.tolist()}
        for col in df.columns:
            clean_col = col.replace('.', '_')
            if pd.api.types.is_numeric_dtype(df[col]):
                json_data[clean_col] = df[col].map(lambda x: f"{x:.1f}").tolist()
            else:
                json_data[clean_col] = df[col].astype(str).tolist()

        with open(path, 'w', encoding='utf-8') as f:
            json.dump(json_data, f, ensure_ascii=False, indent=4)
```

get_space_telemetries

Se utiliza para buscar en el Excel de configuración todos los sensores asignados a uno o varios espacios del campus.

Parámetros de entrada:

- **spaces**: Lista de nombres de espacios/aulas.
- **excel_path**: Ruta al archivo Excel con los dispositivos (por defecto 'IoB_spaces_devices.xlsx').
- **data_folder**: Carpeta base donde se almacenan los datos (por defecto 'IoB_DATA').

Funcionamiento:

- Carga el Excel y filtra por los espacios seleccionados.
- Extrae los sensores disponibles y sus IDs.
- Devuelve una lista de tripletas: nombre del sensor, tipo de sensor, y ID del dispositivo.

```python
def get_space_telemetries(spaces, excel_path='IoB_spaces_devices.xlsx', data_folder='IoB_DATA'):
    df = pd.read_excel(excel_path, sheet_name='dispositivos')
    sensor_columns = df.columns[3:]
    space_telemetries = []
    for space in spaces:
        space_rows = df[df.iloc[:, 0] == space]
        for _, row in space_rows.iterrows():
            for sensor_type in sensor_columns:
                sensor_id = row[sensor_type]
                if pd.notna(sensor_id):
                    sensor_path = os.path.join(data_folder, str(sensor_type), str(sensor_id))
                    pattern = os.path.join(sensor_path, '*raw*.mat')
                    raw_files = glob.glob(pattern)
                    for filepath in raw_files:
                        filename = os.path.basename(filepath)
                        key_name = filename.split('.')[0]
                        space_telemetries.append([key_name, sensor_type, sensor_id])
    return space_telemetries
```

sensor_tt

Esta función recupera las series temporales de datos de sensores físicos y fuentes externas como AEMET, SCADA HVAC o datos de ocupación.

Parámetros de entrada:

- **space**: Nombre del espacio a procesar (ej. "Ada.0.01").
- **keys**: Lista de variables seleccionadas.
- **time_range**: Tupla con timestamps de inicio y fin del intervalo deseado.

Funcionamiento:

- Filtra y carga datos según las variables seleccionadas.
- Aplica lógicas independientes para AEMET, HVAC, ocupación y sensores.
- Devuelve una lista de tuplas con DataFrames y su origen.

```
def sensor_tt(space: str, keys: List[str], time_range: Tuple[pd.Timestamp, pd.Timestamp]) -> List[Tuple[pd.DataFrame, str]]:
    """
    Para el espacio 'space' y la lista de 'keys' (sensores + externos), devuelve
    una lista de tuplas (DataFrame, sensor_name) con:
      - Índice datetime y columnas correspondientes a cada clave.
      - Si 'tmp_aemet.max'/'tmp_aemet.min'/'tmp_meteo' están en keys, lee el Excel AEMET.
      - Si 'occ_boolean' está en keys, lee los Excel de Ocupación 2023/2024/2025 y extrae
        la columna correspondiente a 'space' (p. ej. "Ada.0.01" → columna que empieza "A.01").
      - El resto de keys se tratan como sensores físicos (igual que antes).
    """
```

round_to_nearest_minutes

Sirve para redondear marcas temporales a un múltiplo inferior del intervalo definido.

Parámetros de entrada:

- **dtindex**: Índice temporal de tipo datetime.
- **base**: Intervalo de tiempo (en minutos).

Funcionamiento:

- Aplica una transformación a cada marca de tiempo eliminando los minutos/segundos sobrantes.

```
def round_to_nearest_minutes(dtindex, base):
    """Redondea los timestamps al múltiplo de 'base' minutos inferior (por ejemplo, 10)."""
    return dtindex.map(lambda dt: dt - pd.Timedelta(minutes=dt.minute % base, seconds=dt.second, microseconds=dt.microsecond))
```

añadir_calendar_categories

Añade la columna calendar_categories al DataFrame, indicando el tipo de día (laborable, fin de semana, festivo, etc.).

Parámetros de entrada:

- **df_final**: DataFrame con índice temporal ya sincronizado.
- **calendar_path**: Ruta del archivo Excel con los datos de calendario categorizado.

Funcionamiento:

- Carga el Excel, convierte el campo Time a datetime y lo establece como índice.
- Fusiona los datos con el DataFrame principal usando el índice temporal.

```python
def añadir_calendar_categories(df_final: pd.DataFrame, calendar_path: str = "IoB_DATA_External/Calendar_categories.xlsx") -> pd.DataFrame:
    import pandas as pd

    try:
        calendario = pd.read_excel(
            calendar_path,
            sheet_name="Sheet1",
            parse_dates=["Time"]
        )
    except Exception as e:
        print(f"Error al leer el calendario académico: {e}")
        return df_final

    calendario.set_index("Time", inplace=True)
    calendario.index = pd.to_datetime(calendario.index)

    df_final.index = pd.to_datetime(df_final.index).tz_localize(None)

    df_final = df_final.merge(
        calendario[["calendar_categories"]],
        how="left",
        left_index=True,
        right_index=True
    )

    return df_final
```

export_data

Es la función principal de procesamiento y exportación de los datos del sistema IoT.

Parámetros de entrada:

- **spaces**: Lista de espacios/aulas a procesar.
- **keys**: Lista de variables seleccionadas.
- **i_date**: Fecha de inicio (formato "dd/mm/yyyy").
- **f_date:** Fecha final.
- **hora_inicio, hora_fin**: Horas de inicio y fin del periodo diario.
- **interpolacion:** Intervalo de sincronización en minutos.
- **fmt:** Formato de salida ('csv', 'xlsx', 'json').
- **output_base_path:** Carpeta base para almacenar las exportaciones.
- **export_type:** Base, external, advanced.

Funcionamiento:

- Valida coherencia de fechas, horas y el intervalo.
- Genera un índice de timestamps sincronizados para todo el periodo.
- Llama a sensor_tt para cada espacio y obtiene los datos solicitados.
- Reindexa, interpola y unifica todas las variables.
- Añade información de calendario si se selecciona.
- Crea un informe de completitud de datos.
- Exporta el resultado final en el formato deseado.

Estas funciones conforman el sistema de tratamiento de datos del proyecto, automatizando la generación de datasets depurados y personalizados según los criterios definidos por el usuario.