



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



ESCUELA TÉCNICA  
SUPERIOR INGENIERÍA  
INDUSTRIAL VALENCIA

**TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL**

# **DESARROLLO DE UN SISTEMA DE GESTIÓN DE BATERÍAS DE CONEXIÓN A RED MEDIANTE TÉCNICAS DE BIG DATA E INTELIGENCIA ARTIFICIAL**

AUTOR: Ricardo Gómez-Aldaraví Sotos

TUTOR: Emilio Figueres Amorós

COTUTOR: Raúl Gonzalez Medina

**Curso Académico: 2021-22**



# AGRADECIMIENTOS

*“A mi madre, por permitirme cursar mis estudios*

*A mis amigos, por ayudarme cuando más lo necesitaba*

*A Nacho, por echarse un TFT cuando necesitaba despejarme*

*A Vicent y Valle, por aguantar conversaciones técnicas por Discord*

*A mis tutores, por permitirme desarrollar esta idea*

*Y a Isabel, por confiar siempre en mí y hacerme reír”*

# RESUMEN

Las baterías son el cuello de botella que impide una transición energética rápida y barata. En los últimos años estamos siendo testigos del crecimiento de las tecnologías relacionadas con la inteligencia artificial y *Big Data*. Mediante la aplicación de estas técnicas, se pueden desarrollar nuevas aplicaciones que sean capaces de generar predicciones y tomar decisiones fundamentadas en el análisis de los datos históricos. Así, es posible idear una aplicación y una metodología que mejore la gestión de las baterías, extendiendo su vida útil y permitiendo usar baterías más pequeñas. Todo esto es posible implementarlo en el marco de las denominadas *Smart Grids*, que facilitan la interacción de unos dispositivos con otros en el ecosistema de la red eléctrica, de tal forma que se asegure un sistema energético sostenible y eficiente, con bajas pérdidas y altos niveles de calidad y seguridad de suministro.

**PALABRAS CLAVE:** Inteligencia Artificial, IA, Big Data, Machine Learning, Redes LSTM, Python, Smart Grids, Ciencia de datos, Gestión de baterías, ZigBee

# ABSTRACT

Batteries are the bottleneck that are blocking the way to fast and cheap energy transition. Recently, artificial intelligence and Big Data technologies are getting more and more industry adoption. Applying these new techniques, it is possible to develop new applications capable of generating forecasts and taking decisions based on data. This thesis aims to develop an application and a methodology that lead to better battery management, improving their life expectancy and decreasing the required battery capacity for a given application. This idea fits well in the *Smart Grid* context, ensuring a transparent, sustainable and environmental-friendly system operation that is cost and energy efficient, secure and safe.

**PALABRAS CLAVE:** Artificial Intelligence, AI, Big Data, Machine Learning, LSTM, Python, Smart Grids, Data Science, Battery management, ZigBee

# ÍNDICE DE DOCUMENTOS

- MEMORIA
- PRESUPUESTO
  - ANEXOS

# MEMORIA

# ÍNDICE DE LA MEMORIA

## Contenido

1	Introducción .....	7
1.1	Objetivos .....	7
1.2	Motivación .....	7
1.3	Antecedentes .....	7
1.4	Viabilidad.....	9
1.5	Presentación del sistema .....	10
2	Protocolo de comunicación inalámbrica.....	11
2.1.1	Estudio de alternativas.....	11
2.1.2	Selección del protocolo.....	11
2.2	Hardware empleado.....	13
2.3	Descripción de la red.....	13
	Coordinador .....	13
	Router.....	13
	End device .....	14
2.4	Modo de funcionamiento .....	14
2.5	Comunicación con el programa principal.....	15
3	Marco teórico. Inteligencia artificial .....	18
3.1	Machine Learning.....	18
3.2	Deep Learning .....	19
3.3	Redes Long Short-Term Memory (LSTM).....	21
4	Modelo predictivo de consumo .....	22
4.1	Introducción .....	22
4.2	Datos de entrenamiento .....	23
4.3	Análisis exploratorio de datos.....	24
4.3.1	Variable objetivo .....	24
4.3.2	Limpieza de los datos .....	24
4.3.3	Visualización y adaptación de los datos.....	27
4.4	Construcción del modelo .....	40



4.4.1	Convertir la serie temporal en un problema de aprendizaje supervisado.....	40
4.4.2	Preparación de los datos .....	46
4.4.3	Modelo LSTM .....	50
4.5	Verificación de resultados .....	53
4.6	Guardar modelo .....	58
5	Predicción de la generación .....	59
5.1	Planteamiento .....	59
5.2	Selección de la API.....	59
5.3	Uso de la API.....	61
5.4	Integración en Python .....	62
6	Dimensionamiento de la batería y la superficie de paneles fotovoltaicos .....	64
6.1	Introducción .....	64
6.2	Dimensionamiento de los paneles fotovoltaicos .....	64
6.3	Dimensionamiento de la batería.....	71
7	Algoritmo de gestión de energía.....	78
7.1	Introducción .....	78
7.2	Planteamiento .....	78
7.3	Diagrama de flujo .....	78
7.4	Código.....	80
	is_valley:.....	81
	adapt_data: .....	81
	set_irradiance_next_hour:.....	82
	setup_scheduler:.....	82
7.5	Caso de test.....	82
7.6	Resultados .....	83
8	Conclusión y líneas futuras.....	85
9	Bibliografía .....	<b>¡Error! Marcador no definido.</b>

# ÍNDICE DE FIGURAS

Figura 1. Esquema de Smart Grid.....	8
Figura 2. Ejemplo de Smart Grid [3] .....	9
Figura 3. Esquema general del proyecto.....	10
Figura 4. Ejemplo de red mallada ZigBee [6].....	14
Figura 5. Test de la comunicación entre los módulos.....	15
Figura 6. Preparación del paquete a enviar .....	16
Figura 7. Log de los datos a enviar a la BeagleBone .....	17
Figura 8. Archivo .csv generado de leer la comunicación inalámbrica .....	17
Figura 9. Machine Learning vs programación tradicional [8].....	18
Figura 10. Ejemplo de funcionamiento de Machine Learning .....	19
Figura 11. Estructura de una neurona con una única entrada.....	19
Figura 12. Algoritmo de Machine Learning vs red neuronal [9] .....	20
Figura 13. Entrenamiento de una red neuronal mediante gradiente descendiente estocástico [10] .....	21
Figura 14. Flujo de los datos.....	22
Figura 15. Primeras filas del set de datos de entrenamiento. ....	23
Figura 16. Potencia activa global entre 2007 y 2011 .....	28
Figura 17. Suma mensual de la potencia activa del set de datos de entrenamiento .....	28
Figura 18. Media mensual de la potencia activa del set de datos de entrnamiento. ....	29
Figura 19. Potencia activa global. Suma diaria.....	30
Figura 20. Potencia activa global. Media diaria .....	30
Figura 21. Tensión media diaria .....	31
Figura 22. Tensión media horaria durante una semana .....	31
Figura 23. Consumo mensual.....	33
Figura 24. Consumo por horas .....	33
Figura 25. Histograma del consumo diario .....	34
Figura 26. Matriz de correlación .....	36
Figura 27. Energía activa vs voltaje .....	37
Figura 28. Energía activa vs intensidad .....	37
Figura 29. Subconsumo 1 vs energía activa .....	38

Figura 30. Subconsumo 2 vs energía activa .....	38
Figura 31. Subconsumo 3 vs energía activa .....	39
Figura 32. Subconsumo 4 vs energía activa .....	39
Figura 33. Energía reactiva vs activa .....	40
Figura 34. Reformulación de la serie temporal a problema de aprendizaje supervisado .....	45
Figura 35. División entre datos de entrenamiento y validación [12] .....	49
Figura 36. Overfitting & Underfitting. [14] .....	51
Figura 37. Pérdidas del modelo durante el proceso iterativo .....	53
Figura 38. Predicción vs valor real .....	55
Figura 39. Predicción vs real para el modelo escogido, 200 primeras predicciones .....	58
Figura 40. Método GET 1 Hour of Hourly Forecasts de AccuWeather [4] .....	61
Figura 41. Obtención de la irradiancia a través de la API de AccuWeather .....	62
Figura 42. Media y percentil 75 del consumo a lo largo del día .....	65
Figura 43. Media por horas y percentil 75 del consumo a lo largo del día, potencia propuesta	67
Figura 44. Herramienta “año meteorológico típico”, PVGIS [15] .....	67
Figura 45. Año meteorológico típico, irradiancia global [15] .....	68
Figura 46. Irradiancia diaria en 2016, PVGIS .....	70
Figura 47. Consumo vs generación horaria, año típico .....	71
Figura 48. Saldo de potencia diario .....	72
Figura 49. Periodos horarios de facturación en la nueva factura. Fuente y elaboración: CNMC73	
Figura 50. Saldo de potencia diario excluyendo el periodo valle .....	75
Figura 51. Comparación horaria del saldo de potencia con la carga de la batería .....	76
Figura 52. Diagrama de flujo del programa .....	79
Figura 53. Funcionamiento de adapt_data_test .....	83
Figura 54. Decisión de cargar o no la batería (1 o 0) frente a consumo y generación fotovoltaica .....	84

# ÍNDICE DE FRAGMENTOS DE CÓDIGO

Fragmento de código 1. Lectura del puerto serie .....	16
Fragmento de código 2. Información del dataframe .....	25
Fragmento de código 3. Pandas.info tras arreglar las columnas .....	26
Fragmento de código 4. Suma de entradas nulas .....	26
Fragmento de código 5. Suma de entradas nulas tras interpolar .....	27
Fragmento de código 6. Resample del dataframe en meses .....	28
Fragmento de código 7. Resample del dataframe en horas .....	29
Fragmento de código 8. Manipulación de los datos .....	32
Fragmento de código 9. Adaptación de los datos. Resample en horas .....	35
Fragmento de código 10. Primeras 5 entradas .....	35
Fragmento de código 11. Ejemplo de serie temporal .....	42
Fragmento de código 12. Uso de pandas.shift .....	43
Fragmento de código 13. Ejemplo de dataframe tras adaptar la serie temporal .....	45
Fragmento de código 14. Ejemplo 2, serie temporal en problema supervisado .....	46
Fragmento de código 15. Dataframe escalado entre 0 y 1 .....	47
Fragmento de código 16. Series to supervised, resultado .....	48
Fragmento de código 17. Eliminar variables sin uso .....	48
Fragmento de código 18. División entre entrenamiento y test .....	49
Fragmento de código 19. Redimensionamiento del input a la red LSTM .....	50
Fragmento de código 20. Definición de la red LSTM .....	50
Fragmento de código 21. Definición de la red LSTM, 2 .....	51
Fragmento de código 22. Early Stopping .....	52
Fragmento de código 23. Entrenar la red .....	53
Fragmento de código 24. Resultados del modelo .....	54
Fragmento de código 25. Visualización de la predicción .....	54
Fragmento de código 26. Guardar el modelo .....	58
Fragmento de código 27. Uso de la API .....	63
Fragmento de código 28. Respuesta de la API (1) .....	63
Fragmento de código 29. Respuesta de la API (2) .....	63
Fragmento de código 30. Información de los datos de consumo por horas .....	64

---

Fragmento de código 31. Información del consumo por horas.....	66
Fragmento de código 32. Lectura de los datos de radiación .....	69
Fragmento de código 33. Diferencia de potencia por horas.....	73
Fragmento de código 34. Diferencia de potencia por horas, excluidas horas valle .....	74
Fragmento de código 35. Capacidad requerida y acumulación por horas .....	77

# 1 INTRODUCCIÓN

## 1.1 Objetivos

El objetivo de este TFM es llegar a una solución funcional que pueda asegurar el correcto funcionamiento de la comunicación inalámbrica entre los sensores de una *Smart Grid* y la BeagleBone para, posteriormente, tratar los datos obtenidos mediante técnicas de *Big Data* y *Machine Learning* con el fin de obtener un algoritmo que permita tomar decisiones respecto a la carga de unas baterías.

Para ello se proponen los siguientes objetivos:

- Selección del protocolo inalámbrico
- Creación de un modelo predictor de consumos
- Creación de un modelo predictor de generación
- Dimensionamiento de la batería y la superficie de paneles fotovoltaicos en base a los datos históricos
- Programación del algoritmo
- Contraste de resultados

## 1.2 Motivación

Los hechos que me han llevado a la realización de este trabajo son varios. Por un lado, al haber cursado la especialización en electrónica quería contribuir en la medida de lo posible en algún proyecto del departamento de ingeniería electrónica de la UPV.

Por otro lado, el hecho de trabajar en el sector de la información de la mano de unas prácticas a las que he accedido a través de la UPV me ha permitido dar los primeros pasos en el mundo del *Big Data* y *Machine Learning*. A título personal, pienso que las posibilidades de aplicación en la electrónica de potencia son enormes, y me gustaría tanto profundizar en el tema como aportar ideas nuevas.

## 1.3 Antecedentes

Estamos siendo testigos de la rápida transformación digital que estamos viviendo recientemente. La pandemia ha acelerado el desarrollo digital, y con ello, aparecen nuevos retos y posibilidades relacionados con la gestión de datos y la gestión de los ecosistemas que se pueden crear con ellos. La aparición del coche eléctrico también supone una revolución en el cambio de los patrones de consumo, que junto a la reciente crisis energética actual provoca un cambio de paradigma en el sector energético.

Una de las formas de atajar este desafío es hacer uso de las redes inteligentes o *Smart Grids*, que “es aquella que puede integrar de forma eficiente el comportamiento y las acciones de todos los usuarios conectados a ella, de tal forma que se asegure un sistema energético

sostenible y eficiente, con bajas pérdidas y altos niveles de calidad y seguridad de suministro” según [1].

La tendencia hacia la optimización y la seguridad, junto al creciente interés por el *Big Data*, aseguran que las *Smart Grids* pronto serán una necesidad no solo en el ámbito industrial.

El objetivo no es solo formar una red inteligente, sino implementar la idea de *Energy Management System* (EMS) para gestionar todos los datos recogidos por la red y actuar en consecuencia.

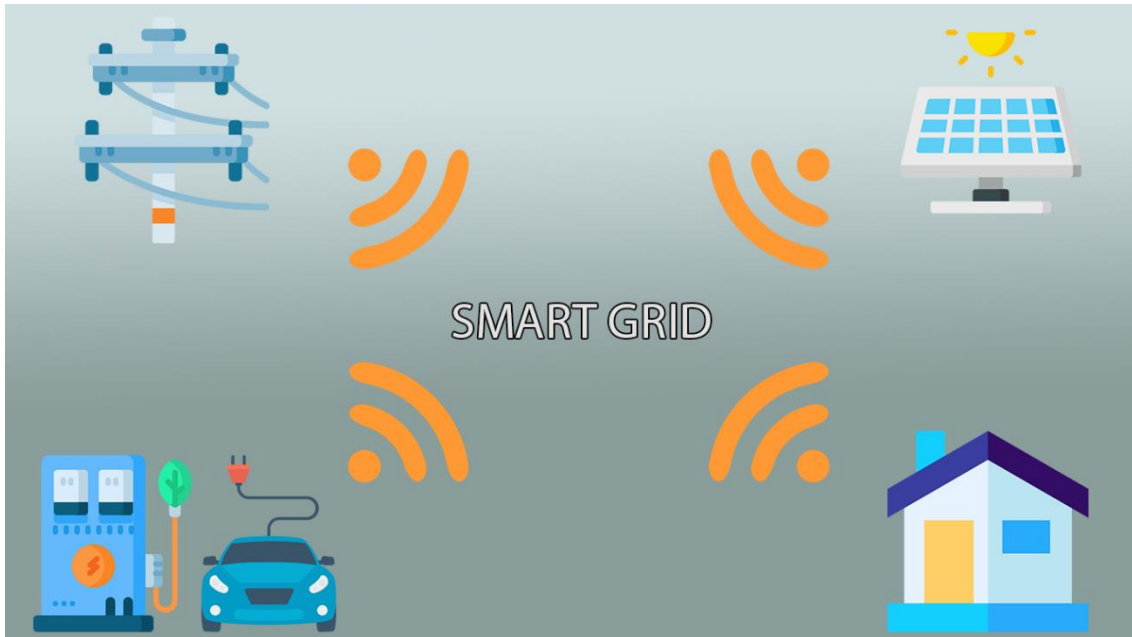


Figura 1. Esquema de Smart Grid

De forma adicional, los dispositivos que conforman la red inteligente deben asegurar el funcionamiento ininterrumpido, con protocolos robustos y mediante mecanismos que verifiquen el buen funcionamiento de estos. Empleando técnicas de inteligencia artificial, se pueden desarrollar aplicaciones que, a través de un set de datos de aprendizaje, puedan generar predicciones. Este último punto es de gran valor en la industria, ya que permite una mejor toma de decisiones.

Además, del análisis de los datos empleados pueden extraerse metodologías y conclusiones de gran valor para la realización del proyecto, en línea con la filosofía de “Data-driven decision-making” (toma de decisiones basadas en datos) que se define como “usar hechos, métricas y los datos para guiar las decisiones estratégicas empresariales” [2]. Importantes empresas tecnológicas siguen esta metodología, como *Google*, *Amazon* o *Tableau*.

Las técnicas de inteligencia artificial son muy recientes, pero gracias a la aparición de librerías de código abierto y la gran cantidad de aplicaciones que tienen en cualquier sector provocan que sea una de las grandes mejoras de la industria en los próximos años.

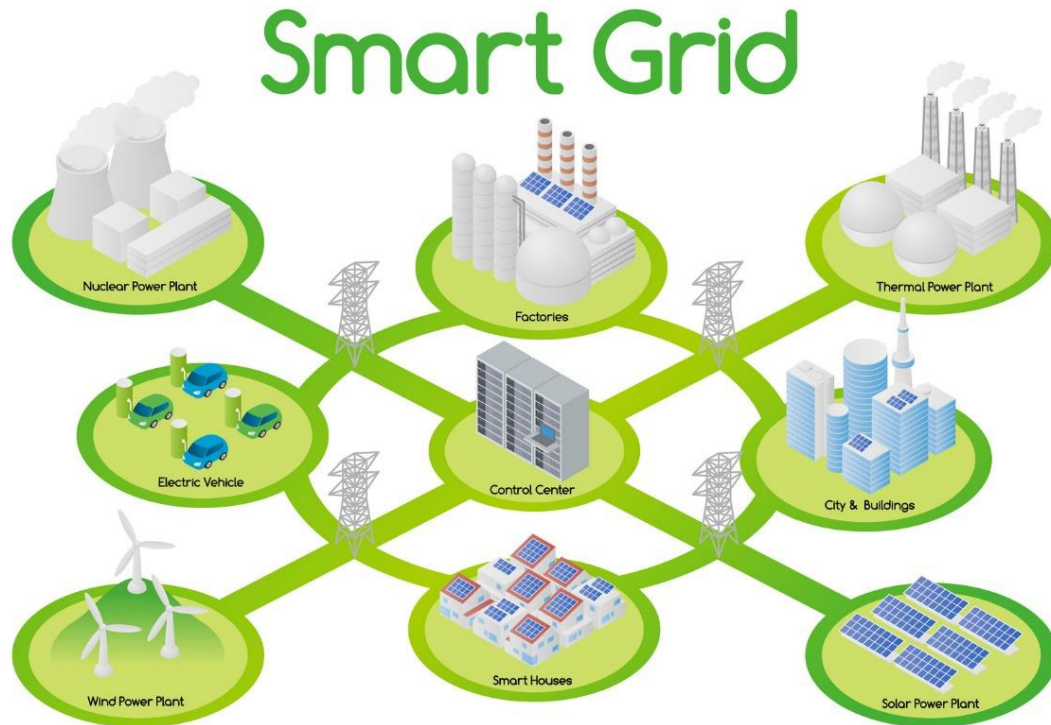


Figura 2. Ejemplo de Smart Grid [3]

El producto resultante de este TFM tiene un objetivo que responde a una necesidad de mercado: alargar la vida útil de las baterías y reducir los requisitos de capacidad. De esta manera, es posible seleccionar baterías más baratas para la misma aplicación y alargar su vida útil frente a metodologías convencionales, con lo que además de la ventaja económica que supone una batería más pequeña y de mayor vida útil, se contribuye al mantenimiento del medio ambiente ya que se generan menos desperdicios.

## 1.4 Viabilidad

Se ha recreado el funcionamiento del sistema final mediante el uso de una malla de sensores, conectados de forma inalámbrica a un ordenador de bajo coste. El sistema empleado ha sido *BeagleBone Black*. Este equipo se define como un “single board computer”, que al igual que su homólogo más popular, la Raspberry Pi, permite funcionar como un sistema empujado, pero con las virtudes de integrar un sistema operativo.

Para el desarrollo de la parte de inteligencia artificial, se ha hecho uso de la librería especializada en IA de código abierto *Keras*, que mediante su API de Python permite un uso sencillo sin necesidad de modelar la herramienta matemática al completo.

El algoritmo se ha programado en el popular lenguaje de programación *Python*. Para obtener los datos meteorológicos necesarios para estimar la generación fotovoltaica se ha hecho uso de la API de *AccuWeather* [4]

Para demostrar la viabilidad económica del proyecto, se ha realizado un estudio económico que puede encontrarse en el documento Presupuesto.



## 1.5 Presentación del sistema

El primer objetivo es lograr la transmisión de datos desde los sensores hasta la BeagleBone para introducir los datos en el modelo de predicción de consumos. El modelo previamente ha sido entrenado con el set de datos de entrenamiento, de manera que es capaz de generar predicciones sobre el consumo en la escala de tiempo indicada. Simultáneamente, se obtienen los datos meteorológicos a través de una API externa y que permiten realizar una estimación de la generación eléctrica en las fechas indicadas. Finalmente, se comparan generación y consumo para tomar una decisión sobre la carga de las baterías.

Considerando los elementos del sistema, la Figura 3 presenta un primer esquema explicativo en el que aparece el hardware principal y cómo debería ser el funcionamiento.

Para entrenar el modelo es necesario un gran volumen de datos de aprendizaje. En un proyecto real y completo se necesitaría una etapa previa de recogida de datos, que puede durar años. Debido a las limitaciones presentes en el ámbito educativo y a que no entra dentro del alcance de este TFM, los datos de entrenamiento se obtendrán de una base de datos pública y de uso libre alojada en [5]. La base de datos contiene información relativa al consumo doméstico de una vivienda durante 4 años.

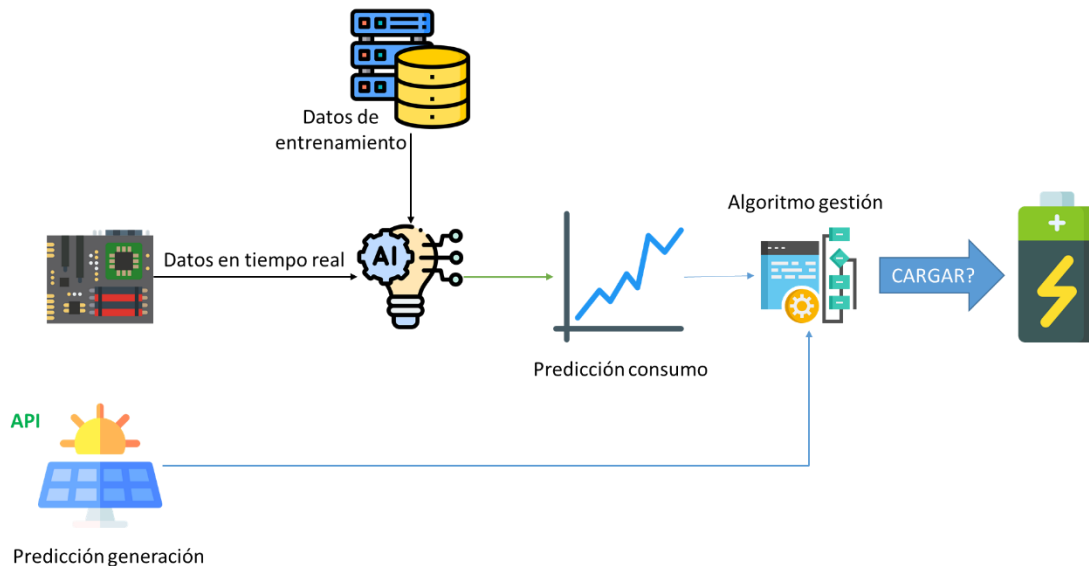


Figura 3. Esquema general del proyecto

La principal virtud de esta forma de proceder es que garantiza la reproducibilidad del producto. Con el único requisito de contar con el histórico de consumo, para desarrollar el proyecto en otra casuística se pueden seguir exactamente los mismos pasos. Además, los históricos de consumo son accesibles a día de hoy a través del número de factura y el código unificado de punto de suministro CUPS que proporcionan las comercializadoras eléctricas.

## 2 PROTOCOLO DE COMUNICACIÓN INALÁMBRICA

El primer problema que se plantea es de qué manera van a comunicarse los dispositivos. Hoy en día, existen infinidad de protocolos de comunicación inalámbrica con los que lograr la transmisión. La elección del protocolo no es trivial, ya que debe satisfacer las necesidades del problema.

Debido a que el presente trabajo se centra en la transmisión a partir de mecanismos de radiofrecuencia de corto-medio alcance, se descarta la alternativa de comunicación a través de WiFi, que, a pesar de ser muy interesante para este objetivo. Una alternativa a esta línea de trabajo es trabajar con comunicaciones a través de Internet para desarrollar un producto basado en servicios web, pero queda fuera del alcance de este proyecto.

### 2.1.1 Estudio de alternativas

Se ha realizado un estudio en base a las principales tecnologías empleadas en dispositivos *IoT* (*Internet of Things*). Si bien no se van a enviar datos a través de internet, los sensores empleados con este fin trabajan con la misma filosofía que se busca en este trabajo. Los protocolos más populares son los siguientes:

- Bluetooth: Bluetooth es el protocolo de comunicaciones de corto alcance por excelencia, ampliamente extendido en dispositivos móviles. Entre sus ventajas se incluyen la amplia adaptación del protocolo y la posibilidad de emplear *Bluetooth Low Energy (BLE)*, una subvariante muy indicada para dispositivos IoT de bajo consumo.
- ZigBee: muy similar al Bluetooth, pero funciona creando una red de área local (LAN) en malla. Si bien no logra velocidades tan altas como Bluetooth, funciona muy bien para redes de dispositivos que intercambian pocos datos. Recientemente, importantes tecnológicas del calibre de Amazon, Apple y Google han creado conjuntamente el proyecto *Project Connected Home over IP*, con ZigBee como principal protagonista, lo que va a potenciar el estándar.
- Z-Wave: similar a ZigBee, emplea una red LAN enfocada a la domótica. Las conexiones entre dispositivos tienen mayor alcance que las de ZigBee, pero a costa de un mayor consumo y limitando la señal a tres saltos entre dispositivos, lo que reduce el alcance de la red.

### 2.1.2 Selección del protocolo

Para la selección final solo se han considerado Bluetooth (en su modalidad Low Energy) y ZigBee. Se ha descartado emplear Bluetooth tradicional frente a BLE ya que esta última modalidad es

más conveniente para aplicaciones IoT porque permite la interconexión de más dispositivos, lo que no limita su escalabilidad.

La principal razón para seleccionar entre estos dos protocolos es el uso extendido de ambos para este tipo de aplicaciones frente al resto de alternativas, lo que permite escoger entre una amplia variedad de módulos comerciales y documentación relativa para su uso.

Entrando más en detalle en lo relativo a cada protocolo, las principales características de cada uno aparecen comparadas en la Tabla 1:

Criterio	Bluetooth BLE	ZigBee
Alcance típico	< 30m	< 100m
Tipo de red	Red y estrella	Red mallada
Tasa de envío	1 Mb/s	250 Kb/s
Consumo	Muy bajo	Bajo
Banda RF	2.4 GHz	2.4 GHz
Máximo de dispositivos conectados	Depende de la implementación	65000
Compatibilidad	Dispositivos compatibles con Bluetooth 4.0+ (muy extensa)	Dispositivos dedicados
Coste del chip	Bajo	Bajo

Tabla 1. Comparativa entre Bluetooth BLE y ZigBee

Si bien el Bluetooth ofrece ventajas muy significativas frente a ZigBee como la compatibilidad con dispositivos móviles, la alta tasa de transferencia y el bajo consumo, la característica más determinante que lleva a escoger ZigBee para esta aplicación es el alcance máximo y la robustez en la comunicación. Este es la ventaja de este protocolo, que lo hace especialmente adecuado en aplicaciones de domótica, industria y smart sensors. Además, la disposición en red mallada permite ampliar el alcance añadiendo más dispositivos a la red. También garantiza que no aparecerán problemas de escalabilidad si se desean añadir nuevos sensores a la red. La baja tasa de transferencia no supone un problema para esta aplicación ya que los datos a enviar no se presuponen complejos.

## 2.2 Hardware empleado

Para materializar las comunicaciones se ha de hacer uso de dispositivos compatibles con las tecnologías seleccionadas. Debido a que el protocolo de comunicación seleccionado ha sido ZigBee, el siguiente paso es encontrar un dispositivo comercial que permita la comunicación entre nodos y la BeagleBone.

Tras un exhaustivo análisis del mercado, se ha optado por hacer uso de los productos de la familia XBee del fabricante Digi International. Esta decisión ha sido motivada debido a su popularidad en proyectos de IoT y su facilidad de uso, así como la gran cantidad de documentación que hay disponible en internet que describe su uso.

Los dispositivos XBee comprenden una familia de módulos de conectividad inalámbrica por radiofrecuencia. Existen varias series de esta familia, siendo la serie XBee 3 la más reciente. Entre sus ventajas, cuenta con la posibilidad de usar distintos protocolos de comunicación como ZigBee o BLE, microcontrolador integrado para funcionar como unidad independiente y una curva de aprendizaje baja para su configuración gracias al uso del software XCTU, propietario de Digi. El lenguaje de programación empleado en el microcontrolador es MicroPython, una versión de nivel bajo del popular lenguaje pensada para dispositivos IoT.

Se ha seleccionado el *XBee 3 Zigbee 3 RF Module*, que es el indicado para trabajar con Zigbee.

## 2.3 Descripción de la red

El fabricante *Digi* dispone de tutoriales para configurar adecuadamente los transmisores en una misma red. Toda la documentación se encuentra disponible en [6].

El primer paso es configurar la red. Siguiendo los pasos del fabricante, se crea una red en la que se dispone de un módulo que funciona de *coordinador* y un segundo como *router*. Los diferentes modos de funcionamiento pueden consultarse en la documentación. De forma resumida, en una red *ZigBee* pueden encontrarse dispositivos de tres tipos: coordinadores, routers y *end devices*.

### ***Coordinador***

Toda red necesita de un coordinador, que es el encargado de distribuir las direcciones de cada dispositivo y controlar otras funciones que definen la red. El coordinador nunca puede entrar en modo de bajo consumo.

### ***Router***

El router es un nodo *ZigBee* con funcionalidad completa. Los routers son capaces de unirse a redes existentes, enviar y recibir información a través de la red. También pueden enviar paquetes a los nodos finales. No pueden entrar en modo de bajo consumo. Se pueden tener múltiples routers en una red. La principal funcionalidad del router es actuar como “repetidor” en la red, siendo capaz de actuar como mensajero en comunicaciones entre otros dispositivos de la red.

### End device

El *end device*, o nodo final, es una versión reducida del router. Estos pueden incorporarse a redes existentes, enviar y recibir información, pero no pueden actuar como mensajeros entre dispositivos. Sus requisitos energéticos son menores y pueden entrar en el modo de bajo consumo si no están en funcionamiento.

Este tipo de configuración permite formar redes como la que aparece en la Figura 4.

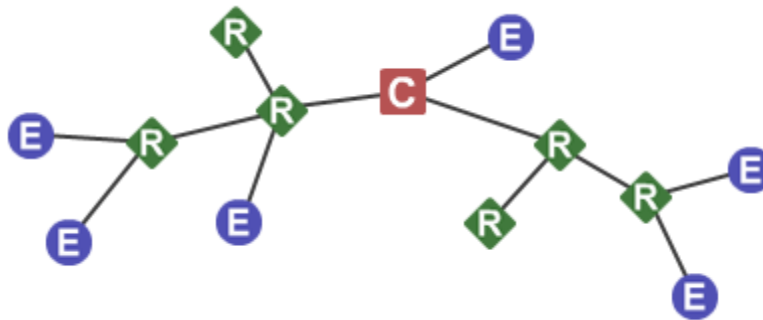


Figura 4. Ejemplo de red mallada ZigBee [6]

## 2.4 Modo de funcionamiento

Dado que este TFM es un prototipo, la red creada tiene como principal función emular la comunicación inalámbrica entre los sensores y la *BeagleBone*. Debido a los recursos limitados del proyecto, la red se compone de un dispositivo coordinador y un router. El funcionamiento deseado es el siguiente: el router debe enviar al coordinador los datos muestreados por los sensores.

Para simular los sensores, el router se ubica en una placa que dispone de un microcontrolador DSP. El DSP enviará por el puerto serie los datos de consumo de todos los sensores, con frecuencia de minutos, de manera que se simula el comportamiento real del dispositivo. El módulo *ZigBee* se comunica con el DSP mediante puerto serie. De esta manera, si se configura la comunicación en modo transparente, el comportamiento es el mismo que si la *BeagleBone* y el DSP se comunicasen por puerto serie mediante cables físicos.

El fabricante *Digi* ofrece en su web el software necesario para crear y configurar la red. Además, es posible verificar el funcionamiento de la comunicación a través de comunicación serie mediante los puertos USB.

A través de la consola integrada en la aplicación, se puede comprobar que si se envía una cadena de bytes desde uno de los módulos es recibido correctamente por el otro. La Figura 5 muestra el ejemplo de comunicación. Los colores del texto indican la dirección de la comunicación, comprobando que funciona de forma bidireccional.

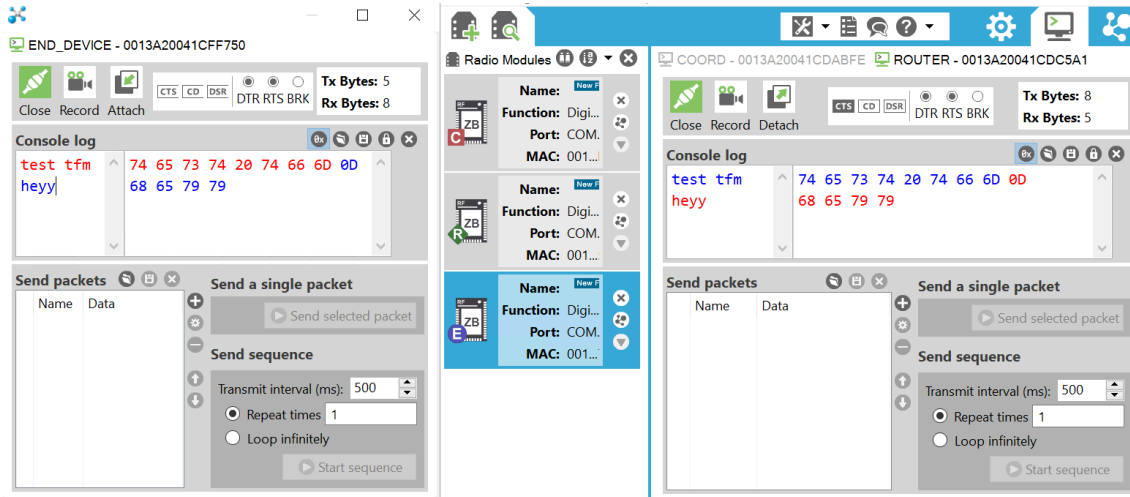


Figura 5. Test de la comunicación entre los módulos

## 2.5 Comunicación con el programa principal

A fin de facilitar el resto de apartados, debido a que la comunicación ya está verificada, todas las pruebas a lo largo de este TFM se van a realizar bajo una máquina virtual que emula la BeagleBone que se encargará de recibir los datos de consumo proporcionados por los sensores.

Para simular este comportamiento, ya que no existen sensores físicos para realizar las pruebas, puede usarse un dispositivo externo conectado al otro módulo inalámbrico, como otro PC o, incluso, el mismo PC desde fuera de la máquina virtual.

Ya planteada la estructura del hardware, hay que concretar en código qué función tiene cada una de las partes.

Por un lado, el PC que simula los sensores debe enviar al puerto serie del *End Device* los datos correspondientes a una medida, minuto a minuto. Este es el comportamiento real del sistema, ya que, aunque se introduzcan los valores en el puerto serie por software, en el funcionamiento real los sensores muestrean una vez por minuto las magnitudes requeridas. La frecuencia de muestreo y el formato de los datos viene explicado en el apartado Datos de entrenamiento.

En el otro lado de la comunicación, la BeagleBone debe leer los datos a través del módulo *XBee* que tiene conectado a través de uno de sus puertos series y escribir los valores recibidos en un archivo que sea reconocible después por el programa principal. Se ha decidido usar el formato *csv* (*comma separated values*) por su simplicidad y fácil integración con las librerías de Python.

Haciendo uso de la librería *Pyserial* [7] se puede acceder a los puertos series del dispositivo de forma sencilla a través de Python. El siguiente código muestra cómo es posible leer a través del puerto serie conectado al *Router XBee* los datos enviados desde el *End Device*, que no tiene conexión física con la máquina que ejecuta el código.

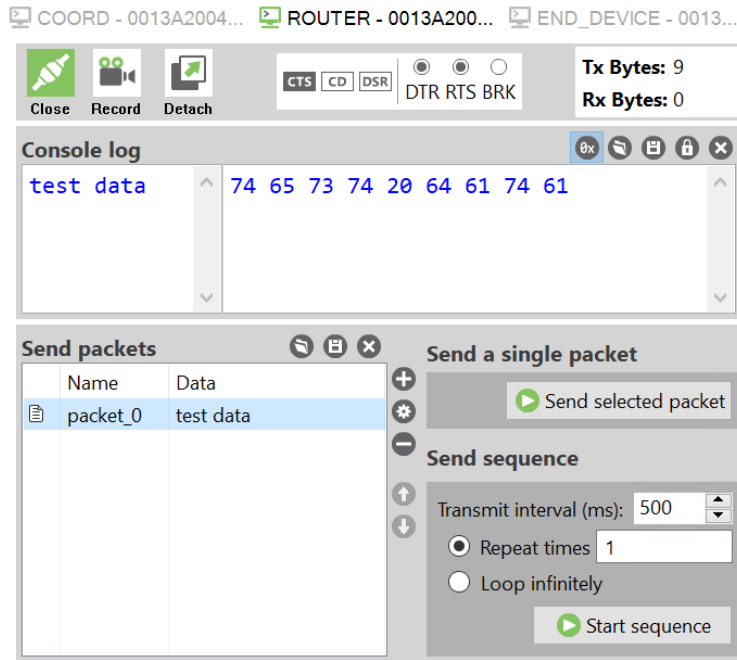


Figura 6. Preparación del paquete a enviar

```

1  import serial
2
3  serialPort = serial.Serial(
4      port='COM3')
5
6  while(1):
7      try:
8          x=serialPort.read()
9          print(x)
10     except:
11         break
12     serialPort.close()
13
Python - read_serial.py:10
b't'
b'e'
b's'
b't'
b' '
b'd'
b'a'
b't'
b'a'

```

Fragmento de código 1. Lectura del puerto serie

Con el siguiente código, se establece un bucle que lee datos del puerto serie siempre que entre un dato nuevo. De esta manera se valida la comunicación entre los dos dispositivos y es posible escribir en un csv cada vez que se detecte una nueva línea. La función `serialPort.read` lee la

cantidad de bytes especificados en el argumento (en este caso es un solo byte al no especificarse), pero puede usarse `read_until` para que lea hasta encontrar un carácter “/r”. Así, puede configurarse un nuevo paquete de datos de test en el envío con la misma estructura que el enviado por los sensores. La estructura de los datos enviados puede consultarse en el apartado Datos de entrenamiento.

Se prepara el formato de los datos a enviar y se programa un script de tal manera que cada vez que entren datos nuevos se añadan a un archivo .csv. Si el archivo no existe se crea automáticamente.

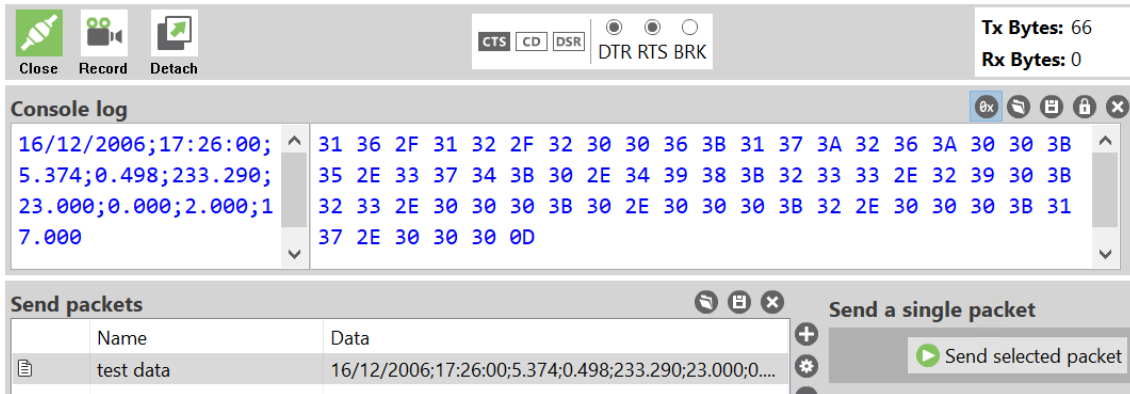



Figura 7. Log de los datos a enviar a la BeagleBone

El código puede encontrarse en el documento Anexo o en el repositorio [GitHub](#) del proyecto, bajo la carpeta `/BeagleBone`. El resultado de ejecutar el programa puede comprobarse al abrir el archivo csv que escribe, que contiene varias líneas en las que se ha enviado el mismo paquete de datos a modo de prueba.

 sensor\_data: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
16/12/2006;17:26:00;5.374;0.498;233.290;23.000;0.000;2.000;17.000
16/12/2006;17:26:00;5.374;0.498;233.290;23.000;0.000;2.000;17.000
16/12/2006;17:26:00;5.374;0.498;233.290;23.000;0.000;2.000;17.000
16/12/2006;17:26:00;5.374;0.498;233.290;23.000;0.000;2.000;17.000
```

Figura 8. Archivo .csv generado de leer la comunicación inalámbrica

Es importante remarcar que es importante que el script siempre se encuentre en ejecución, para no cerrar nunca la comunicación inalámbrica. Por tanto, de forma paralela al programa principal, debe programarse para que se arranque junto al encendido de la máquina.



### 3 MARCO TEÓRICO. INTELIGENCIA ARTIFICIAL

En este capítulo se va a desarrollar el modelo de inteligencia artificial encargado de analizar y resolver el problema propuesto. Estudios recientes han demostrado la efectividad de la inteligencia artificial en multitud de ámbitos, como pueden ser el márketing, finanzas, biología y, por qué no, electrónica de potencia.

En primer lugar, se va a desarrollar una breve introducción teórica al concepto para facilitar el seguimiento y la comprensión de los métodos empleados.

#### 3.1 Machine Learning

Para comprender qué es el *Deep Learning* primero hay que remontarse a los primeros pasos de la inteligencia artificial. El *Machine Learning* aplica técnicas de inteligencia artificial de manera que los sistemas adquieren la capacidad de aprendizaje a través de la experiencia obtenida. Este es un enfoque radicalmente diferente al modelo clásico de automatización basado en la algoritmia, en el que los problemas se resuelven de forma metódica a través de una secuencia de acciones preprogramada y conocida de antemano por el usuario que programa el algoritmo. Este nuevo paradigma supone que los programas que pueden recoger datos pueden usarlos para aprender de una manera muy parecida a la que aprenden los seres humanos. Mediante una serie de inputs y outputs preestablecidos en cada caso, el programa es capaz de realizar análisis para identificar patrones y desarrollar un modelo de forma automática.

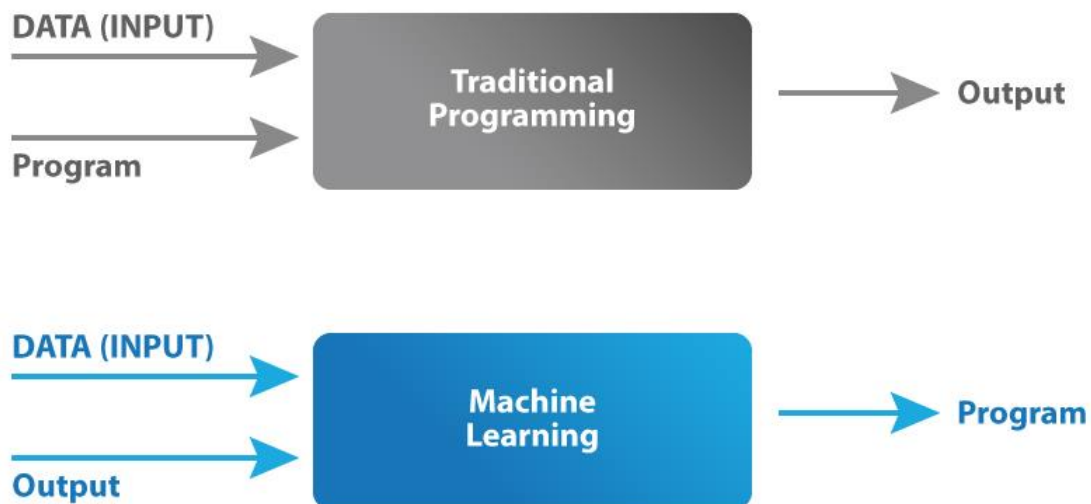


Figura 9. Machine Learning vs programación tradicional [8]

Un ejemplo práctico podría ser el siguiente. Se tiene un conjunto de mediciones de distintas especies de flores, por ejemplo, tamaño de los pétalos, color de la flor y longitud del tallo. Si se tienen los datos clasificados por categorías (por ejemplo, especie 1, 2 y 3), es posible desarrollar

un modelo que, una vez entrenado con los datos de entrada, sea capaz de clasificar las especies de flores en las categorías previamente asignadas dados unos datos de entrada

### Datos etiquetados

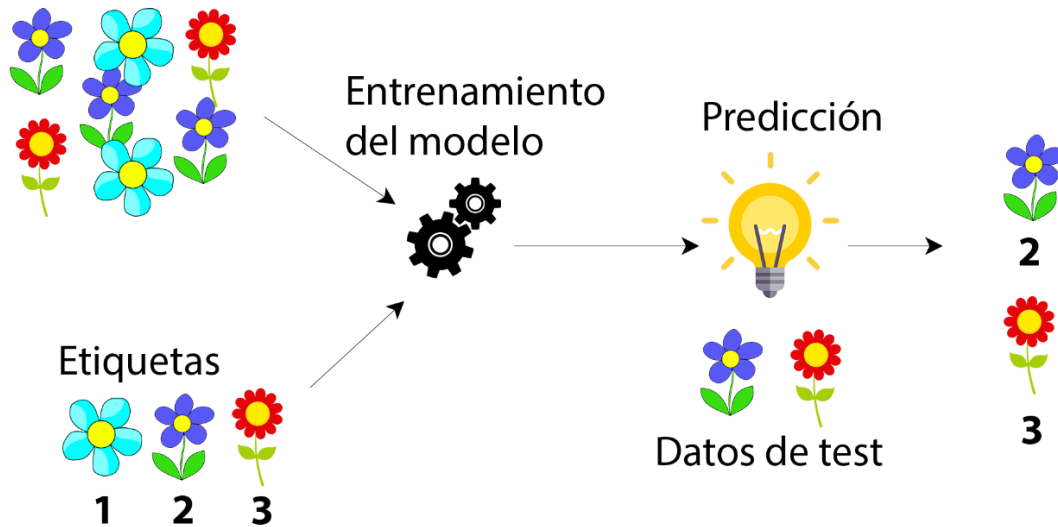


Figura 10. Ejemplo de funcionamiento de Machine Learning

El gran requisito que requiere esta tecnología es el volumen de datos y el análisis de estos. El análisis de los datos es el prerequisite fundamental antes de comenzar a desarrollar un modelo. El análisis de datos engloba un conjunto de procesos relacionados con la recolección de datos, limpieza del *dataset* y búsqueda de nuevas variables derivadas de las originales. Todos estos procesos ayudan a que las predicciones del modelo sean más precisas.

### 3.2 Deep Learning

El aprendizaje profundo, más conocido como *Deep Learning*, es un nuevo enfoque del *Machine Learning* caracterizado por emplear algoritmos inspirados en la estructura de las neuronas del cerebro humano. La unidad básica de la red neuronal es la neurona, que es capaz de ejecutar una única operación. El potencial de los algoritmos basados en redes neuronales proviene de la complejidad de las conexiones que forman sus neuronas.

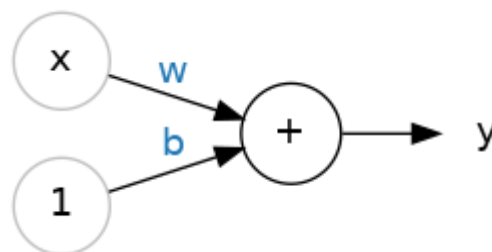


Figura 11. Estructura de una neurona con una única entrada

Cada neurona de la red representa una conexión entre su entrada y salida en forma de ecuación de recta, de la forma  $y = wx + b$ . La red neuronal es capaz de “aprender” ajustando los coeficientes de cada neurona.

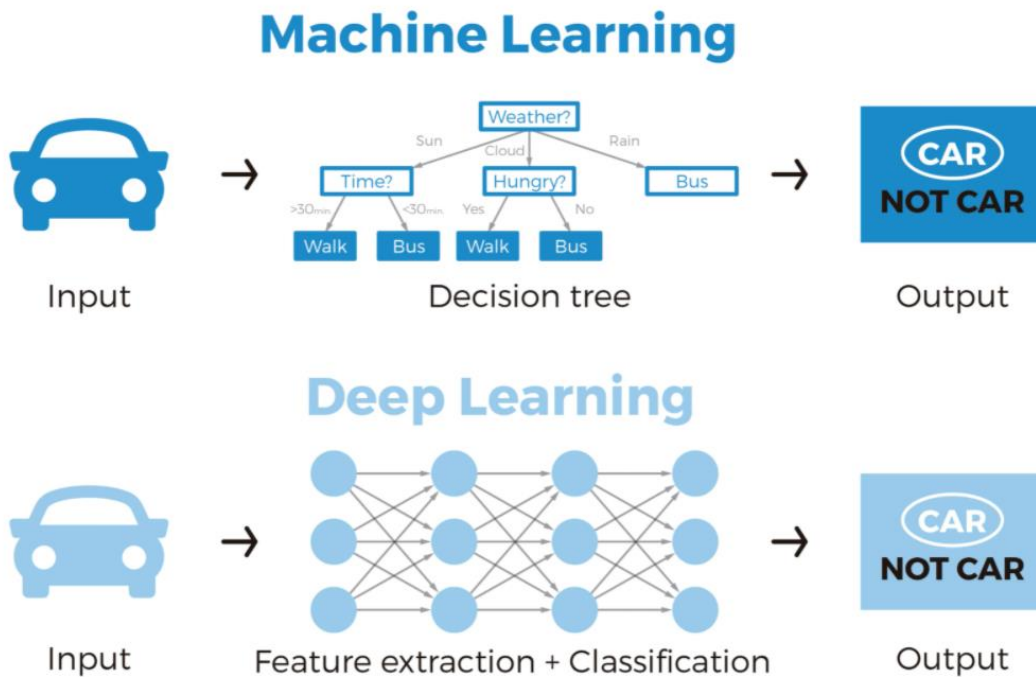


Figura 12. Algoritmo de Machine Learning vs red neuronal [9]

La red neuronal funciona alterando los valores de cada neurona de forma sucesiva en cada iteración hasta alcanzar un objetivo para el error del modelo.

La forma de alcanzarlo es mediante el trabajo del optimizador. El optimizador es un algoritmo que ajusta los pesos de cada neurona para minimizar las pérdidas del modelo. Las pérdidas del modelo se definen como la disparidad entre el valor real de la variable a predecir y el valor generado por el modelo.

La gran mayoría de optimizadores usados en *Deep Learning* pertenecen a la familia de los algoritmos de gradiente descendiente estocástico. Sin entrar en demasiados detalles sobre la herramienta matemática, son algoritmos iterativos que entrenan la red neuronal en sucesivos pasos. Cada paso se desarrolla de la siguiente manera:

1. Muestrear parte de los datos de entrenamiento y realizar predicciones.
2. Medir el error entre las predicciones y el valor real.
3. Ajustar los pesos en una dirección que minimice el error.

Estos tres pasos se realizan una y otra vez hasta reducir el error por debajo de un umbral deseado o hasta que no disminuya más con cada iteración. Cada iteración se conoce como “batch”, mientras que todo un proceso con unos datos de entrenamiento se llama “epoch”. El número de epochs para los que se entrena la red es equivalente al número de veces que la red va a observar todos los datos de ejemplo.

La animación de la Figura 13 representa de manera gráfica todo este proceso. El conjunto de los puntos de color rojo pálido representa todo el conjunto de datos de entrenamiento, mientras que cada grupo de puntos rojo sólido es un batch. En cada iteración los pesos de la red neuronal (W, b) son ajustados hacia sus valores correctos en ese batch.

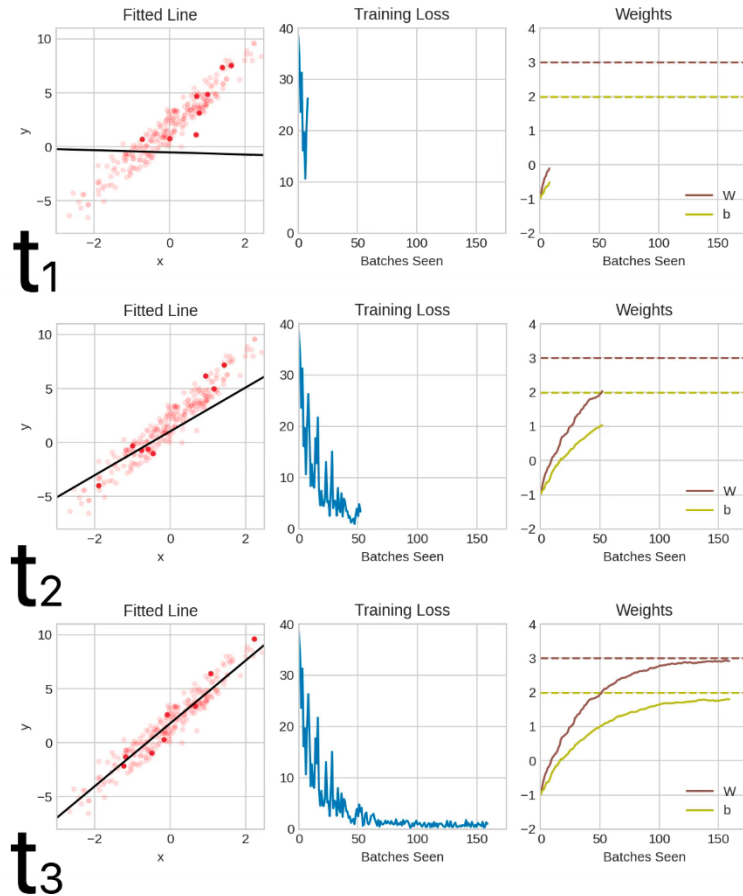


Figura 13. Entrenamiento de una red neuronal mediante gradiente descendente estocástico [10]

Eventualmente, la línea converge hasta su mejor ajuste. Puede observarse que el error se va reduciendo a medida que los pesos convergen hacia sus valores reales.

### 3.3 Redes Long Short-Term Memory (LSTM)

Las redes LSTM son un tipo de redes neuronales que se caracterizan por ser especialmente buenas en aprender secuencias. No es objeto de este TFM describir el complejo funcionamiento de la herramienta matemática de este tipo de redes, así que, *grosso modo*, la gran ventaja que tienen frente a las redes tradicionales es que disponen de bloques de memoria y pueden procesar no únicamente puntos de datos, sino secuencias enteras de datos en forma de vector. Es por ello por lo que las redes LSTM son ampliamente utilizadas en texto predictivo, detección del habla y en general cualquier tipo de datos que se desarrolle en el dominio del tiempo.

El uso de una red LSTM es muy adecuado para este tipo de problema ya que se trata de un problema secuencial en el que se dispone de una gran cantidad de datos.

## 4 MODELO PREDICTIVO DE CONSUMO

### 4.1 Introducción

A fin de construir un modelo fiable, la arquitectura empleada por la red será del tipo LSTM. Esta elección se decide en base a los criterios previamente explicados.

En primer lugar es necesario decidir cómo y en qué formato se van a obtener los datos. La Figura 14 representa el flujo de los datos desde la adquisición hasta la introducción en el modelo.

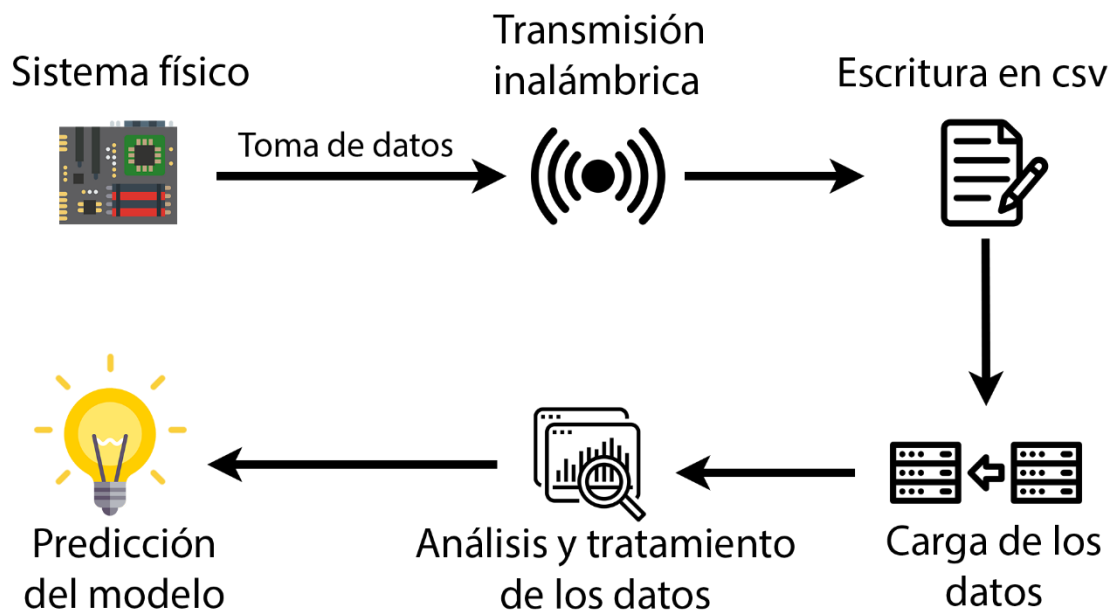


Figura 14. Flujo de los datos

Con la comunicación funcionando, la primera decisión fundamental es elegir cómo se van a enviar los datos. También hay que decidir qué variables se van a enviar y con qué frecuencia. Debido a que el modelo se va a entrenar mediante pasos temporales de 1h (justificación en el apartado Algoritmo de gestión de energía), los datos que se introduzcan posteriormente para generar la predicción deben seguir el mismo formato.

Es importante conocer de antemano qué variables pueden ser de interés, ya que sensor y enviar datos de más supone un aumento de presupuesto. Si bien esto no puede conocerse de antemano antes de realizar un análisis exploratorio de datos, conocer bien el sistema con el que se va a trabajar es fundamental para realizar la decisión correcta. Debido a que ya se dispone de un set de datos de entrenamiento en el que las variables de entrada ya están definidas, las variables que necesitan muestrearse son las mismas en este caso. En un proyecto completo, debería realizarse un estudio para determinar que variables son las más adecuadas para realizar la predicción.

## 4.2 Datos de entrenamiento

El set de datos de entrenamiento, disponible en [5], corresponde a las medidas de consumo eléctrico en una vivienda durante 4 años. El archivo con extensión .txt contiene 2075259 muestras tomadas entre diciembre de 2006 y noviembre de 2010. Se proporcionan diferentes magnitudes y hasta 3 subconsumos diferentes.

Examinando los datos haciendo de la librería de tratamiento de datos *Pandas*, se pueden observar las variables incluidas en el *dataset*. Para trabajar con los datos se hace uso del IDE de programación *JupyterLab*, que tiene la particularidad de que puede ejecutar bloques de código por separado, sin necesidad de completar el script entero. Es de gran utilidad en tareas de depuración y es ampliamente usado en proyectos de *Big Data*. Además, es de código abierto y tiene una comunidad enorme en la que encontrar todo tipo de documentación o funciones añadidas.

Las tres líneas que aparecen en la Figura 15 permiten observar las 5 primeras filas del set de datos:

```
import pandas as pd

df = pd.read_csv('household_power_consumption.txt', sep=';', low_memory=False)

df.head()
```

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3
0	16/12/2006	17:24:00	4.216	0.418	234.840	18.400	0.000	1.000	17.0
1	16/12/2006	17:25:00	5.360	0.436	233.630	23.000	0.000	1.000	16.0
2	16/12/2006	17:26:00	5.374	0.498	233.290	23.000	0.000	2.000	17.0
3	16/12/2006	17:27:00	5.388	0.502	233.740	23.000	0.000	1.000	17.0
4	16/12/2006	17:28:00	3.666	0.528	235.680	15.800	0.000	1.000	17.0

Figura 15. Primeras filas del set de datos de entrenamiento.

El *dataset* incluye información relativa a la fecha y hora de la medición, energías activa y reactiva globales (de toda la vivienda) e información sobre la potencia en 3 nodos diferentes. Variable por variable, el *dataset* queda descrito de la siguiente manera:

1. Date: Fecha de la medición en formato dd/mm/aaaa.
2. Time: Hora, minuto y segundos de la toma en formato hh:mm:ss. La frecuencia de las muestras viene en minutos por lo que los segundos no son relevantes.
3. Global\_active\_power: Potencia activa consumida de media por cada minuto (kW).
4. Global\_reactive\_power: Potencia reactiva consumida de media por cada minuto (kVAr).
5. Voltage: Media de la tensión en un minuto (V).
6. Global\_intensity: La intensidad media durante un minuto (A).
7. Sub\_metering\_1: Medida de la energía activa consumida (Wh) en un punto de consumo. El nodo 1 corresponde a la cocina, que dispone de un lavavajillas, un horno y un microondas.
8. Sub\_metering\_2: Medida de la energía activa consumida (Wh) en un punto de consumo. El nodo 2 corresponde al cuarto de la colada, que contiene una lavadora, secadora, un congelador y un punto de luz.

9. Sub\_metering\_3: Medida de la energía activa consumida (Wh) en un punto de consumo. Corresponde al consumo de un calentador de agua eléctrico y un aire acondicionado.

Es necesario realizar un correcto tratamiento de los datos contenidos en el archivo antes de entrenar al modelo.

### 4.3 Análisis exploratorio de datos

El análisis exploratorio de datos consiste en una primera investigación sobre los datos, a fin de descubrir patrones, anomalías, plantear hipótesis con la ayuda de herramientas estadísticas y representaciones gráficas.

Entender los datos e intentar extraer conclusiones antes de empezar a trabajar con el modelo es vital para lograr buenos resultados.

#### 4.3.1 Variable objetivo

Todavía no se ha decidido la variable objetivo del modelo. Se desea predecir el consumo eléctrico de una vivienda para el día siguiente en el que se realiza la medida. Todo apunta a que la variable de interés es la potencia activa, ya que, por ejemplo, conocer la potencia demandada por horas es gran interés en la industria. Sin embargo, en esta aplicación, se desea conocer el consumo diario de una vivienda, es decir, la energía. Será necesario adaptar los datos de entrenamiento a fin de obtener la medida de la energía, que no aparece en el conjunto original.

#### 4.3.2 Limpieza de los datos

El primer paso antes de comenzar a trabajar con los datos es asegurarse de la calidad del *dataset*. Desgraciadamente, es muy común que las bases de datos con las que se elaboran los modelos predictivos contengan entradas erróneas, nulas o formatos no deseados. Es trabajo del ingeniero asegurarse de que el *dataset* esté limpio y sin errores.

Con la función `pandas.dataframe.info()` puede obtenerse información básica sobre los datos recién cargados:

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2075259 entries, 0 to 2075258
Data columns (total 9 columns):
#   Column                Dtype
---  -
0   Date                  object
1   Time                  object
2   Global_active_power  object
3   Global_reactive_power object
4   Voltage               object
5   Global_intensity     object
6   Sub_metering_1       object
7   Sub_metering_2       object
8   Sub_metering_3       float64
dtypes: float64(1), object(8)
memory usage: 142.5+ MB
```

*Fragmento de código 2. Información del dataframe*

Gracias a esta función se conoce que los formatos con los que se han cargado las variables no son los esperados. La única que parece que ha respetado el formato correcto es la columna 8, Sub\_metering\_3, que se ha cargado en formato float64. Las demás columnas presentan un Dtype igual a "object", lo que quiere decir que se han pasado como cadena de texto o hay tipos mixtos en la columna.

Es necesario corregir los formatos de los datos para poder emplear funciones estadísticas. Se desea que los valores de las mediciones se presenten en número decimal, es decir, float64. Por el contrario, para las columnas correspondientes al tiempo es conveniente usar el formato de fecha que trae la librería Pandas. Si se transforman las fechas, en cadena de texto, a formato "datetime" es posible utilizar muchas de las funciones integradas en esta librería y que son de gran interés para el análisis. Esta práctica de adaptar el formato de las fechas es comúnmente conocida como "date parsing".

Puede realizarse a posteriori con el objeto *dataframe* ya cargado, pero es más rápido incluirlo en los argumentos de la función que lee el archivo .txt:



```
df = pd.read_csv('household_power_consumption.txt', sep=';',
                 parse_dates={'Fecha' : ['Date', 'Time']},
                 infer_datetime_format=True,
                 low_memory=False,
                 na_values=['nan', '?'],
                 index_col='Fecha')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2075259 entries, 2006-12-16 17:24:00 to 2010-11-26 21:02:00
Data columns (total 7 columns):
#   Column                Dtype
---  -
0   Global_active_power    float64
1   Global_reactive_power  float64
2   Voltage                float64
3   Global_intensity       float64
4   Sub_metering_1         float64
5   Sub_metering_2         float64
6   Sub_metering_3         float64
dtypes: float64(7)
memory usage: 191.2 MB
```

Fragmento de código 3. Pandas.info tras arreglar las columnas

Modificando los argumentos de la función es posible solucionar el problema de los formatos en una sola instrucción. Se han realizado dos procesos importantes. El primero, agrupar las columnas “Date” y “Time”, correspondientes a la fecha y hora de cada muestra, y convertirlas en una única columna en formato *datetime* que se ha llamado “Fecha”. Además, se ha indicado que sea el índice de la tabla, ya que cada fecha es única y puede servir como identificador de cada fila. El segundo, indicar a la función que las entradas nulas se van a encontrar como “nan” (not a number) o “?”. De esta manera, configura las entradas nulas con formato NaN, que no se tienen en cuenta, y no como “string”, que es lo que provocaba que las columnas estuviesen formadas por entradas de diferentes formatos.

Se puede comprobar el número de entradas nulas en el *dataframe*, mediante la función `pandas.dataframe.isnull().sum()`.

```
df.isnull().sum()

Global_active_power    25979
Global_reactive_power  25979
Voltage                25979
Global_intensity       25979
Sub_metering_1         25979
Sub_metering_2         25979
Sub_metering_3         25979
dtype: int64
```

Fragmento de código 4. Suma de entradas nulas

Existen un total de 25979 entradas nulas. Al ser el mismo número para todas las variables, es muy probable que se deba a que en determinadas filas todas las variables sean nulas.

Es imposible entrenar al modelo si existen entradas nulas en los datos. Para afrontar este problema, se pueden tomar diferentes soluciones. La más sencilla es eliminar todas las filas en las que existan valores nulos, pero eso supone perder una gran cantidad de datos. Otra técnica habitual consiste en reemplazar los valores nulos con la media de todos los valores de esa columna. Para este caso, no es una buena solución ya que no respetaría la tendencia de la gráfica del consumo. Debido a la naturaleza del patrón de consumo, la mejor opción para no distorsionar demasiado la naturaleza de los datos es interpolar entre los dos valores contiguos y asignar el nuevo valor a la entrada nula.

Para realizar esta operación, se puede emplear la función integrada en la librería *Pandas*:

```
df = df.interpolate()

df.isnull().sum()

Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
```

*Fragmento de código 5. Suma de entradas nulas tras interpolar*

Tras ejecutar la instrucción se comprueba que ya no existen valores nulos.

### **4.3.3 Visualización y adaptación de los datos**

Es conveniente visualizar los datos antes de empezar a extraer conclusiones. Para comenzar, se va a graficar la potencia activa global. El resultado aparece en la Figura 16:

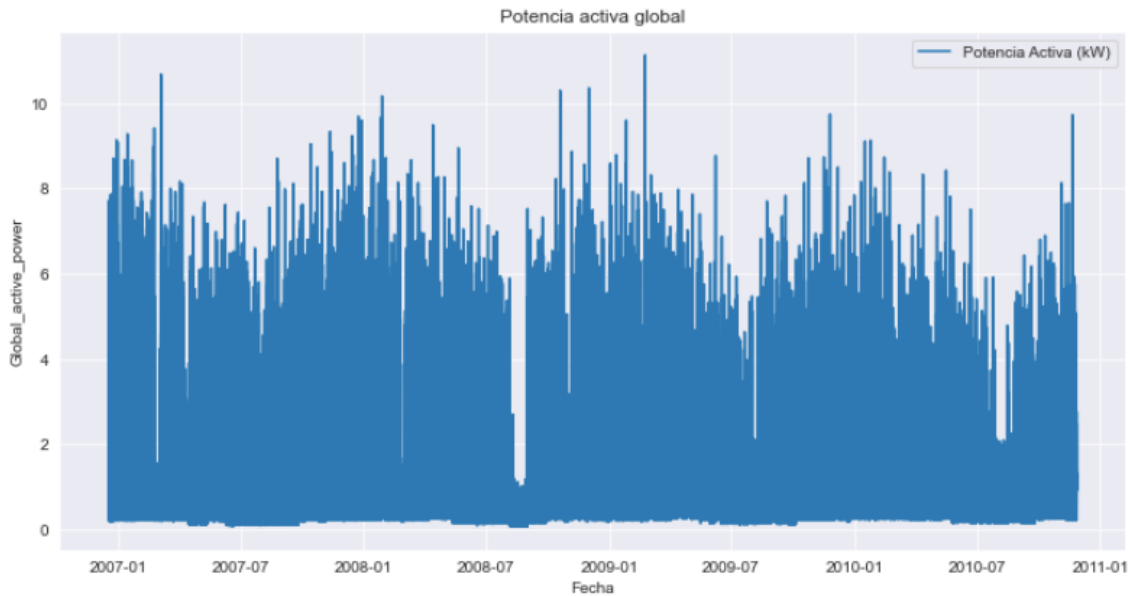


Figura 16. Potencia activa global entre 2007 y 2011

Debido a la gran cantidad de datos existentes, representar toda la potencia activa recogida minuto a minuto durante 4 años no ofrece ninguna información relevante. Los datos pueden condensarse si se agrupan con otra frecuencia temporal. Este proceso se puede realizar fácilmente mediante función *resample* de la librería *Pandas*. Se puede obtener información valiosa agrupando las variables en días o meses. El siguiente código agrupa los datos de la potencia activa global en meses, tanto en suma como media mensual.

```
df.Global_active_power.resample('M').sum().plot(title='Global_active_power suma diaria')
plt.tight_layout()
plt.show()

df.Global_active_power.resample('M').mean().plot(title='Global_active_power media diaria', color='red')
plt.tight_layout()
plt.show()
```

Fragmento de código 6. Resample del dataframe en meses

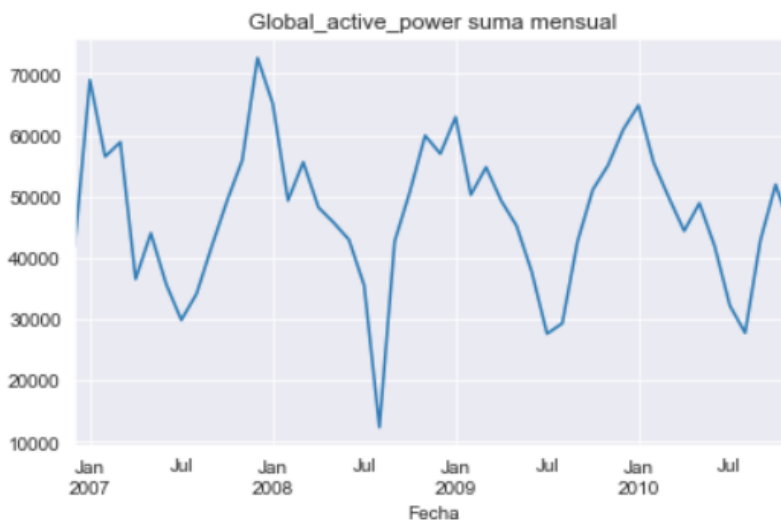


Figura 17. Suma mensual de la potencia activa del set de datos de entrenamiento

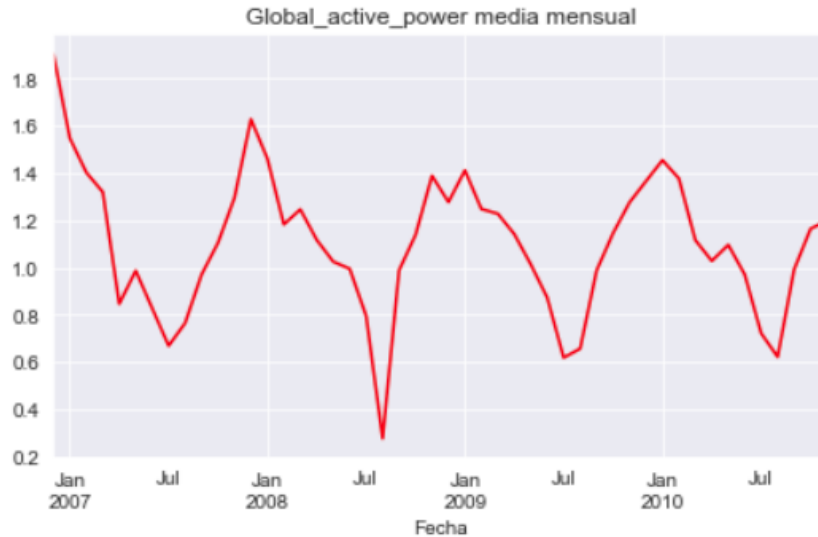


Figura 18. Media mensual de la potencia activa del set de datos de entrenamiento.

Agrupando los datos por meses la información aparece más clara. Se aprecian patrones de consumo anuales, lo cual entra dentro de lo esperado. Aunque es interesante conocer la tendencia de consumo anual, para este tipo de aplicación, en la que se debe decidir sobre la carga de unas baterías, no tiene sentido plantearse una escala temporal tan grande. Se debe hacer el análisis de los datos con la misma frecuencia con la que posteriormente trabajará el modelo predictivo. Se decide trabajar en una escala temporal de horas. Esta decisión está explicada más adelante, en el apartado Algoritmo de gestión de energía

```
#Agrupacion por horas
df.Global_active_power.resample('h').sum().plot(title='Global_active_power suma po horas')
plt.tight_layout()
plt.show()

df.Global_active_power.resample('h').mean().plot(title='Global_active_power media por horas', color='red')
plt.tight_layout()
plt.show()
```

Fragmento de código 7. Resample del dataframe en horas

Adaptando el código anterior para agrupar los datos en días el resultado es el que aparece en la Figura 19 y Figura 20. Se comprueba tanto las medias como la suma por horas presentan una estructura parecida.

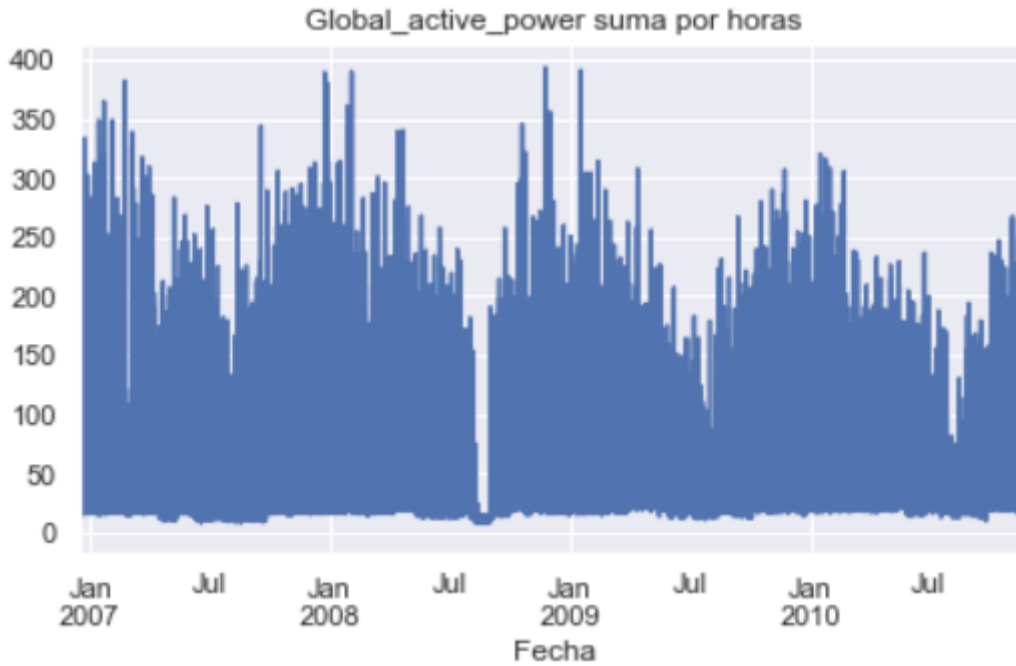


Figura 19. Potencia activa global. Suma diaria

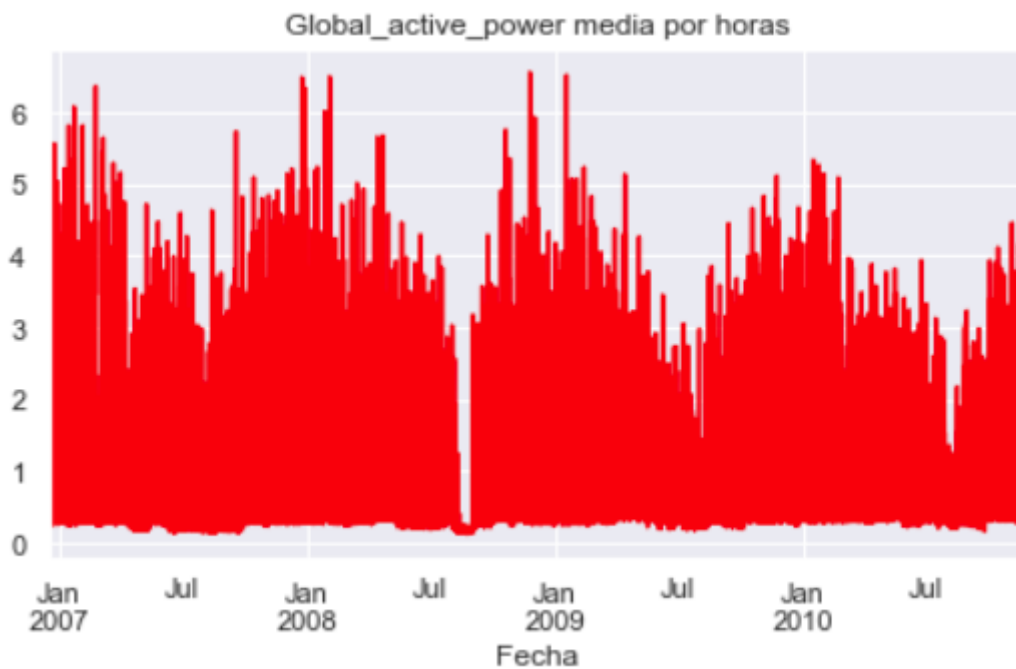


Figura 20. Potencia activa global. Media diaria

De esta manera, no solo se obtiene información más adecuada para el tipo de problema que se quiere abordar, sino que también se facilita el tiempo de ejecución de todo el proceso ya que se ha reducido considerablemente el número de filas de datos.

No todas las variables pueden estudiarse de la misma manera. Para la potencia no tiene sentido realizar la suma diaria. Se ha representado con el fin de comprobar que la estructura de los datos era similar.

Representando la tensión media horaria:

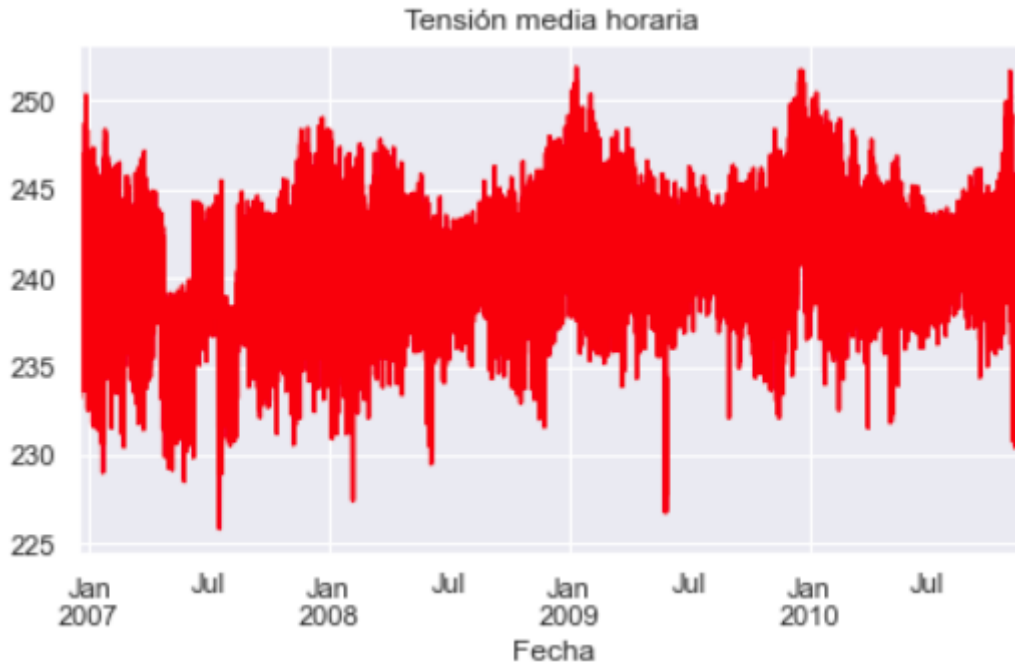


Figura 21. Tensión media diaria

Si se hace zoom sobre una ventana temporal más pequeña pueden extraerse conclusiones sobre la tensión:



Figura 22. Tensión media horaria durante una semana

No se ha representado la suma horaria porque no tiene ningún sentido para esta magnitud. Se aprecia que, aunque existan horas en las que la tensión toma valores anómalos, por lo general no oscila demasiado alrededor de su media, por lo que podría ser poco relevante para el modelo. Además, hay que atender a las condiciones físicas del problema, no únicamente centrarse en la herramienta matemática. Al ser un consumo doméstico puede esperarse que la tensión sea

relativamente estable, ya que existen herramientas que velan por ello. No debería guardar demasiada relación con la variable objetivo, como se verá más adelante.

Uno de los aspectos que llama la atención de este set de datos es que se ofrece información sobre 3 puntos de consumo y la potencia global, pero no de la energía consumida de forma global. A priori, uno podría pensar que la suma de los tres consumos es la energía total consumida, pero cuando se trabaja con sets de datos que no son propios es conveniente asegurarse de cualquier tipo de suposición. Por suerte, se puede comprobar si la suma de los consumos coincide con el equivalente en energía. Mediante el siguiente código, se verifica que la suma no coincide y se crea una variable nueva para el medidor restante:

```
#Cambio de unidades a KWh
df['Sub_metering_1'] = df['Sub_metering_1']/1000
df['Sub_metering_2'] = df['Sub_metering_2']/1000
df['Sub_metering_3'] = df['Sub_metering_3']/1000

df['Suma consumos 123'] = df['Sub_metering_1'] + df['Sub_metering_2'] + df['Sub_metering_3']
df['Potencia a energia'] = df['Global_active_power']/60
df['Sub_metering_4'] = df['Potencia a energia']-df['Suma consumos 123']

#Agrupacion diaria

plt.figure(figsize = (15,10))
df['Sub_metering_1'].resample('M').sum().plot(legend=True)
df['Sub_metering_2'].resample('M').sum().plot(legend=True)
df['Sub_metering_3'].resample('M').sum().plot(legend=True)
df['Sub_metering_4'].resample('M').sum().plot(legend=True)
df['Potencia a energia'].resample('M').sum().plot(title='Consumo mensual KWh', legend=True, label='Energia total')
plt.show()
```

*Fragmento de código 8. Manipulación de los datos*

Se ha creado una columna nueva correspondiente al medidor número 4, derivada de la resta de la energía total menos la suma de los tres consumos. La energía total se ha obtenido a partir de la potencia, que al ser la potencia media en un minuto, la energía consumida en Wh en ese minuto puede obtenerse como:

$$E(kWh) = kWmin \cdot \frac{1000}{60}$$

Para que pueda apreciarse el resultado se ha representado la suma mensual de cada consumo en la Figura 23:

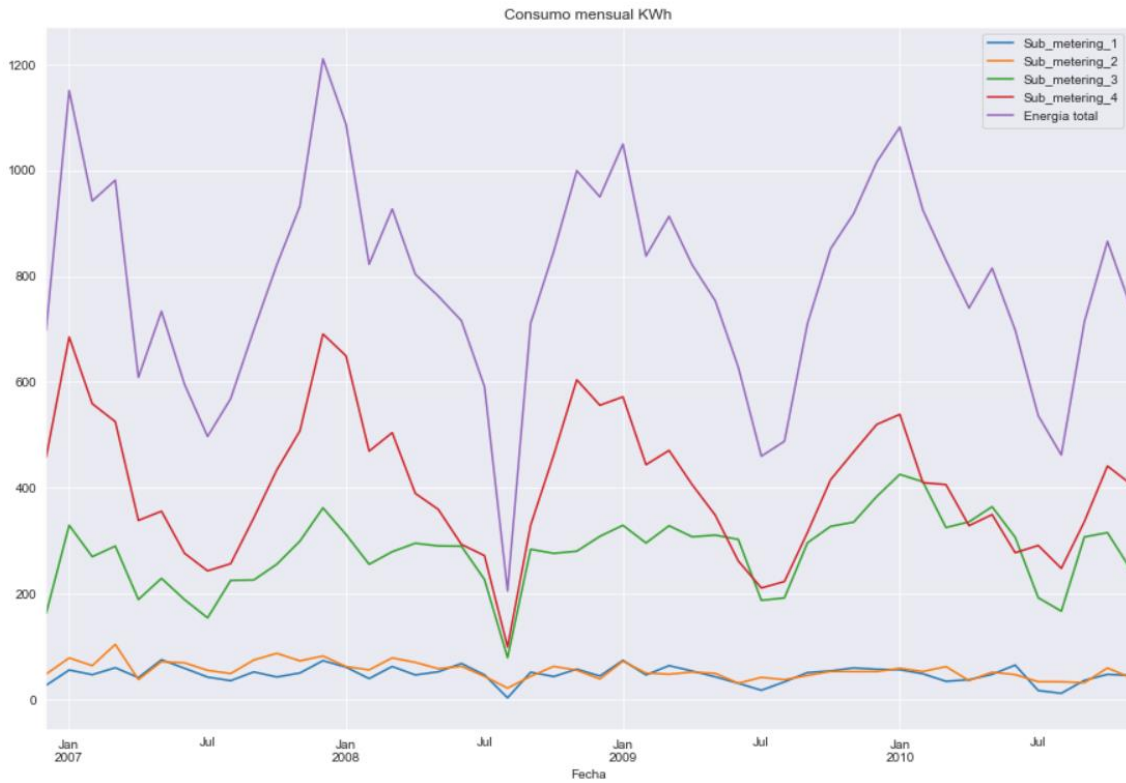


Figura 23. Consumo mensual

Se comprueba que la suma de los 3 nodos originales no coincide con la medida de la energía que se obtiene de la potencia activa y que el nodo 4 añadido a posteriori forma una parte importante del consumo. La suma de la energía por horas se tiene en la Figura 24:



Figura 24. Consumo por horas



También puede representarse un histograma del consumo diario para entender mejor cómo se distribuyen los valores. La forma de la gráfica es algo similar a una distribución normal. También se puede observar que para la mayoría de los días el consumo se encuentra entre 20 y 32 kWh.

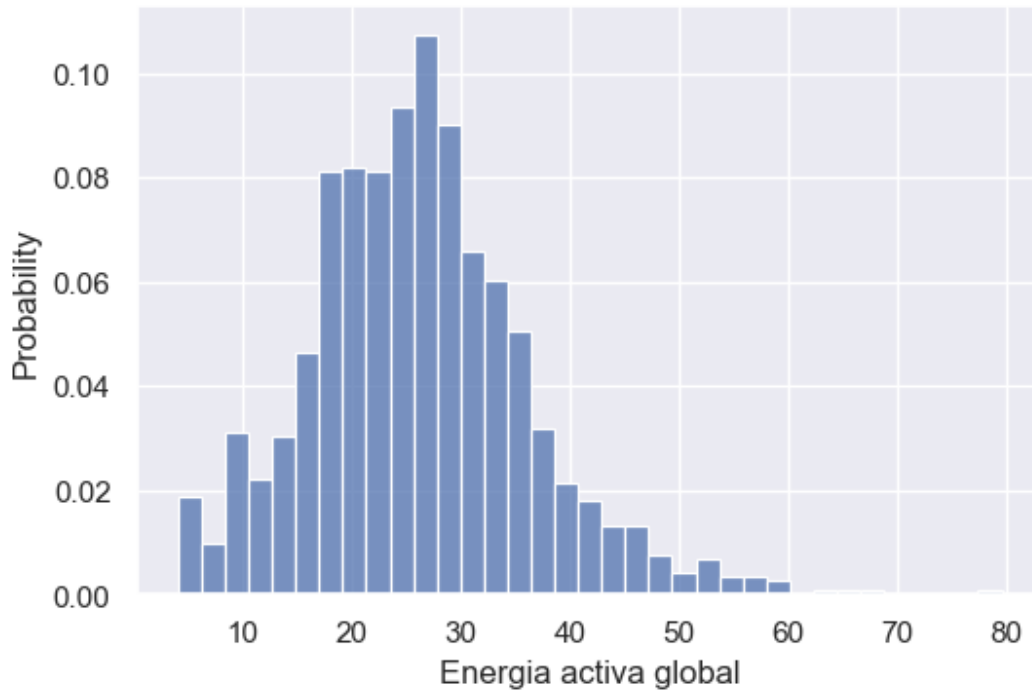


Figura 25. Histograma del consumo diario

Se va a trabajar con datos horarios, por lo que hay que adaptar las variables existentes. Se toman los siguientes datos estadísticos correspondientes a un día:

- Potencia activa global: Se transforma a energía ya que es de más aplicación en este problema. Se obtiene como la suma de la energía consumida en una hora.
- Potencia reactiva global: Se transforma a energía de la misma manera que la potencia activa.
- Voltaje: Sin cambios. Si antes era la media del voltaje en un minuto, ahora se calcula con la media de una hora.
- Intensidad global: Mismo tratamiento que la tensión
- Subconsumos: Se ha transformado la magnitud a kWh. Suma de la energía consumida en una hora.

El siguiente código aplica los cambios y ofrece información sobre el nuevo *dataframe*. No se han considerado los cambios anteriores de unidades para ofrecer mayor claridad en el código.

```

#Cambio de unidades a kWh
df['Sub_metering_1'] = df['Sub_metering_1']/1000
df['Sub_metering_2'] = df['Sub_metering_2']/1000
df['Sub_metering_3'] = df['Sub_metering_3']/1000

#Cambio de potencia a energia (kWh)
df['Energia activa global'] = df['Global_active_power']/60
df['Energia reactiva global'] = df['Global_reactive_power']/60

#Medidor 4
df['Sub_metering_4'] = df['Energia activa global']-(df['Sub_metering_1']+df['Sub_metering_2']+df['Sub_metering_3'])

#Creacion del nuevo dataframe
df_mean = df[['Global_intensity', 'Voltage']].copy()
df_sum = df.drop(['Global_intensity', 'Global_active_power', 'Voltage', 'Global_reactive_power'], axis=1)

#Resampling del nuevo dataframe
df_mean = df_mean.resample('h').mean()
df_sum = df_sum.resample('h').sum()

df1 = df_mean.merge(df_sum, left_index=True, right_index=True)

#Mover energia activa a la ultima posicion
brb = df1.pop('Energia activa global') # remove column b and store it in brb
df1.insert(0, 'Energia activa global', brb) # en primera posicion
    
```

Fragmento de código 9. Adaptación de los datos. Resample en horas

Si se muestran las 5 primeras filas se comprueba que la transformación es correcta

```
df1.head()
```

Fecha	Energia activa global	Global_intensity	Voltage	Sub_metering_1	Sub_metering_2	Sub_metering_3	Energia reactiva global	Sub_metering_4
2006-12-16 17:00:00	2.533733	18.100000	234.643889	0.0	0.019	0.607	0.137400	1.907733
2006-12-16 18:00:00	3.632200	15.600000	234.580167	0.0	0.403	1.012	0.080033	2.217200
2006-12-16 19:00:00	3.400233	14.503333	233.232500	0.0	0.086	1.001	0.085233	2.313233
2006-12-16 20:00:00	3.268567	13.916667	234.071500	0.0	0.000	1.007	0.075100	2.261567
2006-12-16 21:00:00	3.056467	13.046667	237.158667	0.0	0.025	1.033	0.076667	1.998467

Fragmento de código 10. Primeras 5 entradas

Se ha reducido el número de entradas de 2075259 a solo 34589, lo que supone una reducción de un 98,33%. Si bien se pierde bastante información, la nueva es de mejor calidad y mejora el coste computacional.

Ahora que los datos se encuentran en el formato deseado es momento de comprobar las correlaciones entre las variables. Debe generarse la matriz de correlación que muestre las variables relevantes para predecir la energía consumida. La matriz de correlación muestra los coeficientes de correlación entre cada una de las variables, con valores comprendidos entre -1 y 1. Un 1 representa una perfecta correlación lineal entre las variables mientras que un -1 representa una correlación negativa perfecta. Idealmente, se desea que las variables que participen en el entrenamiento del modelo tengan correlación con la variable objetivo.

Haciendo uso de la función integrada de la librería *pandas* `pandas.corr()` y la librería de gráficos *Seaborn* puede obtenerse una visualización sencilla de la matriz de correlaciones. El resultado aparece en la Figura 26.

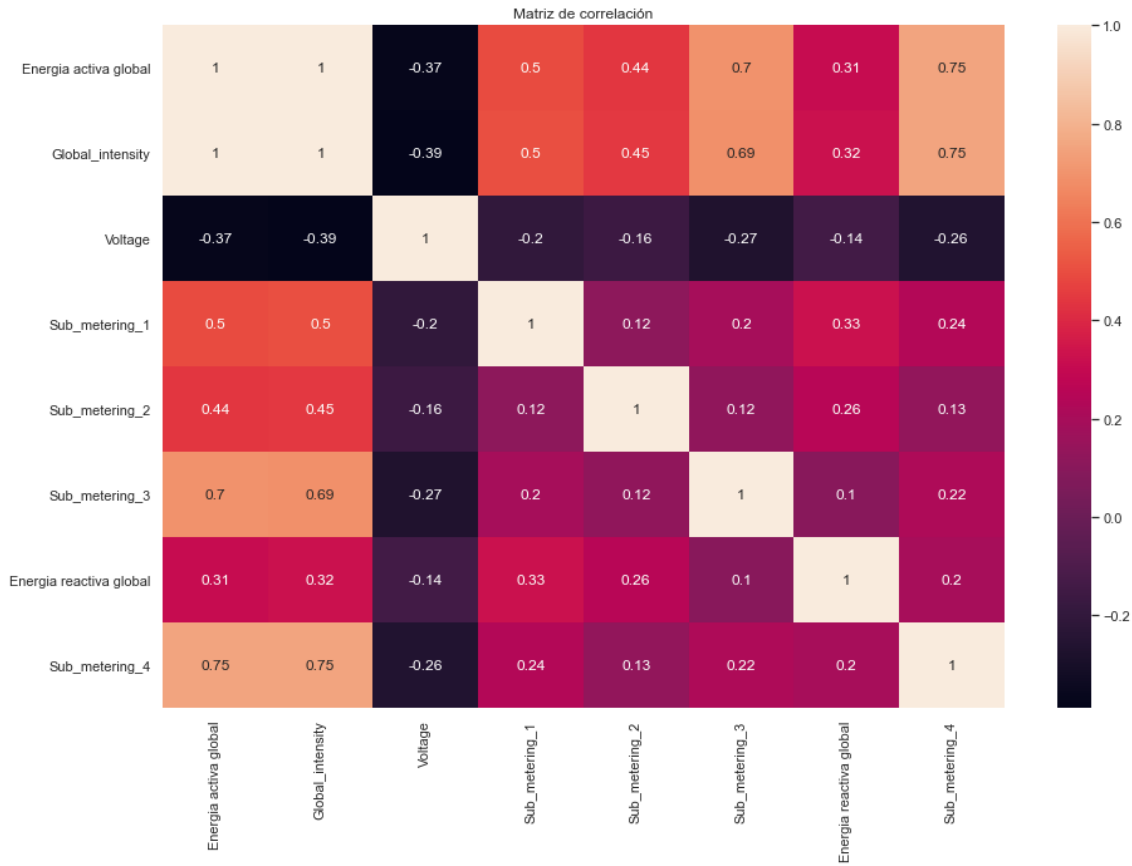


Figura 26. Matriz de correlación

De la matriz de correlaciones se pueden extraer las siguientes conclusiones:

- La energía reactiva global y la tensión apenas están relacionadas con la variable objetivo. Pueden descartarse del análisis.
- La intensidad presenta prácticamente una relación lineal con la variable objetivo
- Los subconsumos 3 y 4 están fuertemente relacionados con el consumo total.

Pueden comprobarse las hipótesis representando en un mismo gráfico cada variable con la variable objetivo. Haciendo uso de la librería *Sweetviz* pueden generarse rápidamente los gráficos necesarios. Las siguientes figuras ilustran la correlación entre la variable objetivo (energía activa global), representada con una línea, frente al histograma de valores de la variable con la que se compara, que aparece en la parte superior del gráfico.

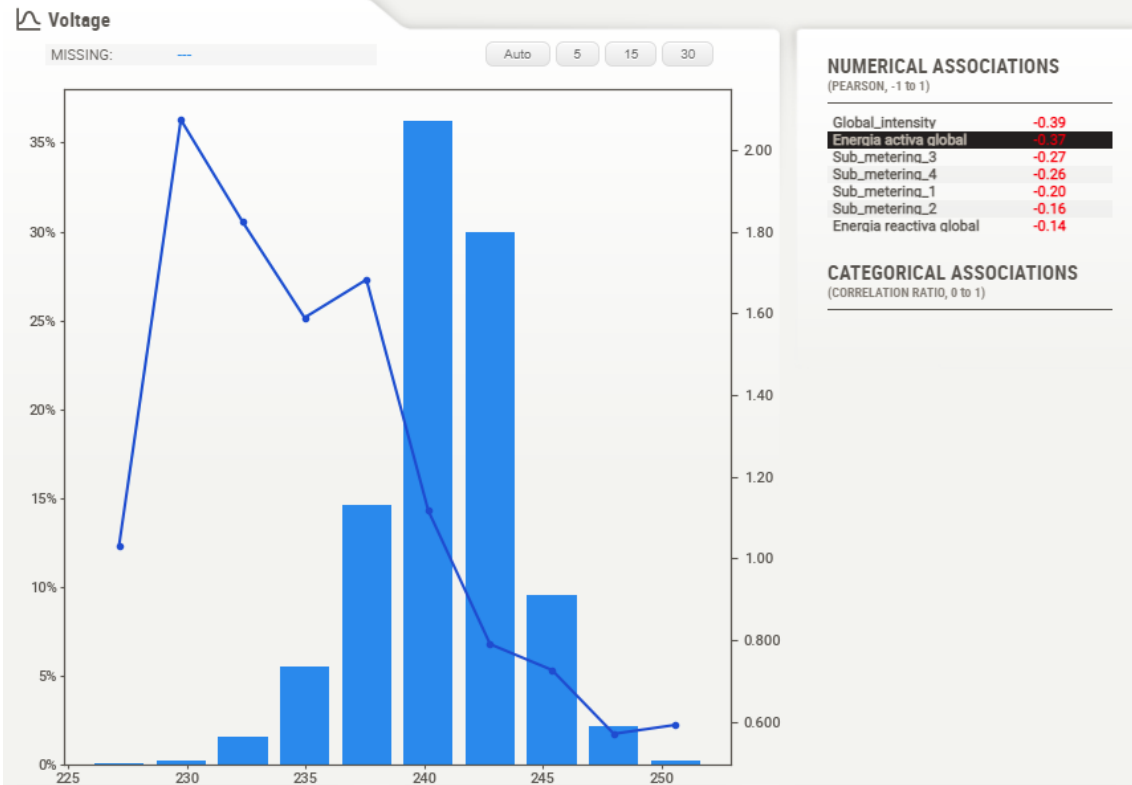


Figura 27. Energía activa vs voltaje

De la Figura 27 se comprueba que ambas variables no están correlacionadas. Esto era de esperar, ya que el voltaje debe mantenerse más o menos constante independientemente del consumo.

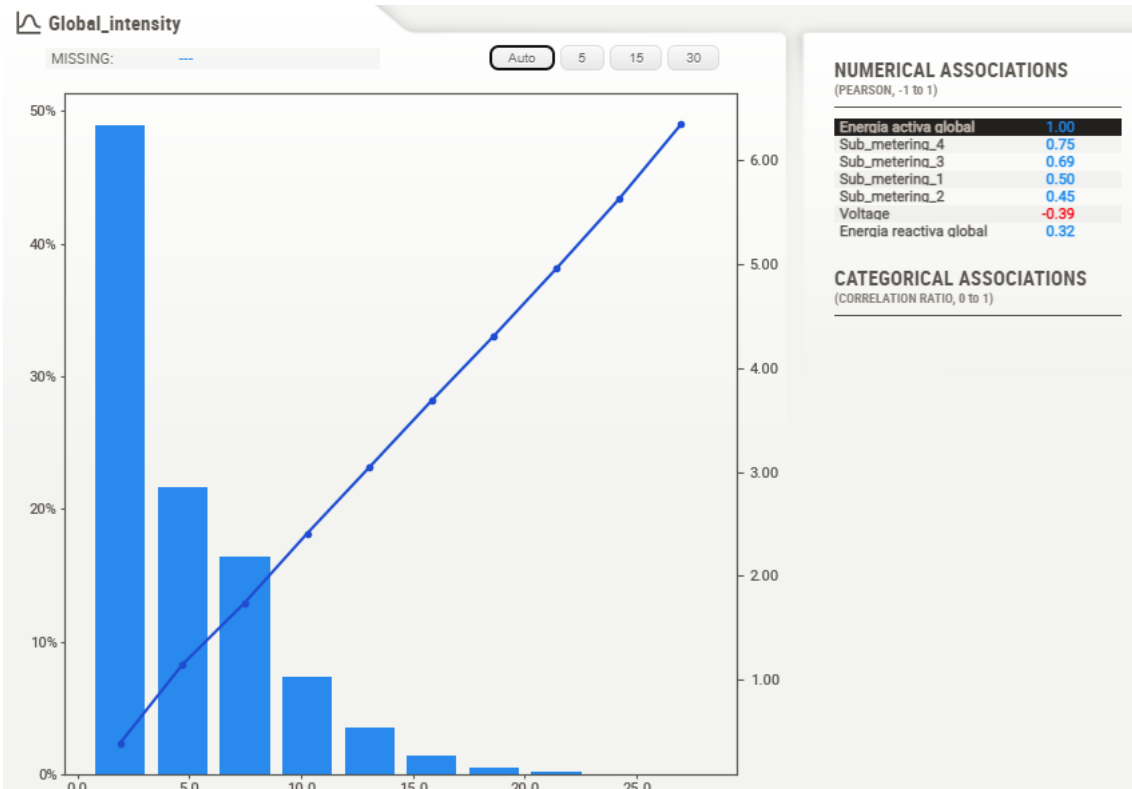


Figura 28. Energía activa vs intensidad

De la Figura 28 se puede confirmar la relación que se veía en la matriz de correlaciones:

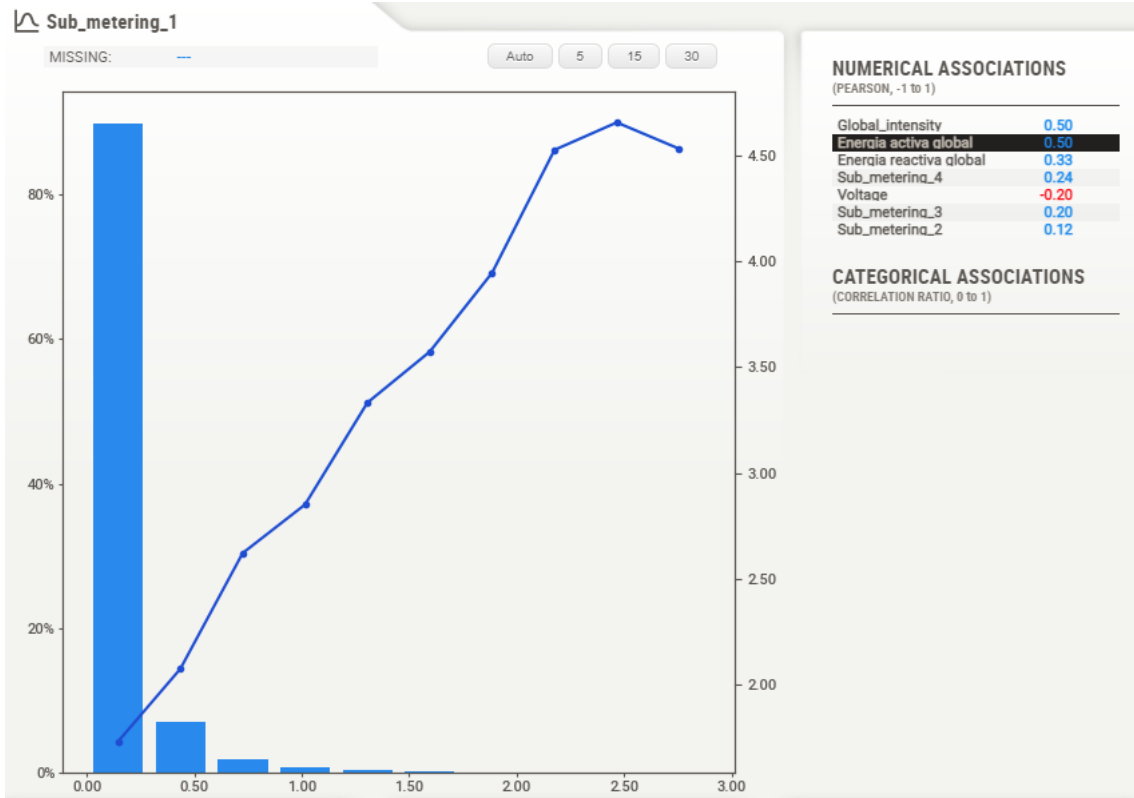


Figura 29. Subconsumo 1 vs energía activa

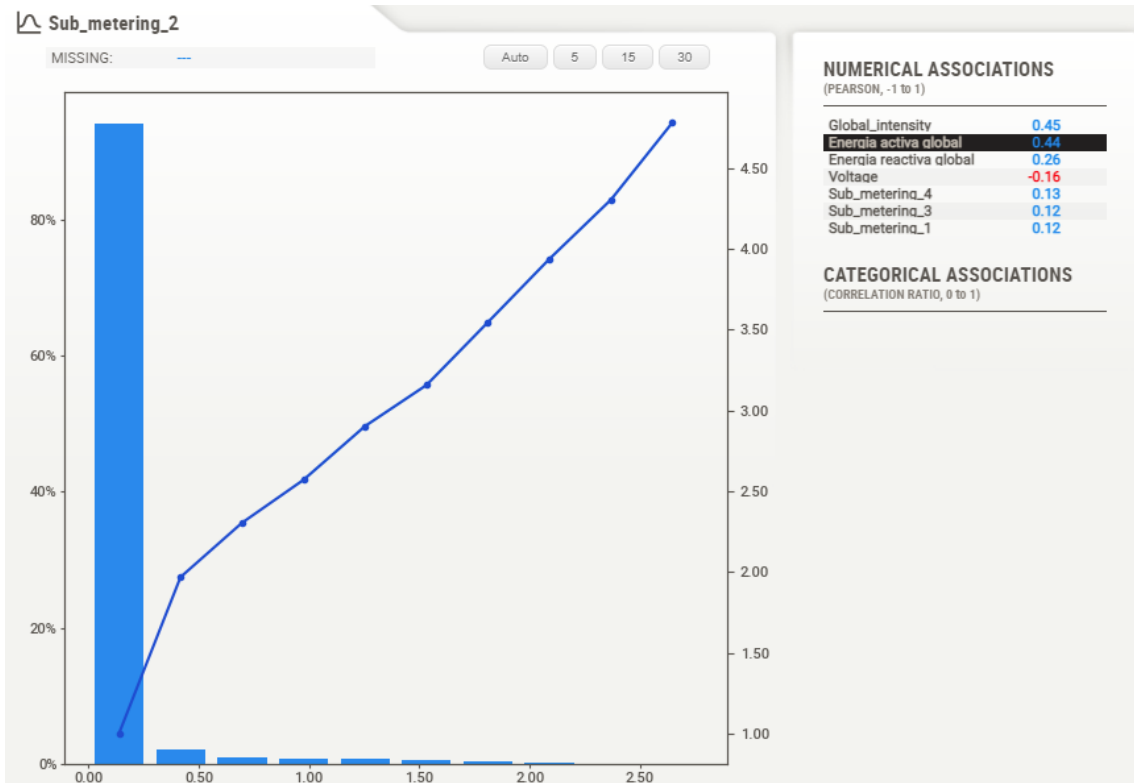


Figura 30. Subconsumo 2 vs energía activa

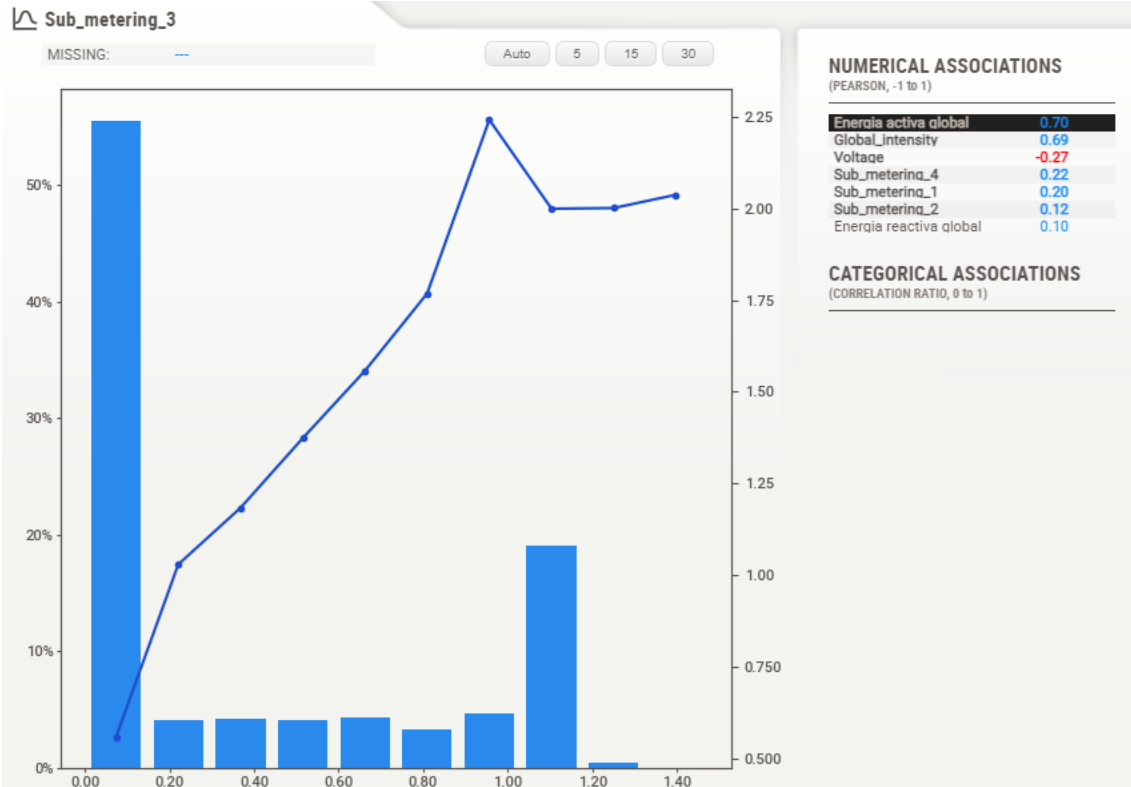


Figura 31. Subconsumo 3 vs energía activa

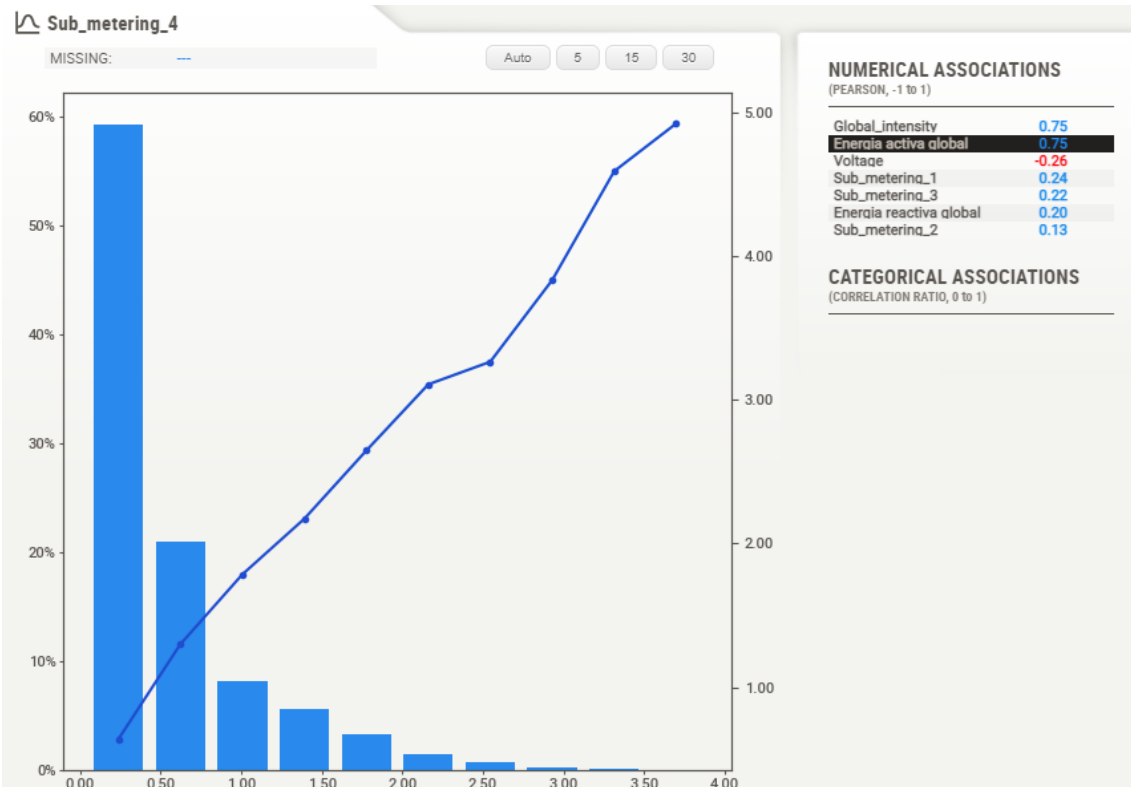


Figura 32. Subconsumo 4 vs energía activa

Las imágenes anteriores ofrecen información sobre la correlación del consumo global con los consumos en cada nodo. Existe una relación que era de esperar, ya que a mayor consumo en los nodos, mayor consumo global.

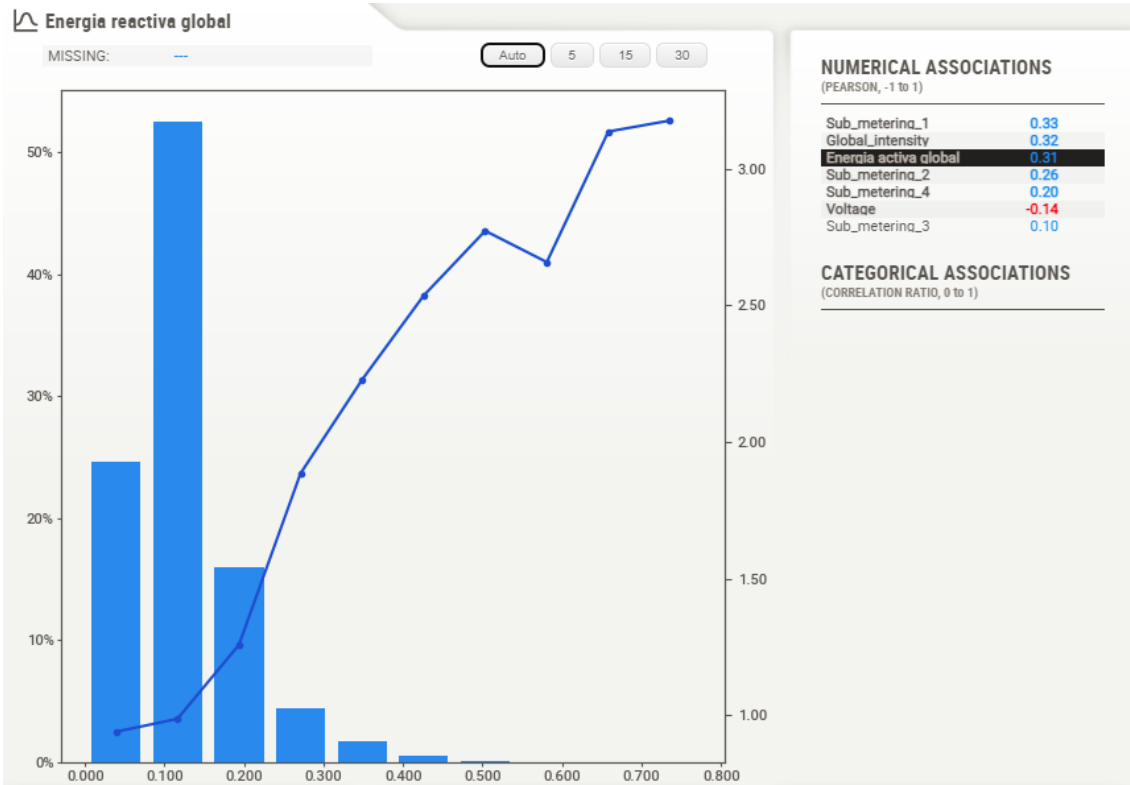


Figura 33. Energía reactiva vs activa

Finalmente, en la Figura 33, parece que se puede apreciar cierta correlación entre ambas variables, pero el coeficiente de Pearson es tan reducido que decide descartarse.

Eliminar variables no implica una mejor precisión en el modelo, pero sí mejora sustancialmente el coste computacional sin sacrificar apenas calidad en las predicciones. Podría haberse omitido este paso sin que impacte demasiado en el resultado final.

#### 4.4 Construcción del modelo

Como ya se ha justificado anteriormente en el apartado Redes Long Short-Term Memory (LSTM), se va a utilizar este tipo de modelo por ser el más adecuado para este tipo de series temporales.

##### 4.4.1 Convertir la serie temporal en un problema de aprendizaje supervisado

Los métodos de *Machine Learning* y *Deep Learning* que trabajan con series temporales necesitan reformular el problema para convertirlo en un problema de aprendizaje supervisado. Los problemas de aprendizaje supervisado necesitan de datos de entrenamiento de tal manera que se tengan pares de objetos en los que uno de los elementos son los datos de entrada y el otro la variable que se quiere predecir. En la Figura 10 se ha visto anteriormente la estructura requerida para este tipo de problemas.

Antes de empezar, se va a explicar la diferencia entre la serie temporal y el problema de aprendizaje supervisado. La metodología seguida puede consultarse en [11].

Una serie temporal es una secuencia de números ordenadas por un índice temporal. Por ejemplo, en este problema se tienen los datos de todas las variables correspondientes a cada instante temporal.

Un problema de aprendizaje supervisado consiste en una serie de entradas y salidas asociadas, de forma que el algoritmo pueda aprender cómo predecir los patrones de salida a partir de la entrada.

La problemática surge al analizar profundamente *qué* es lo que se quiere predecir de la variable objetivo. El enfoque tradicional que se usa generalmente en problemas de *Machine Learning* es predecir el valor de una variable en función de otras de entrada, normalmente llamadas *features*. Al final el problema puede resumirse en obtener:

$$Y = f(X)$$

Siendo  $X$  el conjunto de *features*.

Para este problema quiere predecirse el valor del consumo eléctrico de una vivienda. Analizando el conjunto de variables disponibles:

Fecha	Energía activa global	Global_intensity	Sub1	Sub2	Sub3	Sub4
2006-12-16 17:00:00	2,53	18,10	0,00	0,02	0,61	1,91
2006-12-16 18:00:00	3,63	15,60	0,00	0,40	1,01	2,22
2006-12-16 19:00:00	3,40	14,50	0,00	0,09	1,00	2,31
2006-12-16 20:00:00	3,27	13,92	0,00	0,00	1,01	2,26
2006-12-16 21:00:00	3,06	13,05	0,00	0,03	1,03	2,00

Tabla 2. Variables del set de entrenamiento

Se tienen todas las variables necesarias. El enfoque tradicional trataría de predecir el valor de la energía activa global en función del resto de variables. Sin embargo, esto no tienen ningún tipo de aplicación en este problema ya se va a conocer el consumo en todo momento. La predicción que interesa es el consumo *en el siguiente periodo*. ¿Cómo se puede reformular el problema para que la variable objetivo sea el siguiente periodo? Se ha reformulado el problema de manera que el objetivo ahora es:

$$Y_{t+1} = f(X, Y)$$

Para facilitar la comprensión del problema, la variable objetivo  $Y$  se incorpora al conjunto de *features*,  $X$ . La nueva variable objetivo es  $Y' = Y_{t+1}$ .



Ahora hay que transformar los datos para generar la nueva columna correspondiente a la predicción.

### La función *Pandas shift()*

La función clave para convertir la serie temporal en un problema de aprendizaje supervisado es la función *shift()* integrada en la librería *Pandas*.

Dado un *dataframe*, dicha función es capaz de desplazar los valores de una columna tantas filas como se desee y crear una columna con esa copia. Este es el proceso que se quiere para crear los valores retardados que se necesitan. El siguiente ejemplo ilustra el comportamiento de la función. Para el siguiente *dataframe* de ejemplo:

```
ejemplo = pd.DataFrame()
ejemplo['t'] = [x for x in range(10)]
ejemplo
```

	t
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9

Fragmento de código 11. Ejemplo de serie temporal

Se pueden desplazar todas las entradas un paso temporal, creando una nueva fila en la primera posición. Como la nueva fila no tiene datos, se usa el formato *NaN* para representarla.

```

ejemplo = pd.DataFrame()
ejemplo['t'] = [x for x in range(10)]
ejemplo['t-1'] = ejemplo['t'].shift(1)
ejemplo

```

	t	t-1
0	0	NaN
1	1	0.0
2	2	1.0
3	3	2.0
4	4	3.0
5	5	4.0
6	6	5.0
7	7	6.0
8	8	7.0
9	9	8.0

*Fragmento de código 12. Uso de pandas.shift*

Desplazando la serie temporal genera un problema de aprendizaje supervisado, aunque sin la nomenclatura correcta. La primera fila debe ser descartada por contener el valor *NaN*. Si se considera  $t_{-1}$  la entrada, es decir,  $X$ , y  $t$  la salida, se ha reformulado el problema tal y como se deseaba. A partir de unos valores temporales,  $X$ , se tienen los valores correspondientes al instante temporal sucesivo que se necesita para entrenar al modelo.

El uso de la función no está limitado a una única ventana temporal. Modificando el argumento de la función, se puede repetir el proceso para crear largas secuencias de datos de entrada ( $X$ ) que pueden ser utilizadas para predecir la variable objetivo. Es decir, se puede escoger el tamaño de la ventana temporal que el modelo va a utilizar como variable de entrada.

Reformulado, el tamaño de la ventana temporal determinará el número de días anteriores que el algoritmo considerará para entrenar al modelo. Una ventana temporal más grande por lo general resultará en mejores predicciones, a costa de aumentar considerablemente el tiempo de computación necesario para entrenar al modelo. Aumentar el tamaño de la ventana sin límite tampoco es una opción, ya que a partir de cierto tamaño la mejora en la precisión suele ser marginal.

### *La función `series_to_supervised()`*

Se ha hecho uso del código que aparece en [11] para adaptar la serie temporal al formato necesario. La función toma cuatro parámetros:

- **data:** La secuencia de datos como una lista o una matriz 2D NumPy.
- **n\_in:** Número de secuencias anteriores que se van a tomar como entrada ( $X$ ). Puede tomar valores entre 1 y el tamaño del vector *data*. Opcional. Por defecto es igual a 1
- **n\_out:** Número de observaciones como salida ( $y$ ). Toma valores entre 0 y el tamaño del vector *data* -1. Opcional. Por defecto es igual a 1.
- **dropnan:** Booleano que indica si eliminar o no los valores *NaN*. Por defecto es True

La función devuelve un único valor:

- **return:** *Dataframe* de la serie adaptado para aprendizaje supervisado.

El código de la función es el siguiente:

```

1. def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
2.     """
3.     Frame a time series as a supervised learning dataset.
4.     Arguments:
5.         data: Sequence of observations as a list or NumPy array.
6.         n_in: Number of lag observations as input (X).
7.         n_out: Number of observations as output (y).
8.         dropnan: Boolean whether or not to drop rows with NaN values.
9.     Returns:
10.        Pandas DataFrame of series framed for supervised learning.
11.    """
12.    n_vars = 1 if type(data) is list else data.shape[1]
13.    df = pd.DataFrame(data)
14.    cols, names = list(), list()
15.    # input sequence (t-n, ... t-1)
16.    for i in range(n_in, 0, -1):
17.        cols.append(df.shift(i))
18.        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
19.    # forecast sequence (t, t+1, ... t+n)
20.    for i in range(0, n_out):
21.        cols.append(df.shift(-i))
22.        if i == 0:
23.            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
24.        else:
25.            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
26.    # put it all together
27.    agg = pd.concat(cols, axis=1)
28.    agg.columns = names
29.    # drop rows with NaN values
30.    if dropnan:
31.        agg.dropna(inplace=True)
32.    return agg
    
```

Ejecutando un ejemplo se muestra mejor su uso, con el vector de datos que aparece a continuación:

```

values = [x for x in range(10)]
values

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    
```

Ejecutando la función para una ventana temporal de los 3 periodos anteriores se obtiene el siguiente *dataframe*:

```
data = series_to_supervised(values, 3)
data
```

	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
3	0.0	1.0	2.0	3
4	1.0	2.0	3.0	4
5	2.0	3.0	4.0	5
6	3.0	4.0	5.0	6
7	4.0	5.0	6.0	7
8	5.0	6.0	7.0	8
9	6.0	7.0	8.0	9

Fragmento de código 13. Ejemplo de dataframe tras adaptar la serie temporal

El resultado obtenido en el *dataframe* es el equivalente a la operación que aparece en la Figura 34. De esta manera, para cada fila se tienen tantos inputs como tamaño de ventana se haya seleccionado y una variable objetivo que se corresponde con el valor de la variable en el siguiente periodo.

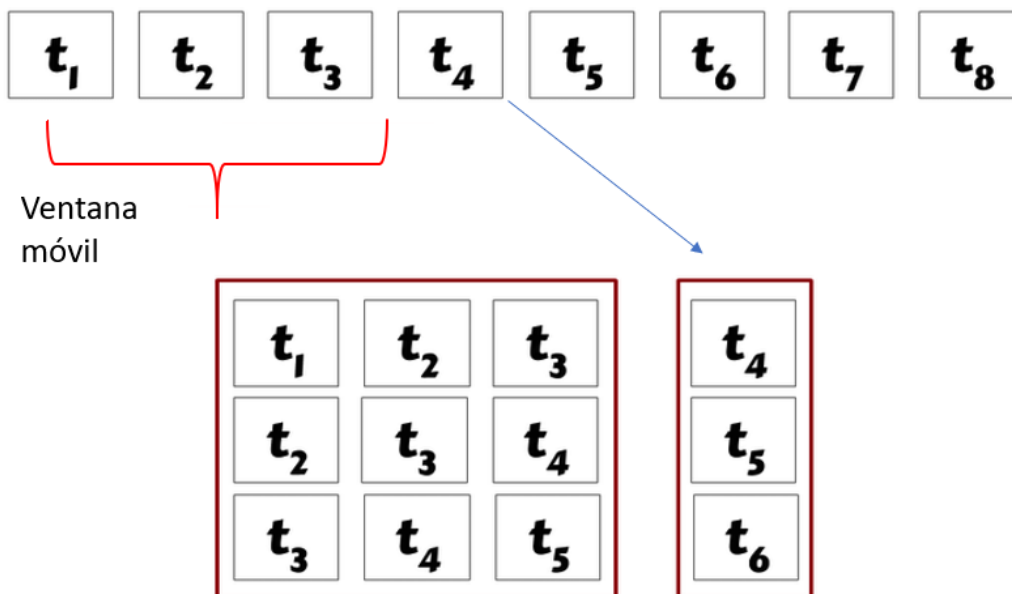


Figura 34. Reformulación de la serie temporal a problema de aprendizaje supervisado

Este ejemplo es para el caso en el que únicamente existe una variable de entrada, pero para este problema se tiene más de una. La función también está adaptada para estos casos. Por ejemplo, para el siguiente conjunto de datos:

```
raw = pd.DataFrame()
raw['ob1'] = [x for x in range(10)]
raw['ob2'] = [x for x in range(50, 60)]
values = raw.values
data = series_to_supervised(values)
data
```

	var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	0.0	50.0	1	51
2	1.0	51.0	2	52
3	2.0	52.0	3	53
4	3.0	53.0	4	54
5	4.0	54.0	5	55
6	5.0	55.0	6	56
7	6.0	56.0	7	57
8	7.0	57.0	8	58
9	8.0	58.0	9	59

Fragmento de código 14. Ejemplo 2, serie temporal en problema supervisado

Se crean dos nuevas columnas, una para cada variable, con las observaciones pasadas para un periodo. La Figura 34 ofrece algo más de información del proceso a seguir para.

#### 4.4.2 Preparación de los datos

Antes de introducir los datos al modelo, se necesitan una serie de pasos previos

##### 4.4.2.1 Normalización

Normalizar los datos es una buena práctica antes trabajar con redes neuronales. De esta manera, se reescala el vector de forma que todos los valores están comprendidos entre 0 y 1. Esta operación se puede realizar de forma rápida mediante la función integrada en la librería *sklearn* *MinMaxScaler()*. La transformación es dada por:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

El motivo por el que debe normalizarse antes de entrenar al modelo es que las variables que se miden con diferentes escalas no contribuyen de la misma manera en los pesos del modelo, lo que puede generar inestabilidades. Normalizar las variables para que todas tomen valores entre 0 y 1 mejora el proceso de entrenamiento.

El uso de la función es muy sencillo. Tomando los datos de entrenamiento:

```

values = df1.values
scaler = MinMaxScaler()
scaled = scaler.fit_transform(values)

pd.DataFrame(scaled, columns=df1.columns)

```

	Energia activa global	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4
0	0.381805	0.631157	0.0	0.006820	0.412513	0.484194
1	0.550442	0.541487	0.0	0.144652	0.687748	0.564711
2	0.514830	0.502152	0.0	0.030869	0.680272	0.589697
3	0.494617	0.481110	0.0	0.000000	0.684350	0.576254
4	0.462055	0.449904	0.0	0.008973	0.702019	0.507801
...	...	...	...	...	...	...
34584	0.257786	0.240794	0.0	0.000000	0.524645	0.236026
34585	0.234384	0.219393	0.0	0.000000	0.000000	0.397225
34586	0.247567	0.235055	0.0	0.001436	0.000000	0.418525
34587	0.171477	0.158178	0.0	0.022972	0.000000	0.273960
34588	0.000000	0.118245	0.0	0.000000	0.000000	0.000000

34589 rows × 6 columns

Fragmento de código 15. Dataframe escalado entre 0 y 1

Se observa que ninguna de las columnas contiene valores superiores a 1 ni inferiores a 0.

#### 4.4.2.2 *Uso de series\_to\_supervised()*

Una de las decisiones importantes es decidir el tamaño de la ventana temporal que se va emplear en la predicción. Este parámetro suele decidirse de forma empírica, atendiendo al tipo de problema. Como se está trabajando con consumos eléctricos, que pueden variar fuertemente a lo largo del día, se va a utilizar una ventana temporal de 24 horas como primera aproximación.

En función de la evaluación del modelo este parámetro podría aumentarse o reducirse. Cuando se tiene que decidir el valor de parámetros de forma empírica, es mejor empezar desde abajo, ya que resulta en un menor tiempo de computación. Probablemente el entrenamiento deberá realizarse repetidas veces hasta dar con un resultado satisfactorio, por lo que es contraproducente iniciar el proceso con tiempos de entrenamiento elevados.

Haciendo uso de la función:

```
supervised = series_to_supervised(scaled, n_hours, 1)
```

```
supervised.head()
```

	var1(t-24)	var2(t-24)	var3(t-24)	var4(t-24)	var5(t-24)	var6(t-24)	var1(t-23)	var2(t-23)	var3(t-23)	var4(t-23)	...
24	0.381805	0.631157	0.0	0.006820	0.412513	0.484194	0.550442	0.541487	0.0	0.144652	...
25	0.550442	0.541487	0.0	0.144652	0.687748	0.564711	0.514830	0.502152	0.0	0.030869	...
26	0.514830	0.502152	0.0	0.030869	0.680272	0.589697	0.494617	0.481110	0.0	0.000000	...
27	0.494617	0.481110	0.0	0.000000	0.684350	0.576254	0.462055	0.449904	0.0	0.008973	...
28	0.462055	0.449904	0.0	0.008973	0.702019	0.507801	0.330590	0.323529	0.0	0.002872	...

5 rows × 150 columns

*Fragmento de código 16. Series to supervised, resultado*

Se ha transformado el número de inputs de 6 columnas a  $6 \times 24 + 6 = 144$ . Esto se debe a que, a las 6 variables originales, se le añaden los 24 periodos previos para cada una, es decir, 25 columnas por cada variable.

Del *dataframe* resultante sobra información. La variable objetivo es el consumo eléctrico en el periodo siguiente, así que debería ser la única que disponga información en el instante t, que es el que se utilizará como salida. Se deben eliminar el resto de las variables en el instante t. Como la variable objetivo es "var1" y se sabe la posición que ocupa en el *dataframe*, se seleccionan las últimas columnas para posteriormente eliminarlas.

```
#Eliminamos variables que no vayan a predecirse
variables_eliminadas = supervised.columns[-(n_features-1):]
supervised.drop(variables_eliminadas, axis=1, inplace=True)
```

*Fragmento de código 17. Eliminar variables sin uso*

Se han eliminado las variables necesarias.

#### 4.4.2.3 Dividir entre set de entrenamiento y validación

El modelo necesita datos para su entrenamiento, pero también necesita otros datos con los que contrastar el modelo y verificar su funcionamiento. Para solucionar este problema, normalmente los datos de entrenamiento se fraccionan en dos grupos: datos de entrenamiento y datos de validación. La Figura 35 muestra esta división:

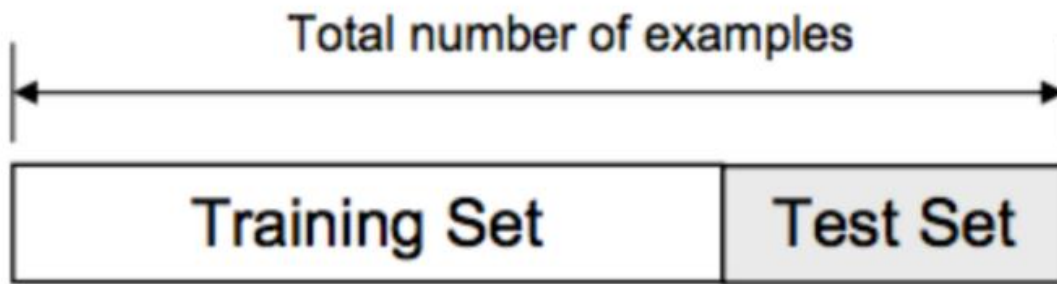


Figura 35. División entre datos de entrenamiento y validación [12]

Normalmente esta división puede ser tan sencilla como elegir, por ejemplo, el 75% de los datos para entrenamiento y el resto para test, en orden de aparición en el *dataframe*. Mediante el siguiente trozo de código se genera la división

```
# Dividir en entrenamiento y test
values = supervised.values

n_train_time = int(supervised.shape[0]*0.75)
train = values[:n_train_time, :]
test = values[n_train_time:, :]

# Dividir en entradas y salidas
X_train, y_train = train[:, :-1], train[:, -1]
X_test, y_test = test[:, :-1], test[:, -1]
```

Fragmento de código 18. División entre entrenamiento y test

Hay técnicas más elaboradas para conseguir esta división, como por ejemplo, tomar datos no consecutivos de forma aleatoria para que los datos de entrenamiento sean más homogéneos. No obstante, al trabajar con series temporales no se desea perder el orden de la secuencia de los datos ya que es relevante para el análisis.

#### 4.4.2.4 Redimensionar la entrada de la red LSTM

La red LSTM necesita de un *input* en tres dimensiones. Las tres dimensiones de este *input* son:

- Muestras: Número de secuencias temporales en el set de entrenamiento.
- Secuencias: Número de observaciones en una muestra. En este caso son 7
- Variables: Número de columnas



```
# Redimensionar el input a 3d [muestras, secuencias, variables]
X_train = X_train.reshape((X_train.shape[0], n_hours, n_features))
X_test = X_test.reshape((X_test.shape[0], n_hours, n_features))
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

(25923, 24, 6) (25923,) (8642, 24, 6) (8642,)
```

*Fragmento de código 19. Redimensionamiento del input a la red LSTM*

Se han redimensionado las matrices X para cumplir con los requisitos de entrada a la red LSTM.

### 4.4.3 Modelo LSTM

#### 4.4.3.1 Definición del modelo

Una red neuronal está formada por capas. La red más básica que existe está formada por una capa. En el caso de una red LSTM, mediante el uso de la librería *Keras*, la sintaxis para crear una red es la siguiente:

```
# definir modelo
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

*Fragmento de código 20. Definición de la red LSTM*

Esta red consiste de una única capa LSTM con 50 neuronas, más una capa de salida convencional “Dense” de una sola neurona. La capa de salida es necesaria para adaptar el output del modelo al formato deseado.

El primer paso es añadir la capa de la función de activación. Esta va integrada en la capa LSTM, en este caso se utiliza la función ReLU. No es objeto de este TFM explicar la herramienta matemática detrás de la red neuronal, pero a grandes rasgos, la función ReLU (Rectified Linear Unit) se encarga de asegurar que la salida de la capa tome únicamente valores positivos. Para más información sobre la función de activación puede consultarse [13]. Se emplea esta función por su buen funcionamiento en este tipo de problemas de regresión.

La función de pérdidas elegida es el error medio cuadrático (rmse). El motivo de esta decisión es que es deseable que las pérdidas estén en las mismas unidades que la variable objetivo. Para este fin pueden elegirse tanto el error cuadrático medio como el error absoluto medio como el error medio absoluto (mae). Se ha escogido el error medio cuadrático por ser más penalizante en las predicciones. Además, es ampliamente más usado.

Realmente, el modelo empleado no va a ser tan sencillo como el del ejemplo. Es interesante añadir más capas LSTM encadenadas a fin de que la red sea capaz de encontrar más relaciones entre las variables.

```
#Preparar la red LSTM
units = 50
#n_outputs = y_train.shape[1]

model = Sequential()
model.add(LSTM(units, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
```

Fragmento de código 21. Definición de la red LSTM, 2

Añadir más capas o neuronas a las capas supone un aumento de coste computacional y la posibilidad de que aparezca *overfitting*. El *overfitting* es un fenómeno que ocurre cuando el modelo estadístico se ajusta exactamente a los datos de entrenamiento. Cuando esto ocurre, el algoritmo no puede trabajar con datos que todavía no haya visto, perdiendo su funcionalidad [14].

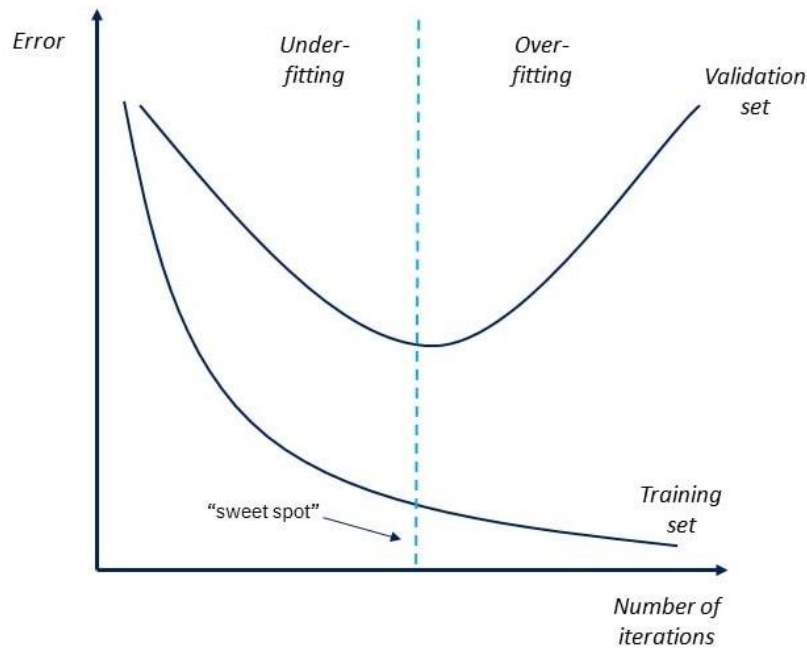


Figura 36. Overfitting & Underfitting. [14]

De la misma manera, se puede generar la situación contraria, *underfitting*, cuando la red todavía tiene potencial de mejora y no se ha entrenado lo suficiente al modelo. El objetivo es alcanzar el punto en el que el error de validación y el error de entrenamiento alcanzan el mínimo.

Para solucionar esto se implementa lo que se conoce como una función *callback*. Una función *callback* es una que se le pasa a otra función como argumento, y es ejecutada cuando se cumple determinada condición. En este caso, se impone como función *callback* la función integrada en la librería *Keras EarlyStopping*.

```
from keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    min_delta=0.0005, # minimum amount of change to count as an improvement
    patience=10, # how many epochs to wait before stopping
    restore_best_weights=True,
)
```

Fragmento de código 22. Early Stopping

Indicando al modelo que se desea implementar *EarlyStopping*, lo que se consigue es detener el proceso de iteración cuando el error de validación no mejora con las sucesivas iteraciones. Si esto sucede, lo que ocurre es que el modelo está ajustando los pesos al ruido presente en los datos de entrenamiento, y es necesario detener el proceso antes de que el modelo sea inservible.

La función detiene el entrenamiento cuando el error de validación no mejora por encima de un delta asignado, *min\_delta*, que se ha asignado a un valor de 0.0005. El número de epochs que se comprueban con esta condición se especifica en el parámetro *patience*, asignado en 10. Y por último, indicando la opción *restore\_best\_weights* se indica que se recuperen los pesos que presentan menor error. Estos valores son seleccionados de forma empírica, y se han ajustado a valores comunes que se suelen utilizar en este tipo de aplicaciones. Reducir el delta o aumentar el parámetro *patience* implica que el entrenamiento será más largo, pero pueden lograrse mejores resultados. Sin embargo, para esta aplicación, los efectos de usar parámetros más exigentes no han ofrecido una mejora significativa en los resultados.

Otra forma de evitar el *overfitting* es incorporar capas *dropout* entre las capas del modelo. Estas capas eliminan de forma aleatoria algunas de las entradas en las capas ocultas en cada iteración, provocando que sea mucho más difícil para la red encontrar patrones extraños en el set de entrenamiento. De esta manera, busca patrones generales, presentes en todos los datos, cuyos patrones de pesos tienden a ser más robustos. El parámetro especificado en la capa *dropout* es el porcentaje de entradas que van a eliminarse.

Solo resta por indicar el número de iteraciones (epoch) y el *batch size*, explicado en el apartado **¡Error! No se encuentra el origen de la referencia.** El número de epochs inicial puede ser elevado, ya que al disponer de *EarlyStopping* el proceso durará hasta que el error no mejore más, por lo que es conveniente entrenar el modelo hasta que esto suceda. El *batch size* afecta principalmente al tiempo de computación. Incrementar el *batch size* sin límite normalmente genera resultados peores. Por lo general, se recomienda empezar con 32, e ir experimentando con 64, 128 y 256. Estos parámetros se ajustan de forma iterativa hasta encontrar resultados satisfactorios.

#### 4.4.3.2 Entrenar la red

Finalmente llega el momento de entrenar la red. Mediante el método *model.fit()* se invoca al método que realiza esta función.

```
#Elegir hiperparametros
batch_size = 32
epochs = 50

# fit network
history = model.fit(X_train, y_train,
                    epochs=epochs, batch_size=batch_size,
                    validation_data=(X_test, y_test),
                    verbose=1, shuffle=False, callbacks=[early_stopping])
```

Fragmento de código 23. Entrenar la red

Tras un proceso iterativo que se muestra por pantalla el algoritmo llega a su fin y se genera el modelo. Puede representarse el proceso grafando los errores de entrenamiento y validación frente al número de iteraciones.

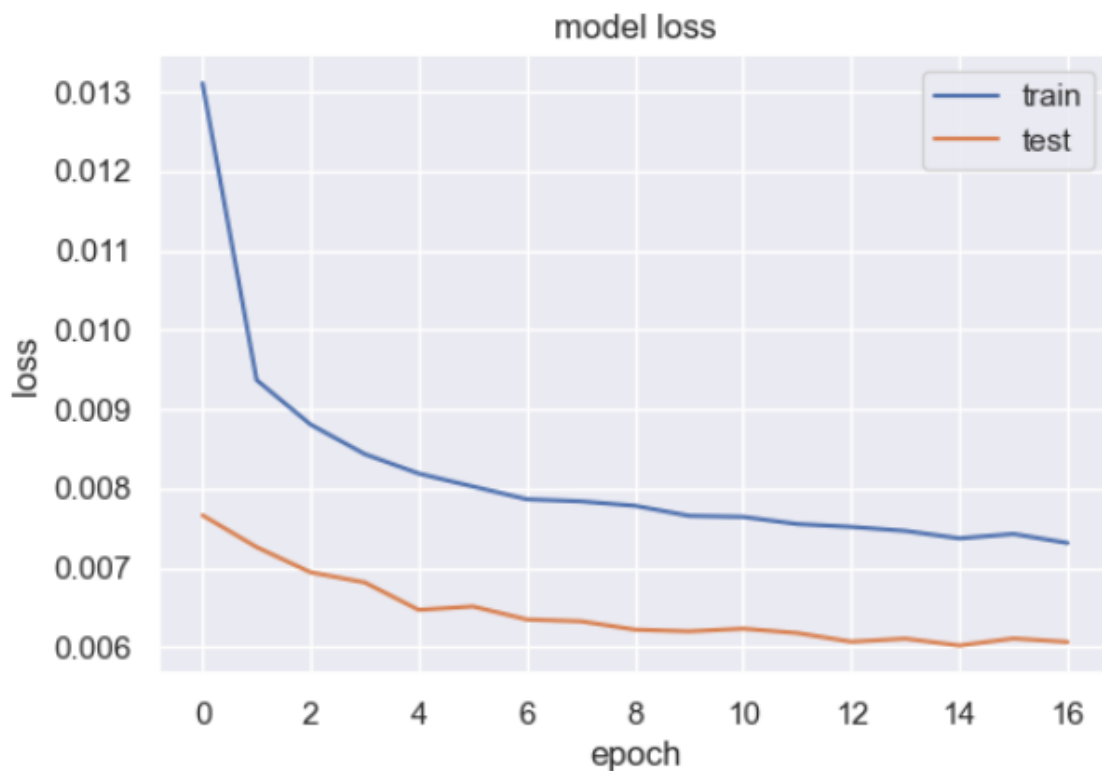


Figura 37. Pérdidas del modelo durante el proceso iterativo

Puede observarse que el error en el set de entrenamiento cae muy rápidamente al principio, pero no mejora al cabo de cierto número de iteraciones. Como el error de validación tampoco mejora, la función *EarlyStopping* detiene el proceso al cabo de 17 epochs, a pesar de que se había especificado que el número de epochs a entrenar era de 50.

#### 4.5 Verificación de resultados

Tras entrenar el modelo resta verificar la calidad de los resultados. Para ello, se comparan las predicciones que se generan al introducir el set de datos de validación y se comparan frente a frente con los valores reales.

Si bien obtener la predicción es extremadamente sencillo, es necesario adaptar el formato de salida, ya que anteriormente se habían escalado los datos entre 0 y 1 para facilitar el trabajo a la red neuronal. Ahora hay que deshacer el cambio, se puede realizar fácilmente invocando el método `inverse_transform()` adaptado a la transformación realizada previamente. Para ello hay que reconstruir la matriz original, redimensionando `X_test` y concatenándola a la matriz de predicciones. Esta operación se realiza tanto para la predicción generada por el modelo como para el valor real de la variable de salida.

```
# Hacer una predicción
yhat = model.predict(X_test)
X_test2 = X_test.reshape((X_test.shape[0], n_hours*n_features))
# Invertir el escalado de la predicción
inv_yhat = np.concatenate((yhat, X_test2[:, -(n_features-1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# Invertir el escalado del real
test_y = y_test.reshape((len(y_test), 1))
inv_y = np.concatenate((test_y, X_test2[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calcular RMSE
rmse = np.sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)

Test RMSE: 0.534
```

Fragmento de código 24. Resultados del modelo

Una vez deshecha la transformación, pueden compararse ambas variables para obtener el error de la predicción, que es el indicador que marca la calidad del modelo. Puede representarse la predicción frente al valor real de la variable objetivo, para los valores del set de validación. El resultado aparece en la Figura 38

```
## time steps, every step is one hour (you can easily convert the time step to the actual time index)
## for a demonstration purpose, I only compare the predictions in 200 hours.

hours = 200

plt.figure(figsize = (16,7))
plt.title('Predicción vs real')

aa=[x for x in range(hours)]
plt.plot(aa, inv_y[:hours], marker='.', label="actual")
plt.plot(aa, inv_yhat[:hours], 'r', label="prediction")
plt.ylabel('Energía activa global (kWh)', size=15)
plt.xlabel('Horas', size=15)
plt.legend(fontsize=15)
plt.show()
```

Fragmento de código 25. Visualización de la predicción

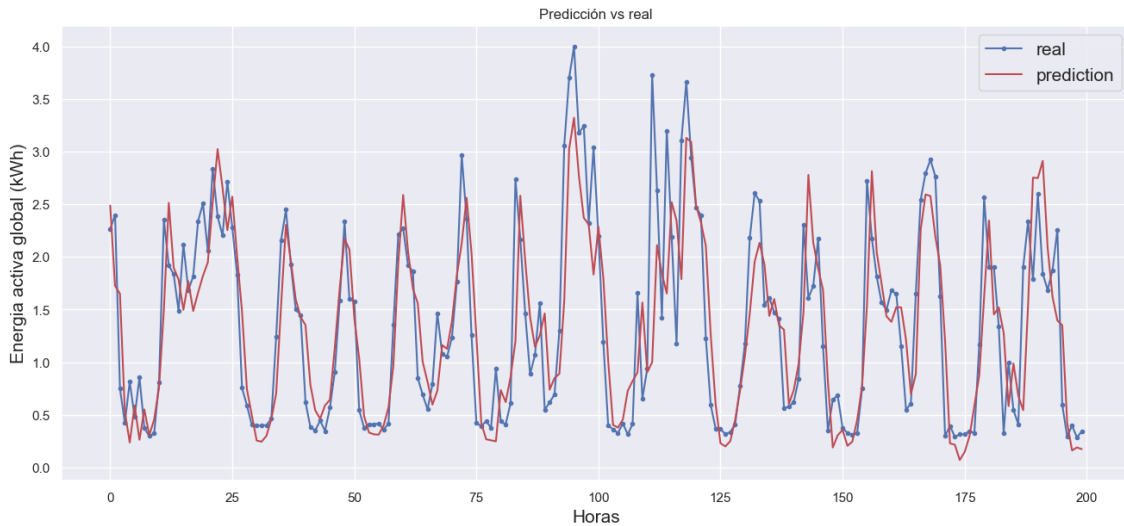


Figura 38. Predicción vs valor real

Puede verificarse que el resultado es bastante bueno. Es conveniente repetir el proceso con hiperparámetros de distinto valor. La naturaleza estocástica del algoritmo implica que los resultados varíen aunque se repita el entrenamiento con la misma configuración. Los parámetros que van a modificarse son los siguientes:

- Ventana temporal
- Tamaño del batch
- Número de neuronas en cada capa
- Número de capas

El objetivo es encontrar la configuración con menor error. Se ha preparado un bucle que entrena la red sucesivas veces cambiando los parámetros deseados y guarda el error para compararlo con el resto de los experimentos. Los resultados pueden consultarse en el documento Anexo. El resumen de los resultados aparece en las Tabla 3, Tabla 4 y Tabla 5 para ventanas temporales de 12, 24 y 48 horas:

12 HORAS	
iteración	12
units	150
batch_size	128
capas	1
error	0,532

Tabla 3. Mejor error para ventana de 12h

24 HORAS	
<b>iteración</b>	21
<b>units</b>	100
<b>batch_size</b>	64
<b>capas</b>	2
<b>error</b>	0,511

Tabla 4. Mejor error para ventana de 24h

48 HORAS	
<b>iteración</b>	9
<b>units</b>	100
<b>batch_size</b>	32
<b>capas</b>	2
<b>error</b>	0,507

Tabla 5. Mejor error para ventana de 48h

De los resultados obtenidos pueden obtenerse varias conclusiones. Aumentar el número de capas implica por lo general un mejor aprendizaje de la red, así como el número de neuronas en cada capa. Sin embargo, llega un punto que el error deja de mejorar. La otra gran observación es que para mayor tamaño de ventana temporal el error se reduce. No obstante, aumenta considerablemente el tiempo de entrenamiento requerido.

El experimento que ha obtenido menor error ha sucedido para una ventana de 48 horas. Atendiendo al error, hay que compararlo con la variable objetivo. Para los datos de entrenamiento se tiene:

Energia activa global	
count	34589.000000
mean	1.090237
std	0.894216
min	0.046733
25%	0.342867
50%	0.803216
75%	1.578700
max	6.560533

Tabla 6. Datos estadísticos sobre la variable objetivo

Puede compararse el error cuadrático medio con el rango de la variable objetivo ( $R$ ):

$$R = 6.56 - 0.04 = 6.52$$

El porcentaje del error sobre el rango es:

$$E(\%) = \frac{RMSE_{48h}}{R} = 7.77\%$$

Podría intentar justificarse que los valores mínimos y máximos se encuentran muy alejados de los percentiles 25 y 75% de los datos, con lo que no son una buena forma de evaluar el rango en el que se mueven los datos. No obstante, los patrones de consumo se caracterizan por oscilar entre picos elevados y periodos de baja actividad, por lo que tomar estas medidas extremas sí que es representativo. La Figura 24. Consumo por horas, ofrece más información al respecto.

El error es aceptable, pero comparando el resultado obtenido con el experimento para una ventana temporal de 24h es prácticamente el mismo. Reducir el modelo a una ventana temporal de la mitad supone una gran mejora en el tiempo de computación, ya que por cómo se adaptan los datos, al trabajar con una ventana de 24h se le introduce al modelo un *dataframe* de 24 columnas, mientras que el de 48h supone trabajar con 48. Si se trabaja con el modelo de 24h el error supone:

$$E(\%) = \frac{RMSE_{24h}}{R} = 7.84\%$$

Por lo que se decide trabajar con esta configuración. Si se compara el resultado obtenido con este modelo para el set de validación se obtiene la Figura 39. Además, puede decirse que gran parte del error proviene de las horas en las que el consumo se dispara y el modelo no es capaz de predecir con precisión los picos de consumo. Con todo, para el resto de los casos el modelo presenta buenos resultados.



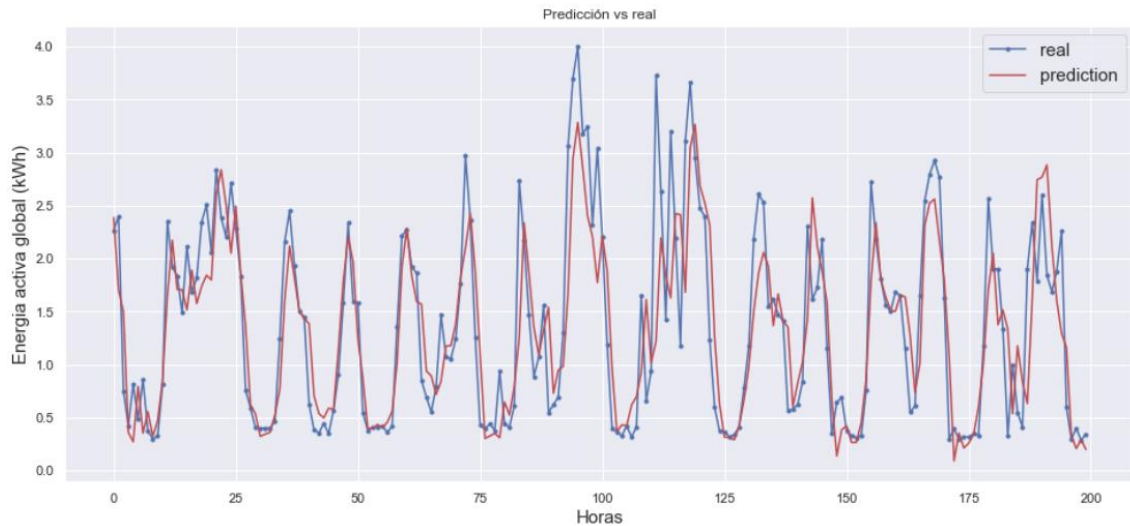


Figura 39. Predicción vs real para el modelo escogido, 200 primeras predicciones

## 4.6 Guardar modelo

Una vez seleccionado el modelo, es necesario guardarlo en disco para poder cargarlo posteriormente. Si no se guardase el modelo, sería necesario entrenar un modelo nuevo cada vez que se ejecute el script del algoritmo. Debido a la gran cantidad de datos presentes en el set de entrenamiento esto se considera inviable, si bien este planteamiento presenta una gran ventaja. Si el set de entrenamiento fuese mucho más reducido y la capacidad de computación no fuese un problema, podrían actualizarse los datos de entrenamiento con los nuevos datos recogidos durante el funcionamiento del dispositivo, actualizando los patrones de comportamiento y adaptándolos a la actualidad cada cierto tiempo. De todas maneras, se deja planteado como posible mejora para el futuro ya que complica la ejecución del algoritmo y requiere de un hardware más potente.

La librería *Keras* dispone de una API sencilla para guardar y restaurar modelos en ficheros. De la propia documentación de la librería:

```
# Guardar el Modelo
model.save('path_to_my_model.h5')

# Recrea exactamente el mismo modelo solo desde el archivo
new_model = keras.models.load_model('path_to_my_model.h5')
```

Fragmento de código 26. Guardar el modelo

Al guardar el modelo realmente se están serializando los pesos de la red neuronal en un fichero que puede restaurarlos más tarde. El script externo que haga uso del modelo deberá cargar el archivo antes de generar predicciones. En este caso, el modelo generado se ha guardado en el mismo directorio desde el que se ejecuta el script, es decir, en el dispositivo local.

## 5 PREDICCIÓN DE LA GENERACIÓN

En este capítulo se va a desarrollar el programa que permite obtener los datos necesarios para generar la predicción de consumo. Junto con la predicción de consumo, es la segunda parte vital para desarrollar el algoritmo que permite tomar la decisión sobre la carga de las baterías

### 5.1 Planteamiento

El objetivo es obtener el dato de irradiancia de la hora siguiente al momento de ejecución para poder comparar la energía generada con el consumo eléctrico. Al tratarse de datos meteorológicos, hay gran cantidad de herramientas online que permiten obtener las consultas necesarias.

Una de las herramientas más potentes a disposición de todo el mundo es la base de datos PVGIS. PVGIS es un sitio web de la Unión Europea que ofrece información sobre la radiación solar y la potencia fotovoltaica. Es de gran utilidad para el diseño de proyectos fotovoltaicos y permite obtener cálculos de forma sencilla. Si bien en un primer momento se planteó hacer uso de la API de PVGIS, surge un problema relacionado con la manera en la que se ofrecen los datos. Todos los datos que se pueden obtener a partir de PVGIS provienen de cálculos estadísticos en base a datos históricos. Es decir, puede consultarse el histórico de datos para una localización, o los valores más probables en base a situaciones del pasado.

Sin embargo, no es el enfoque que se necesita para este problema. Se necesita la predicción para la hora siguiente. Basarse en datos estadísticos puede estar bien para el diseño de una instalación, pero no son predicciones que tengan en cuenta la situación meteorológica actual.

La solución que puede cumplir con estos requisitos es el uso de una API de terceros. Una API REST es una interfaz de programación de aplicaciones (API) que permite la interacción con servicios que se ajustan a la arquitectura REST. REST comprende las interfaces entre sistemas que usen el protocolo HTTP para obtener datos u operar sobre los mismos en todos los formatos posibles.

En otras palabras, las API permiten interactuar con un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla. En este caso, se va a emplear para obtener el valor de la irradiancia en la ciudad de Valencia una hora después del momento de la solicitud.

### 5.2 Selección de la API

Existen multitud de servicios de terceros compatibles con esta aplicación. La gran limitación existe en el tipo de licencia disponible, que normalmente limita la cantidad de peticiones según el tipo de cuentas. Se ha realizado un estudio de alternativas de los servicios que ofrecen la predicción de irradiancia de forma directa

- [AccuWeather](#): Este popular servicio es ampliamente usado en aplicaciones meteorológicas por la cantidad de información que ofrece y su fácil uso.

- [Solcast](#): Potente API centrada en ofrecer datos relacionados con la producción de energía fotovoltaica.
- [Forecast.solar](#): API sencilla orientada a la obtención de previsiones de energía solar.

Desgraciadamente, la limitación del plan de precios impide seleccionar libremente el servicio deseado. Todos los servicios ofrecen distintos planes según el uso previsto. La gran limitación es el número de llamadas a la API en un periodo de tiempo. Los distintos planes también difieren en el horizonte temporal máximo de la previsión meteorológica, pero para este caso, al necesitar únicamente una hora de predicción no supone un problema. La Tabla 7 ofrece una comparativa entre los distintos servicios:

Servicio	Ventajas	Inconvenientes	Máximo de peticiones del plan gratuito
<b>Solcast</b>	<ul style="list-style-type: none"> <li>• Previsión precisa y contrastada</li> <li>• Buena documentación</li> </ul>	<ul style="list-style-type: none"> <li>• Precio (99 USD/mes)</li> </ul>	<ul style="list-style-type: none"> <li>• 10/día</li> </ul>
<b>AccuWeather</b>	<ul style="list-style-type: none"> <li>• Uso extendido</li> <li>• Multitud de funciones</li> <li>• Buena documentación</li> </ul>	<ul style="list-style-type: none"> <li>• Necesidad de incluir el logo en la aplicación o producto</li> <li>• Precio (25 USD/mes)</li> </ul>	<ul style="list-style-type: none"> <li>• 50/día</li> </ul>
<b>Forecast.solar</b>	<ul style="list-style-type: none"> <li>• Sencillez</li> <li>• Precio 12 EUR/año</li> </ul>	<ul style="list-style-type: none"> <li>• Mala documentación</li> </ul>	<ul style="list-style-type: none"> <li>• Sin datos</li> </ul>

Tabla 7. Análisis de alternativas de APIs

Si bien podría considerarse adquirir un plan de pago en el caso de desarrollar más de un dispositivo, al tratarse este trabajo de un TFM con un único prototipo, se va a priorizar el servicio que mejor plan gratuito ofrezca.

Se selecciona la API de AccuWeather por cumplir el requisito de las 24 peticiones necesarias al día (1 por hora). Solcast se descarta por no cumplir y además, en caso de querer desarrollar más este trabajo, su precio desorbitado no puede competir con el resto de alternativas. Para desarrollar futuros trabajos, se recomienda el servicio de Forecast.solar por su precio económico al año, a pesar de la documentación deficiente. No hay datos en el sitio web sobre el número de llamadas máximo, pero tras probar el servicio se descubre que está limitado a unas 10 llamadas al día.

### 5.3 Uso de la API

Para probar la API antes de implementarla en el código del algoritmo se hace uso del servicio que ofrece [Postman](#). Postman es una aplicación usada para testear APIs a través de una cómoda interfaz web.

El primer paso para hacer uso de la API de AccuWeather es crear una cuenta gratuita. A continuación, es necesario solicitar una API key que permita el uso de los métodos de la API, Al ser una cuenta gratuita únicamente se puede hacer uso de una key. Una vez obtenida, se requiere conocer la clave de la localización en la que se va a utilizar el servicio. Para este TFM se va a asumir en todo caso que la localización es la ciudad de Valencia. Para futuras aplicaciones se podría obtener la clave en función de un campo modificable por el usuario, a costa de incrementar el número de peticiones necesarias a la API.

Una vez obtenida la clave de Valencia pueden hacerse uso de las funciones integradas. Para este caso se hace uso del método GET “1 Hour of Hourly Forecasts”, que devuelve exactamente la predicción de la hora siguiente al momento de la petición [4]

**GET** 1 Hour of Hourly Forecasts

Returns forecast data for the next hour for a specific location. Forecast searches require a location key. Please use the Locations API to obtain the location key for your desired location. By default, a truncated version of the hourly forecast data is returned. The full object can be obtained by passing "details=true" into the url string.

Resource URL  
<http://dataservice.accuweather.com/forecasts/v1/hourly/1hour/{310683}>

Query Parameters

Name	Values	Description
<b>apikey</b> <i>(required)</i>	<input type="text"/>	Provided API Key
<b>language</b>	<input type="text" value="en-us"/>	String indicating the language in which to return the resource
<b>details</b>	<input type="text" value="true"/>	Boolean value specifies whether or not to include full details in the response.
<b>metric</b>	<input type="text" value="false"/>	Boolean value specifies whether or not to display metric values.

[Send this request](#)  
using the values above [Reset](#)

Figura 40. Método GET 1 Hour of Hourly Forecasts de AccuWeather [4]

Al realizar la petición los datos se devuelven en formato JSON. Este tipo de formato es fácil de interpretar después, y Python dispone de librerías que permiten trabajar con él de forma sencilla. Si se atiende a la respuesta completa al realizar la petición se obtienen multitud de datos que no son necesarios para esta aplicación. Por tanto, el único dato que se conservará es el del valor de la irradiancia solar. La Figura 41 ofrece información sobre el proceso de obtención del dato deseado, realizado a través de Postman:

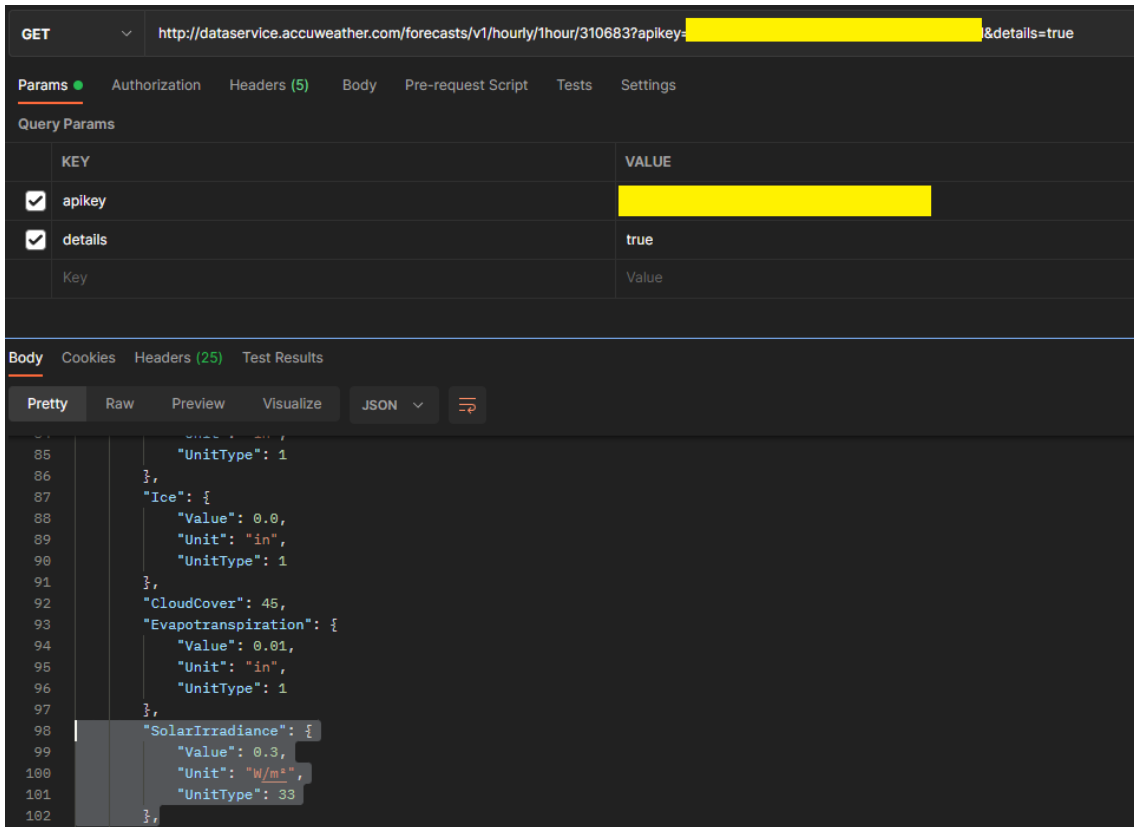


Figura 41. Obtención de la irradiancia a través de la API de AccuWeather

Es importante destacar que, observando las unidades devueltas, es bastante probable que exista una errata en las unidades de la irradiancia, ya que 0.3 W/m<sup>2</sup> es un valor inusualmente bajo. Se considera que realmente la medida se toma en kW/m<sup>2</sup>.

## 5.4 Integración en Python

Se deben realizar peticiones a través de la API desde el código periódico que gestionará el algoritmo de carga de baterías. Por tanto, se debe utilizar código Python para usar la API. Postman es adecuado para familiarizarse con nuevas APIs o testeo, pero a la hora de la verdad no se puede disponer de su interfaz agradable.

Por suerte, existen librerías para facilitar el uso. La librería `requests` es ampliamente usada para este fin. Se va a probar su uso a través de un libro en Jupyter Lab a fin de documentar esta sección, pero posteriormente el código deberá ser incluido en el script final.

```

import requests

# API parametros
baseurl = 'http://dataservice.accuweather.com/forecasts/v1/hourly/1hour/'
location_key = '310683'
apikey = '-----'

parameters = {
    'apikey': apikey,
    'details': 'true'
}

url = baseurl + location_key

response = requests.get(url, params=parameters)
    
```

*Fragmento de código 27. Uso de la API*

La librería es muy sencilla de usar. Basta con indicar la URL base y los parámetros necesarios para realizar la petición. La respuesta se guarda en el objeto `response`. Los datos se pueden obtener en formato json a través del método de clase `json()`. Del largo diccionario que se ofrece, únicamente interesa el campo “SolarIrradiance”.

```

response.json()[0]['SolarIrradiance']

{'Value': 0.0, 'Unit': 'W/m²', 'UnitType': 33}
    
```

*Fragmento de código 28. Respuesta de la API (1)*

El campo deseado también es un diccionario, por lo que para acceder a la medida basta con filtrar por el campo `['Value']`:

```

response.json()[0]['SolarIrradiance']['Value']

0.0
    
```

*Fragmento de código 29. Respuesta de la API (2)*

Ya se dispone del dato necesario para trabajar con el algoritmo.

## 6 DIMENSIONAMIENTO DE LA BATERÍA Y LA SUPERFICIE DE PANELES FOTOVOLTAICOS

### 6.1 Introducción

El software que se quiere desarrollar no tiene como objetivo dimensionar ni el array fotovoltaico ni la capacidad de la batería. Sin embargo, es necesario disponer de estos datos antes de realizar cualquier cálculo. En un proyecto completo, la idea es que el usuario introduzca estos valores en el programa para que el cálculo se adapte a cada situación.

Para este caso, al no disponerse de la instalación de forma previa, se va a realizar una estimación de una instalación fotovoltaica típica adaptada al tipo de vivienda de la que provienen los datos de entrenamiento. A fin de ajustarse lo máximo posible a un caso real, y que se den las dos situaciones posibles (consumo mayor que generación y viceversa), la decisión se toma tras observar detenidamente los datos de entrenamiento.

### 6.2 Dimensionamiento de los paneles fotovoltaicos

Remontándonos a los datos originales, antes de sufrir cualquier tipo de transformación, se tiene información sobre la potencia media demandada en cada minuto. Uno de los primeros pasos era redimensionar el dataset en horas, así que se procede de la misma manera.

```
df=df['Global_active_power'].resample('h').mean()

df.describe()

count      34168.000000
mean         1.091728
std          0.897619
min          0.124000
25%         0.341925
50%         0.802850
75%         1.579342
max          6.560533
Name: Global_active_power, dtype: float64
```

*Fragmento de código 30. Información de los datos de consumo por horas*

Puede obtenerse información interesante mediante la función *describe()*. Se puede concluir que, de forma media por horas, con 1.57 kW sería suficiente para cubrir el 75% de los consumos. De todas maneras, este razonamiento no es lo suficientemente elaborado como para dimensionar los paneles.

Por un lado, se ve claramente que el valor máximo supera con creces el percentil 75, lo que indica que el patrón de consumo podría presentar picos puntuales en intervalos específicos que no estarían cubiertos por la generación. Por otro lado, no tiene ningún sentido evaluar todos los consumos sin distinguirlos por horas. La generación solar únicamente se da en las horas con luz solar, y es ahí donde es interesante analizar el consumo.

Es posible visualizar los datos en función de la hora del día. De esta manera se puede comprobar rápidamente cómo está distribuida la potencia demandada a lo largo del día. Los valores se han obtenido como la media de todos los datos contenidos en el dataset. La Figura 42 representa la media del consumo junto con el intervalo de confianza del 95%. Solo con la media no se tiene información suficiente, ya que la existencia de valores muy dispares puede falsear la interpretación. Se añade al gráfico el valor del percentil 75% para cada hora del día.

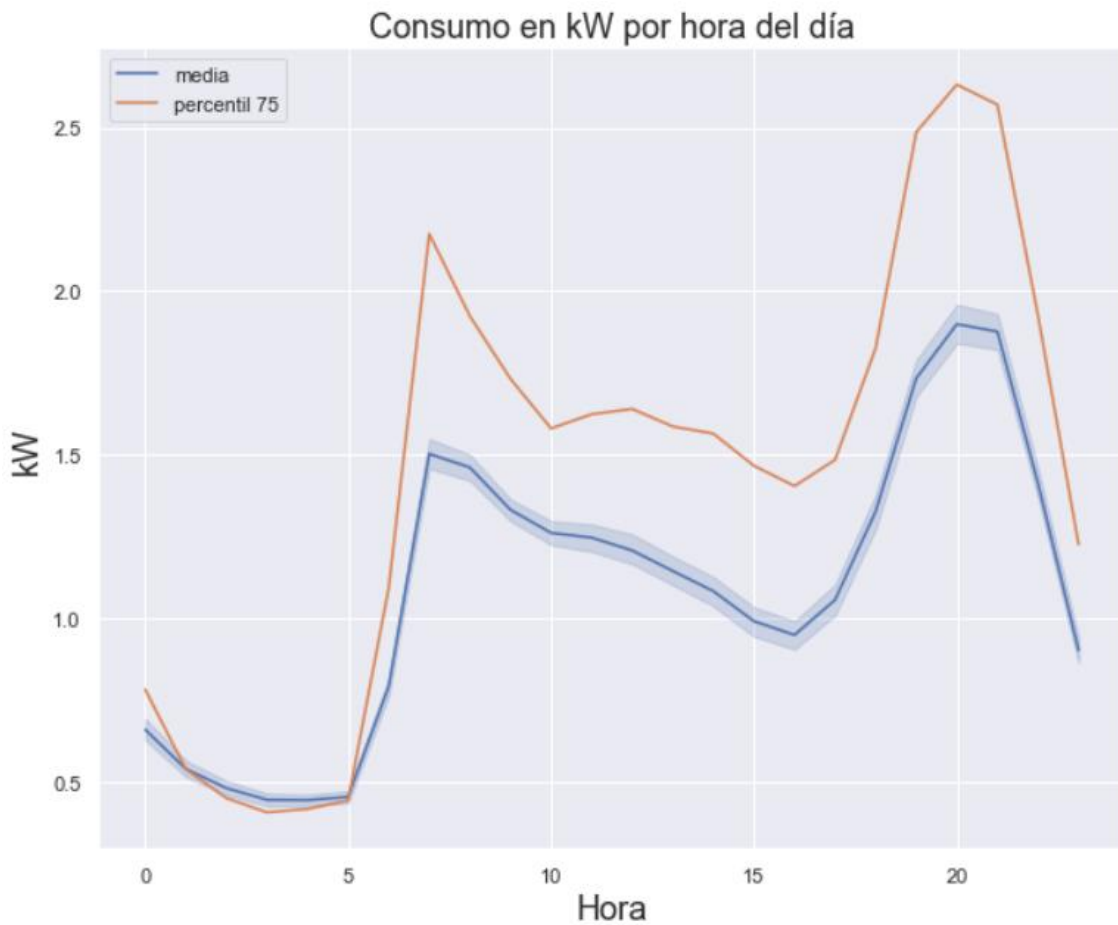


Figura 42. Media y percentil 75 del consumo a lo largo del día

También pueden obtenerse estadísticas útiles sobre esta nueva agrupación por horas



```
consumo_horas_dia = df.groupby(df.index.hour).mean()

consumo_horas_dia.describe()

count      24.000000
mean       1.091485
std        0.444124
min        0.443844
25%       0.758595
50%       1.113610
75%       1.351902
max        1.899073
Name: Global_active_power, dtype: float64
```

*Fragmento de código 31. Información del consumo por horas*

Puede apreciarse que al realizar la media por horas los valores no difieren tanto como en el caso anterior. De la gráfica de la Figura 42 se puede obtener una deducción importante: los picos de consumo no se corresponden con las horas de mayor generación. Por tanto, si el objetivo es determinar una potencia tal que para que se den situaciones tanto de superávit de generación como de mayor consumo, hay que dimensionar la generación fotovoltaica para las horas de sol. El objetivo es conseguir, de forma generalizada, que durante las horas de luz la generación esté ligeramente por encima del consumo, para cargar las baterías, y aprovechar más tarde ese excedente para cubrir las horas sin generación.

No es objetivo dimensionar para que la vivienda sea autosuficiente, por lo que no es un requisito que la generación excedente cubra el consumo del resto del día. Podría imponerse esto como requisito si se quisiera una casa completamente autosuficiente, pero la superficie de paneles solares sería extremadamente elevada para una vivienda con este perfil de consumo. Además, al disponer de conexión a red y tener la posibilidad de cargar las baterías por la noche, no hay ninguna necesidad de generar tanta energía.

Por tanto, con la información disponible, se propone una instalación que sea capaz de generar aproximadamente entre 1.5 y 2 kW pico, de tal manera que durante las horas de luz se genere más potencia de la consumida. La Figura 43 añade al análisis anterior la banda de potencia propuesta.

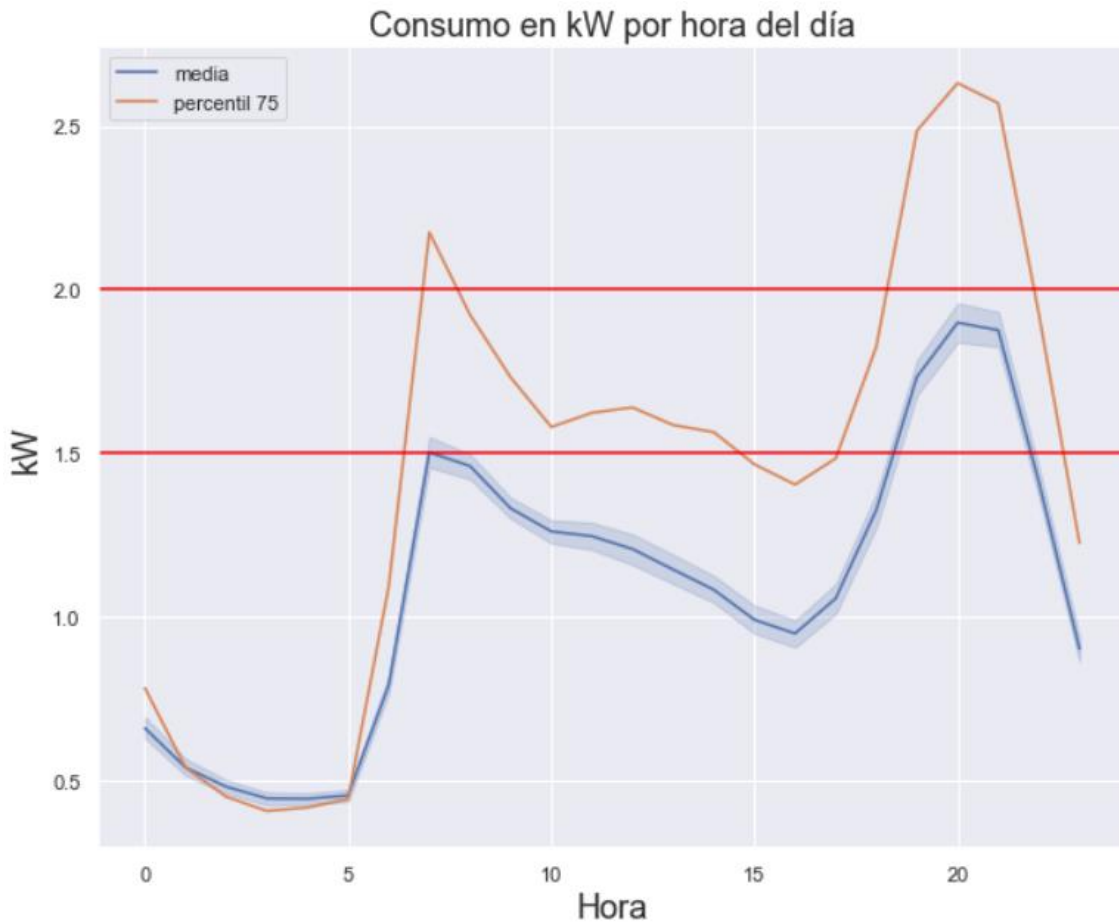


Figura 43. Media por horas y percentil 75 del consumo a lo largo del día, potencia propuesta

De esta manera, se asegura que, según la potencia escogida, se supera el consumo para la media de los valores horarios o para el 75% de los consumos horarios. Se decide trabajar con 2 kW ya que es deseable que en la mayoría de los días los paneles cubran el consumo típico.

Para determinar cómo se distribuye la generación a lo largo del día hace falta recurrir a datos meteorológicos. La base de datos europea PVGIS [15] ofrece herramientas sencillas para consultar datos históricos acerca de la radiación solar. Por ejemplo, pueden obtenerse datos de un año típico en la ciudad de Valencia para consultar el perfil de la radiación a lo largo de un año

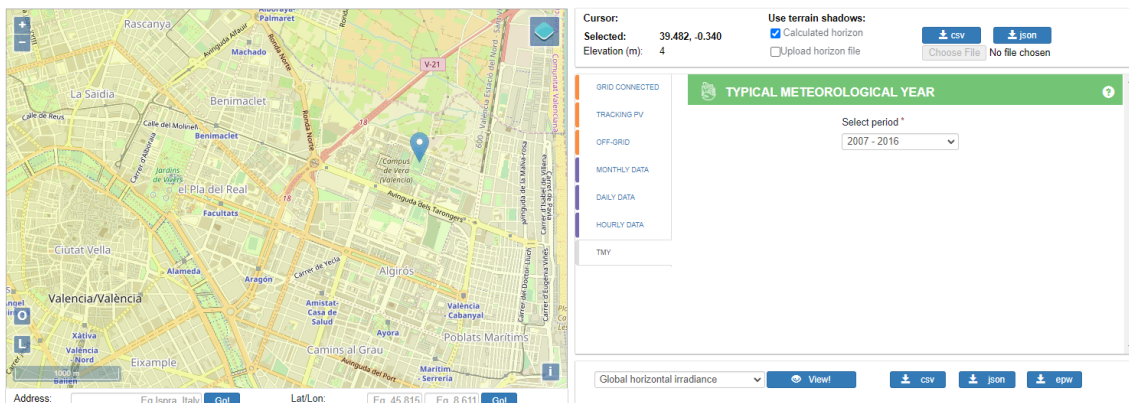


Figura 44. Herramienta “año meteorológico típico”, PVGIS [15]

Introduciendo los datos de localización, tal y cómo aparece en la Figura 44, se obtiene un gráfico interactivo con la irradiación global en un año. El resultado aparece en la Figura 45.

Sin embargo, aunque puede realizarse zoom a escala diaria, no ofrece datos estadísticos en función de la hora del día. Se podría tomar el valor de la radiación tal y como aparece en el gráfico para cierto momento del año, como verano en el caso optimista o invierno en el optimista. No obstante, se decide abordar el problema de otra manera.

Es posible descargar los datos de irradiancia horarios en formato csv. Esto permite la manipulación y visualización a través de Python. Se cargan los datos en un libro *Jupyter* tal y como se ha hecho anteriormente con los datos de entrenamiento. Los datos corresponden a los valores horarios del año 2016.

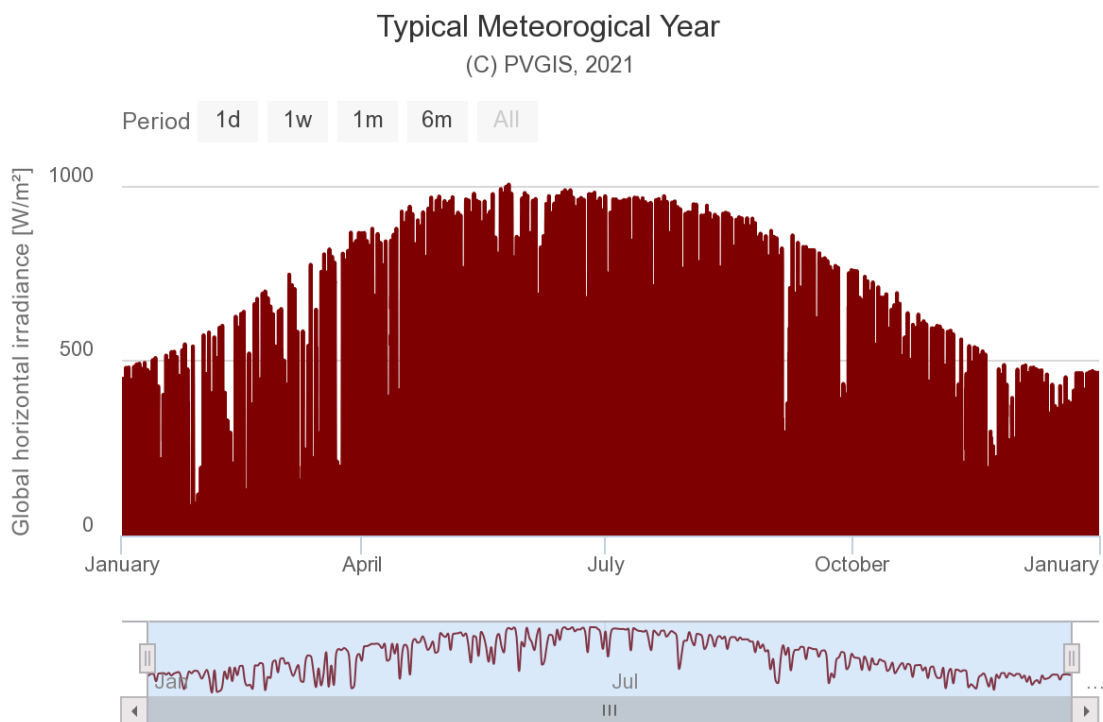


Figura 45. Año meteorológico típico, irradiancia global [15]

```
#Lectura del archivo

string = '%Y%m%d:%H%M'

df = pd.read_csv('radiationdata.csv', sep=',',
                parse_dates={'Fecha' : ['time']},
                infer_datetime_format=True,
                low_memory=False,
                na_values=['nan', '?'],
                index_col='Fecha')

string = '%Y%m%d:%H%M'
df.index = pd.to_datetime(df.index, format=string)
df['hour'] = df.index.hour

df=df[['G(i)', 'hour']]
```

*Fragmento de código 32. Lectura de los datos de radiación*

Se puede graficar la irradiancia diaria frente a la hora. La Figura 46 representa la media de la irradiancia junto a su intervalo de confianza del 95%. Si bien el valor máximo de irradiancia puede variar mucho de una estación a otra, se comprueba que el máximo siempre se da en las mismas horas

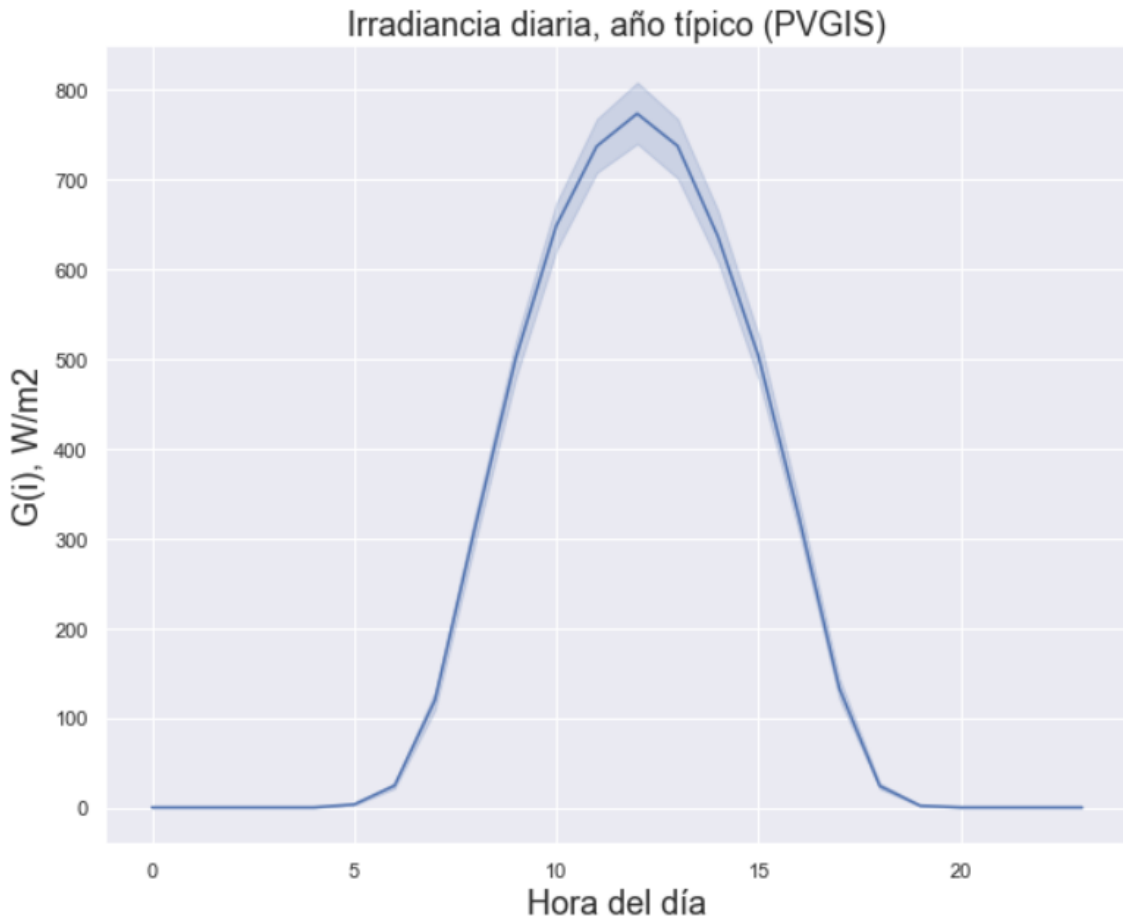


Figura 46. Irradiación diaria en 2016, PVGIS

La irradiación determina la producción eléctrica de los paneles solares. Al no disponer de especificaciones de fabricante, se toma como aproximación que la potencia generada es un 15% de la irradiación global, que es un valor de rendimiento muy conservador para paneles fotovoltaicos. Por lo general, los modelos comerciales presentan rendimientos mucho mayores, por encima del 20%, pero tomando un rendimiento menor también se consideran otros factores que afectan a la producción y no pueden calcularse debido a la sencilla aproximación que se realiza.

También hay que determinar la superficie de paneles deseada. De forma aproximada, puede calcularse la superficie necesaria como:

$$S = \frac{P_{pico}}{G_{pico} \cdot \mu} = \frac{2000 \text{ W}}{800 \frac{\text{W}}{\text{m}^2} \cdot 0.15} \approx 17 \text{ m}^2$$

Se ha redondeado el resultado a fin de facilitar el análisis.

Con datos de superficie y rendimiento, ya es posible comparar el perfil típico de consumo con el perfil típico de generación.

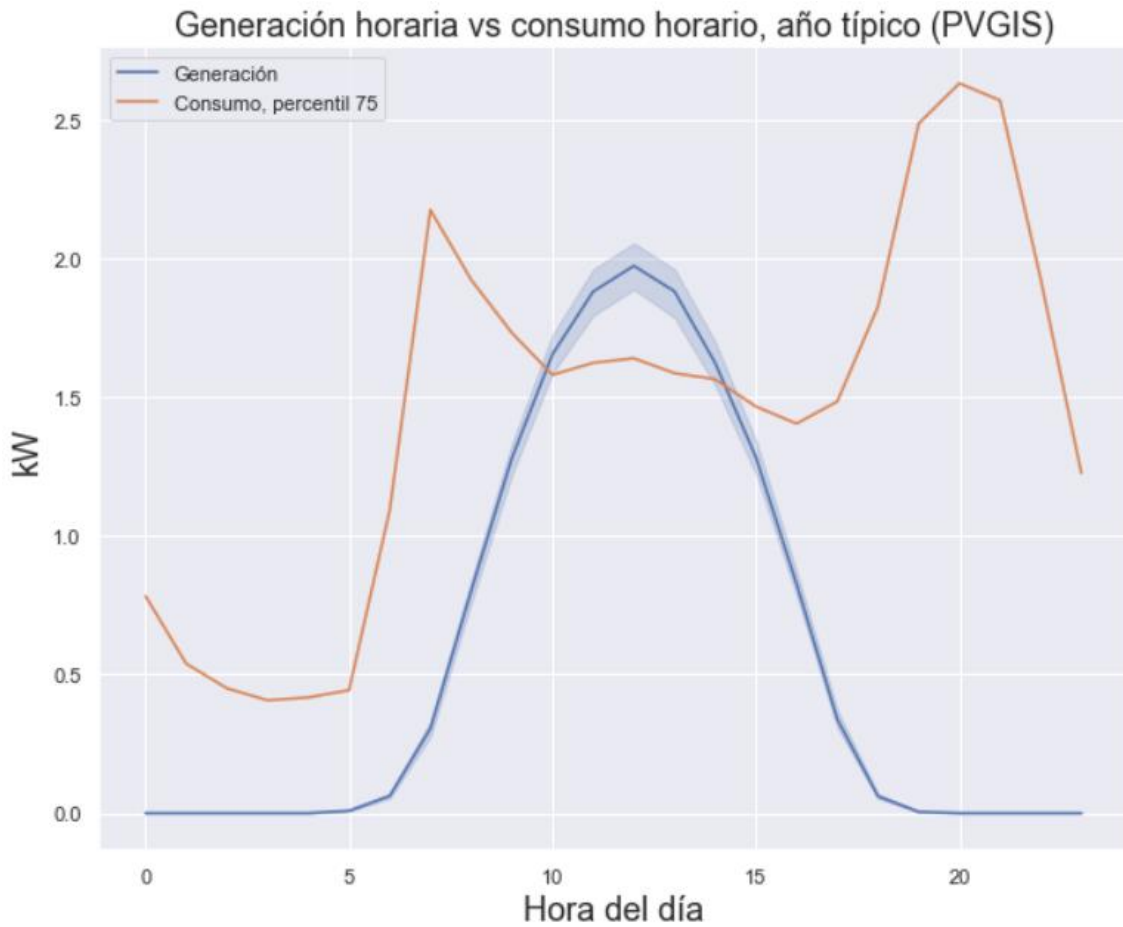


Figura 47. Consumo vs generación horaria, año típico.

De esta manera se asegura que para el 75% de los valores en cada hora la generación superará al consumo en las horas de mayor irradiancia. Hay que recordar que el análisis se ha realizado con datos estadísticos de consumo y generación, por lo que realmente el resultado en tiempo real puede variar bastante.

### 6.3 Dimensionamiento de la batería

Si se restan las gráficas anteriores se obtiene el saldo de potencia por horas. La Figura 48 muestra el resultado. Si se realiza la integral de la curva (sumatorio de todas las horas) se tiene el saldo de energía para un día. De esta manera, se puede tomar rápidamente una decisión sobre la capacidad de las baterías.

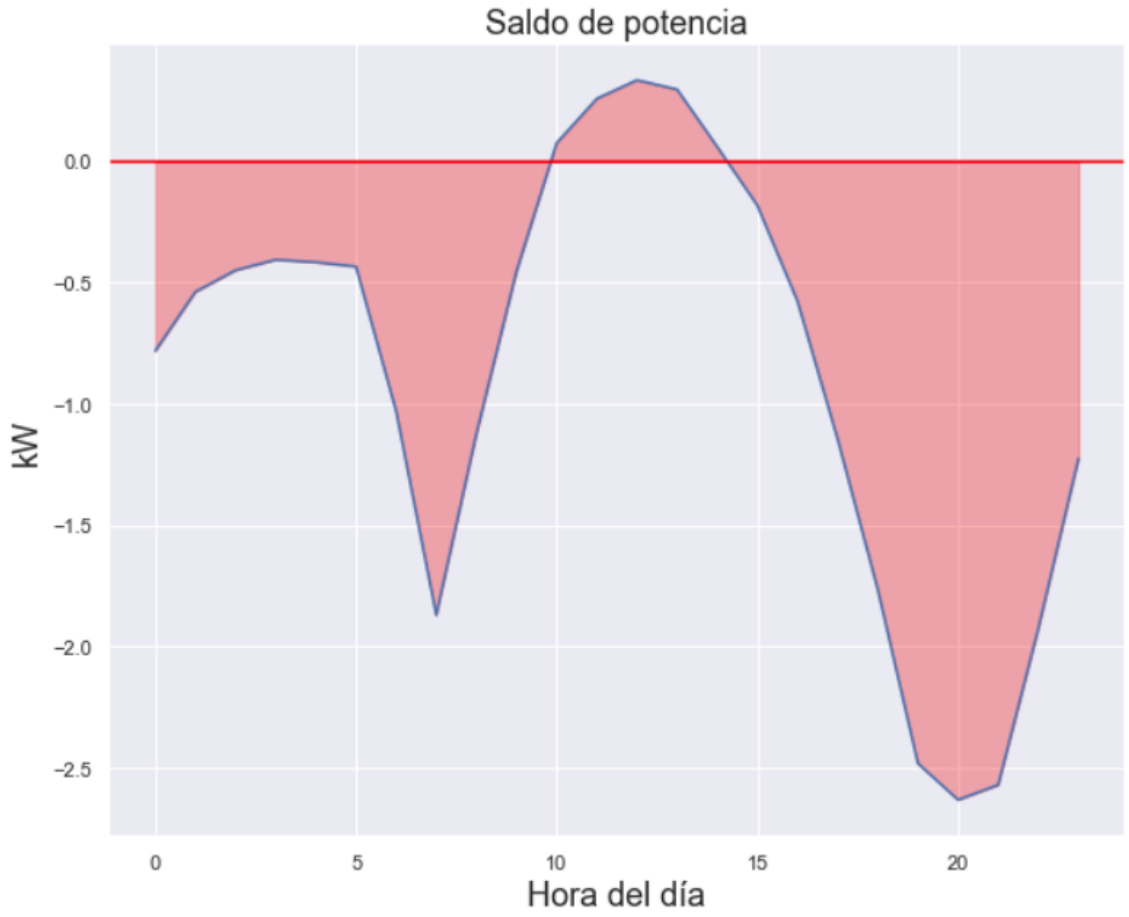


Figura 48. Saldo de potencia diario

Si bien la gráfica muestra el saldo total para un día, hay que considerar que durante los períodos valle (en los que la electricidad es más barata) la batería siempre está en estado de carga, y todo el consumo será satisfecho a través de la acometida de red. De esta manera, se ahorra tanto en precio de electricidad, ya que en el periodo valle se asegura que se obtiene la electricidad al mínimo precio posible, y además, en el tamaño de la batería. Al eliminar esta franja horaria del análisis solo debe considerarse el saldo durante las horas en las que se va a emplear la batería.

Hay que realizar algunas modificaciones al *dataframe* resultante para adaptarlo a estas condiciones. En primer lugar, deben eliminarse las entradas correspondientes al periodo valle. A fecha de elaboración de este TFM, de manera general las tarifas domésticas tienen el precio segmentado por tramos, siendo estos punta, llano y valle, ordenados de más caro a más barato según el precio del kWh. La Figura 49 muestra de manera resumida los periodos establecidos.



Figura 49. Periodos horarios de facturación en la nueva factura. Fuente y elaboración: CNMC

Hay que considerar que debido a que la potencia está calculada como la media de la hora anterior, para eliminar las horas correspondientes hay que sumar 1 a las horas límite. Por ejemplo, el dato de potencia a las 9 corresponde a la medida durante la hora anterior, de 8 a 9.

```

start_valle = 0
end_valle = 8
index = list(range(start_valle+1,end_valle+1))

day_diff = diff[diff.index.difference(index)]
day_diff

Fecha
0    -0.782217
9    -0.453693
10   0.074019
11   0.256911
12   0.333388
13   0.295071
14   0.058772
15  -0.182795
16  -0.577356
17  -1.145550
18  -1.765000
19  -2.481072
20  -2.631333
21  -2.569842
22  -1.925783
23  -1.225742
dtype: float64
    
```

Fragmento de código 33. Diferencia de potencia por horas



Una vez eliminado el periodo valle, es necesario reordenar los datos para que la hora 00:00 se reemplace por "24" para que al dibujar la gráfica no aparezca en primer lugar.

```
reindex = day_diff.index[1:].to_list()
reindex.append(day_diff.index[0])
day_diff_re = day_diff.reindex(reindex)
day_diff_re.index.values[-1] = 24
day_diff_re
```

Fecha	
9	-0.453693
10	0.074019
11	0.256911
12	0.333388
13	0.295071
14	0.058772
15	-0.182795
16	-0.577356
17	-1.145550
18	-1.765000
19	-2.481072
20	-2.631333
21	-2.569842
22	-1.925783
23	-1.225742
24	-0.782217

dtype: float64

*Fragmento de código 34. Diferencia de potencia por horas, excluidas horas valle*

Se debe tener cuidado de reordenar el índice a la vez que la serie de datos. Una vez lista la serie, puede grafarse de la misma manera que se ha hecho anteriormente:

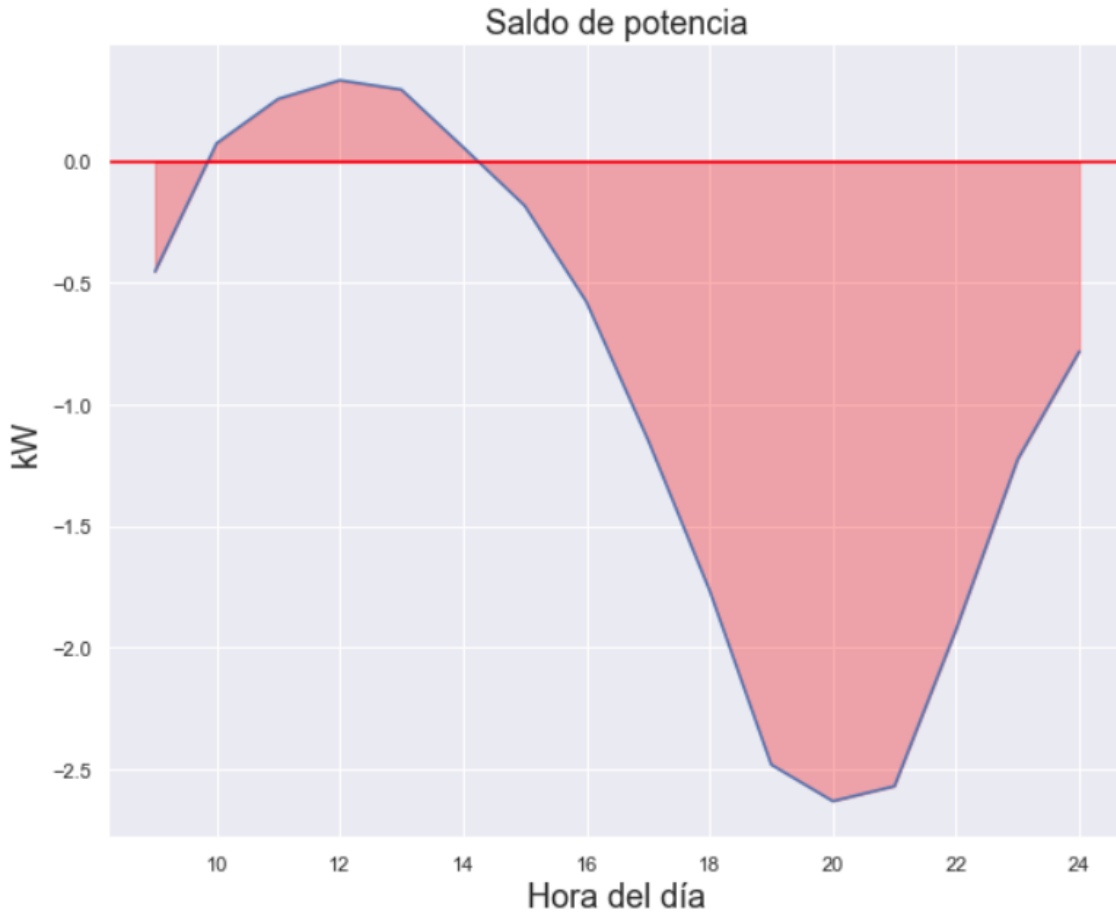


Figura 50. Saldo de potencia diario excluyendo el periodo valle

Sumando todos los periodos, al ser potencia por horas, se tiene el saldo total en un día durante el periodo en el que está funcionando la batería:

```
capacidad=day_diff_re.sum()
print(capacidad)
-14.72222377049185
```

Puede concluirse que la batería necesita entregar aproximadamente 15 kWh cada día. La batería puede dimensionarse con ese valor, que además, es un valor normalizado en el mercado

$$C_{baterías} \approx 15 \text{ kWh}$$

Sin embargo, existe un problema que surge al cargar la batería al 100% por la noche. Si se observan las primeras horas de funcionamiento del día se puede realizar un gráfico aproximado que explica la situación.

En la Figura 51 se aprecia el problema. Si se empieza la jornada con la batería cargada al 100%, se da la situación a las pocas horas de funcionamiento que la energía acumulada es positiva durante ciertas horas del día, lo que implica que se excede la capacidad máxima de la batería. La solución es sencilla: limitar la carga de la batería por la noche para que exista capacidad suficiente para absorber el excedente generado por los paneles.

## Energía acumulada en batería vs saldo energético

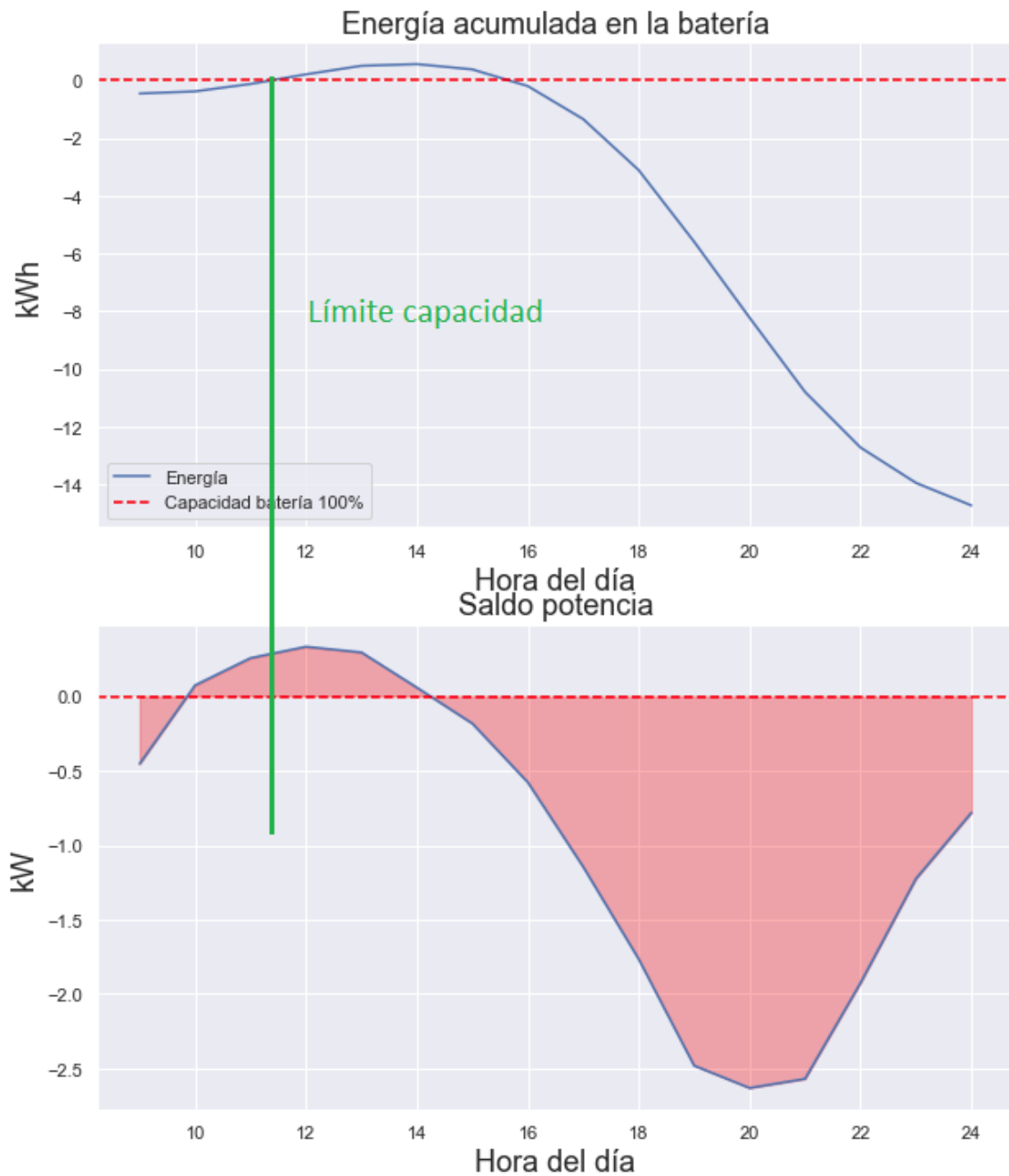


Figura 51. Comparación horaria del saldo de potencia con la carga de la batería

El excedente puede obtenerse como el valor máximo positivo de la serie “Energía acumulada en la batería”:

```
max(energy)
0.5644676038251333
```

Si se considera este valor sobre la capacidad total se tiene el porcentaje máximo de carga al que hay que limitar durante el periodo valle:

$$\%Carga_{max} = \left(1 - \frac{0.56}{15}\right) * 100 = 96.26\%$$

Para garantizar el buen funcionamiento en cualquier situación, se decide rebajar este porcentaje a un 85%, ya que al realizar el análisis con datos estadísticos nada asegura que para ciertos días la generación sea mucho mayor o el consumo más reducido. Además, encaja con uno de los objetivos, evitar que la batería se cargue o descargue por completo, beneficiando enormemente a la vida útil de la batería.

Para obtener la gráfica nueva de la Figura 51 se ha empleado el sumatorio acumulado por horas. El resultado se ha conseguido mediante el siguiente trozo de código:

```
energy = []
last_value = 0
values = []
for x in day_diff_re:
    value=x+last_value
    energy.append(value)
    last_value = value
print(energy[-1])
print(capacidad)

-14.722222377049183
-14.722222377049185
```

*Fragmento de código 35. Capacidad requerida y acumulación por horas*

Puede comprobarse que el resultado es correcto comprobando el último valor con el sumatorio realizado anteriormente.

## 7 ALGORITMO DE GESTIÓN DE ENERGÍA

### 7.1 Introducción

El algoritmo corresponde a la parte final y más importante de este proyecto. De su éxito depende que se cumpla el objetivo del dispositivo. Con acceso a las predicciones, tanto de consumo como de generación, es posible idear un algoritmo que tome decisiones gracias a la información de la que se dispone. El objetivo es idear un método que, debidamente justificado, mejore la gestión energética de la vivienda.

### 7.2 Planteamiento

Volviendo a la Figura 3 que se mostraba al principio del documento, se vuelve a recordar la idea principal del algoritmo. Hay que definir la comparación entre el consumo y la generación. Se pueden dar las siguientes situaciones:

- Consumo > Generación: Si la generación no puede cubrir el consumo, se deben cargar las baterías
- Consumo  $\leq$  Generación: No hace falta cargar las baterías ya que la energía sobrante puede almacenarse

Se decide trabajar en una escala temporal de horas para realizar las comparaciones. Esta decisión viene tomada por dos motivos que la justifican:

- Reducir al máximo el número de ciclos de carga y descarga completos de la batería: Modificando la salida del algoritmo cada hora se asegura que el porcentaje carga de batería no oscila demasiado a lo largo del día, con lo que se alarga la vida útil de la misma.
- Limitar el número de predicciones a futuro: Cuanto menos se aleje la predicción del momento en el que se ejecuta más precisos será el modelo. En este caso, al trabajar con la información en horas, lo más sencillo y preciso es obtener la predicción de la hora siguiente.

### 7.3 Diagrama de flujo

Antes de comenzar a programar el código es importante tener una visión general de qué es lo que debe hacer la aplicación. Para facilitar el seguimiento del lector, la mejor forma de entender el programa es realizar un diagrama de flujo en pseudocódigo con el flujo de ejecución de las funciones más importantes.

La Figura 52 muestra los pasos a seguir por el algoritmo. La mayoría de las funciones se han ido desarrollando a lo largo de este TFM, con lo que ahora solo resta unificarlas en un mismo script que se ejecute de forma indefinida.

Se puede comprobar que el programa no tiene una instrucción de fin, dado que para que el algoritmo funcione necesita estar en continuo funcionamiento las 24 horas del día. La única forma de terminar con el programa es mediante el apagado del dispositivo, que no debería

suceder bajo condiciones normales. Es importante asegurarse que el script comience de forma automática con el encendido del equipo para evitar problemas de funcionamiento. Esto puede realizarse mediante la herramienta *Cron* que ofrece Linux.

**Diagrama de flujo del algoritmo**

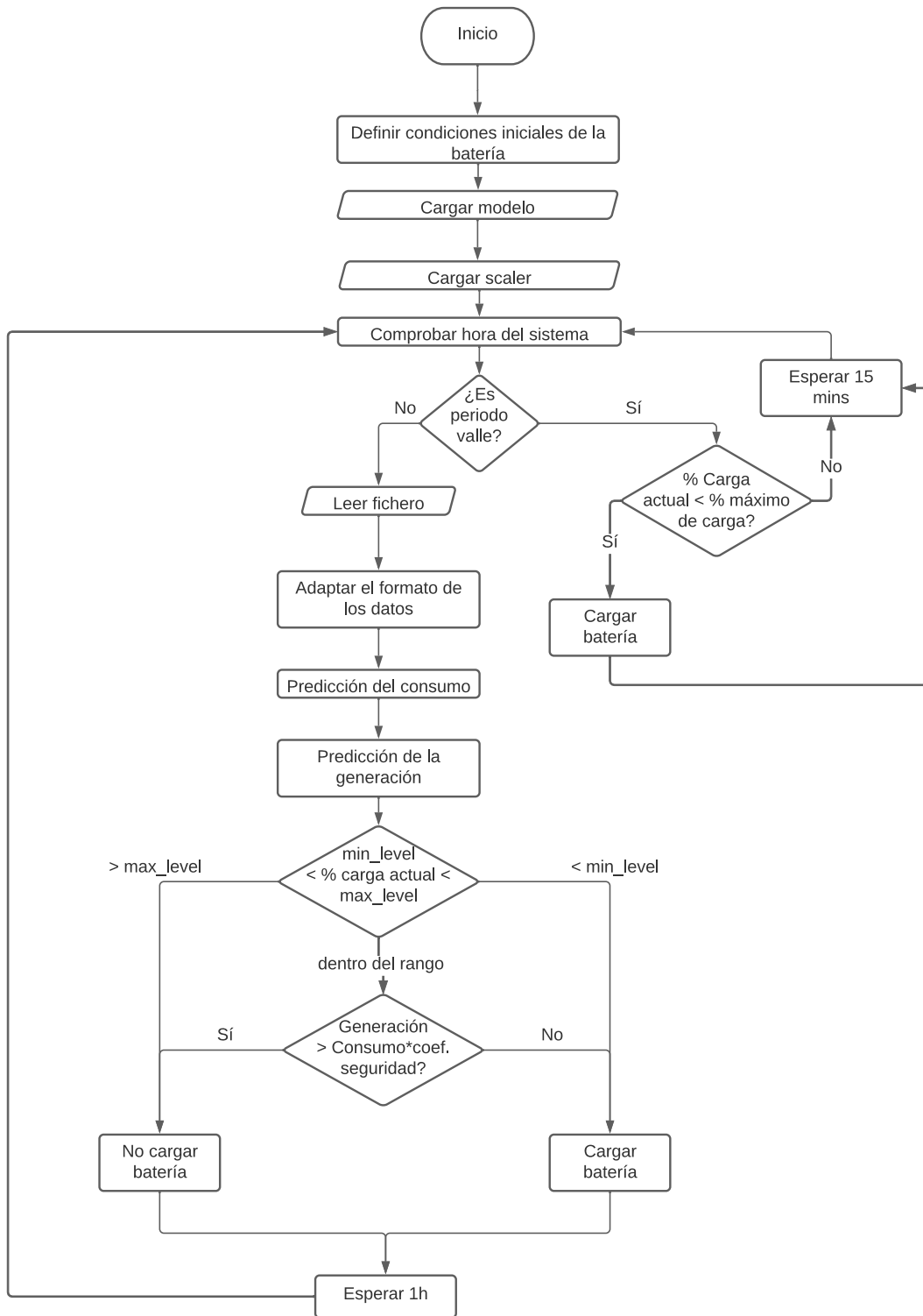


Figura 52. Diagrama de flujo del programa

El primer paso será realizar las tareas de setup. Estas incluyen cargar el modelo y el *scaler* guardados anteriormente y definir una serie de propiedades iniciales de la batería. A continuación, hay que determinar si el instante de ejecución corresponde a una hora valle o no. El programa cambia radicalmente según el periodo horario.

Si es hora valle, el algoritmo se limitará a cargar la batería, comprobando cada 15 minutos que la capacidad de la batería no excede el máximo establecido en las condiciones iniciales. Estos 15 minutos se han seleccionado de manera que el programa no consuma demasiados recursos solicitando una nueva comprobación cada poco tiempo, pero podrían ajustarse según la velocidad de carga de la batería.

Si no es una hora valle, se entra dentro del flujo de ejecución normal del programa. Es necesario obtener tanto la predicción del consumo como de la generación. Primero, se cargan y se adaptan los datos para obtener la predicción del consumo con el modelo previamente cargado. A continuación, se solicita a la API de AccuWeather la previsión de la generación para la hora siguiente. Una vez obtenidos, se comparan los resultados y se toma la decisión. El bucle se ejecuta una vez cada hora.

## 7.4 Código

Se va a explicar de forma resumida los aspectos más importantes del código de la aplicación. Todo el código fuente puede encontrarse en el documento anexo, así como en el repositorio público de [GitHub](#) del proyecto. En este enlace aparece toda la información y archivos necesarios para la instalación del programa en cualquier dispositivo con compatibilidad con Python. Es de licencia abierta con lo que cualquier usuario puede hacer uso del mismo, así como sugerir modificaciones mediante una *pull request* a través de la popular red social de programadores.

A fin de garantizar la escalabilidad y legibilidad del código, la mayoría de las funciones relacionadas con la gestión de la batería se han incluido en la clase `SmartAlgorithm`. Al crear el objeto de la misma clase, en el constructor se inicializan los valores internos de configuración necesarios para el correcto funcionamiento de la aplicación. Además, se incluye una clase interna con el nombre `Battery` que contiene información sobre los parámetros de la batería. Estos son:

- **battery\_level**: El nivel actual de porcentaje de batería. Inicialmente se establece al 50%. No tiene uso para esta aplicación, pero es interesante incluirlo para futuras aplicaciones en las que se pueda incluir el nivel de la batería para realizar acciones extra.
- **max\_level\_night**: Máximo nivel de carga permitido en las horas valle. Se inicializa al 85% según lo explicado en el apartado Dimensionamiento de la batería.
- **cargar\_bateria**: La variable de salida que determina si se debe o no cargar la batería en ese momento. 1 es para cargar y 0 para no actuar.
- **max\_level\_day**: Máximo nivel de carga permitido durante el día. A fin de preservar la vida útil de la batería, se recomienda que el nivel de carga se encuentre entre un 20 y un 80% durante el funcionamiento normal en el que somete a cargas y descargas continuamente. Se establece en un 80%.
- **min\_level\_day**: Por las mismas razones explicadas anteriormente, se inicializa a un 20%

Una vez definida la clase batería, pueden establecerse los parámetros particulares de esta aplicación, que se encuentran inicializados en el constructor de clase:

- **model:** El modelo de IA guardado anteriormente. Puede cargarse mediante la función de *keras load\_model*.
- **scaler:** Similar al modelo, el escalado generado durante el entrenamiento.
- **filepath:** la ruta del archivo desde el que se van a leer los datos. Por defecto se encuentra en la carpeta *Data* del proyecto
- **n\_hours:** Número de horas consideradas para la ventana temporal de la red LSTM.
- **sup\_paneles:** Metros cuadrados de superficie de paneles solares.
- **rendimiento:** rendimiento de los paneles solares.
- **x\_security:** Coeficiente de seguridad empleado en la comparación entre el consumo y la generación
- **irradiance:** es necesario guardar el dato de la irradiancia para que pueda ser compartido entre todas las funciones de la clase. Los motivos están explicados en la función *get\_irradiance\_next\_hour*

El resto de las funciones pueden encontrarse en el documento anexo. Todas se han realizado en base a los apartados ya explicados en este documento y son auto explicativas. Las que requieren de mayor explicación se encuentran a continuación:

#### ***is\_valley:***

Tomando la fecha actual en el momento de la llamada, devuelve un 1 si se encuentra en zona horaria valle o un 0 si no lo es. Es de vital importancia para distinguir el flujo de ejecución. De momento el proyecto no contempla considerar los días festivos como días valle. Puede considerarse como una mejora de cara a desarrollar el producto completo.

Si bien se incluye la posibilidad en el código de distinguir como periodo valle los fines de semana, no tiene sentido incluirlos en el flujo de ejecución del algoritmo, ya que si se considerasen como valle durante la totalidad del día, se estaría desperdiciando el excedente de generación.

#### ***adapt\_data:***

Ejecuta las operaciones necesarias sobre los datos para adaptar el formato correspondiente a la entrada del modelo de inteligencia artificial. Como entrada, toma los datos leídos a través de la función *load\_data*. El proceso de adaptación de los datos comprende:

- Limpieza de los datos: eliminación de valores NaN.
- Transformación de las variables necesarias para contar con el mismo formato que los datos de entrenamiento.
- Remuestreo del dataframe para contar con los datos horarios.
- Selección de los últimos *n\_hours* valores.
- Escalar los datos de acuerdo con el *scaler* guardado durante el entrenamiento.
- Uso de la función *series\_to\_supervised*. Necesaria para el correcto funcionamiento de la red LSTM.
- Redimensionar el input a 3d [muestras, secuencias, variables].



### ***set\_irradiance\_next\_hour:***

Debido a las limitaciones de la API, solo es posible obtener la predicción de la irradiancia para la hora siguiente al momento de la llamada, por lo que si se realiza la petición al mismo tiempo que se ejecuta el algoritmo (cada hora en punto), realmente la predicción obtenida no es para la hora que entra sino la siguiente. Este comportamiento debe evitarse y por tanto se crea este método de clase, cuya única función es guardar en un atributo el valor de la predicción.

El método debe ejecutarse minutos antes del algoritmo, antes de que finalice la hora en curso, para que la predicción sea correcta.

### ***setup\_scheduler:***

Gracias al uso de la librería *apscheduler* [16], es posible programar funciones para que se ejecuten de forma periódica en momentos determinados, de forma similar a la herramienta *cron* de linux. De esta manera, es posible configurar el bucle principal del programa para que se ejecute de forma horaria.

Además, es posible cambiar la frecuencia de ejecución de la rama de código que se ejecuta en periodo valle. Ya que es una comprobación que no requiere potencia de procesado, se establece en 15 minutos, no por determinar zona horaria en la que se está, sino para asegurarse de que la capacidad de la batería no excede el porcentaje de carga máximo durante la noche. Si se realizase la comprobación una vez cada hora, es posible que al cargar de forma continuada la batería pueda sobrepasarse el límite establecido.

También se programa el método *set\_irradiance\_next\_hour* para que se ejecuta de forma horaria en el minuto 55 de cada hora, por los motivos explicados anteriormente.

## **7.5 Caso de test**

Para su comprobación, a fin de poder realizar las pruebas independientemente del hardware y con volumen y variabilidad de datos suficiente, se ha hecho uso del mismo set de entrenamiento con el que se ha trabajado en el modelo. Para asemejar el funcionamiento el máximo posible a la realidad, se ha diseñado la función *adapt\_data\_test*, disponible en la clase *SmartAlgorithm*. Se ha incluido un script de test, con nombre *test.py* que usa esta función en lugar de la empleada en el funcionamiento real.

La forma de proceder de este método es, únicamente para la primera iteración, seleccionar un punto aleatorio que comparta hora con el momento de la llamada, de manera que el perfil de consumo generado sea similar en todos los casos.

Hay gran cantidad de días en el dataset que se pueden elegir, pero hay que asegurar que por lo menos existan 24 entradas anteriores al primer punto o de lo contrario no se tendrá suficiente información para realizar la primera predicción. Además, es preferible que no esté demasiado cerca del final del dataset o podría detenerse la ejecución antes de lo previsto. Estas dos consideraciones se tienen en cuenta para realizar la selección de un índice aleatorio.

No tendría sentido elegirlo de forma completamente aleatoria, ya que la tendencia es muy diferente según la hora del día. Una vez seleccionado, en las sucesivas ejecuciones cada hora se

actualiza el índice del punto seleccionado, y de esta manera siempre se toma un punto en la hora correcta.

La Figura 53 detalla el funcionamiento de la lectura de datos en modo test. El script de test usado en la validación del programa puede encontrarse tanto en el documento anexo como en el repositorio online de [GitHub](#) del proyecto.

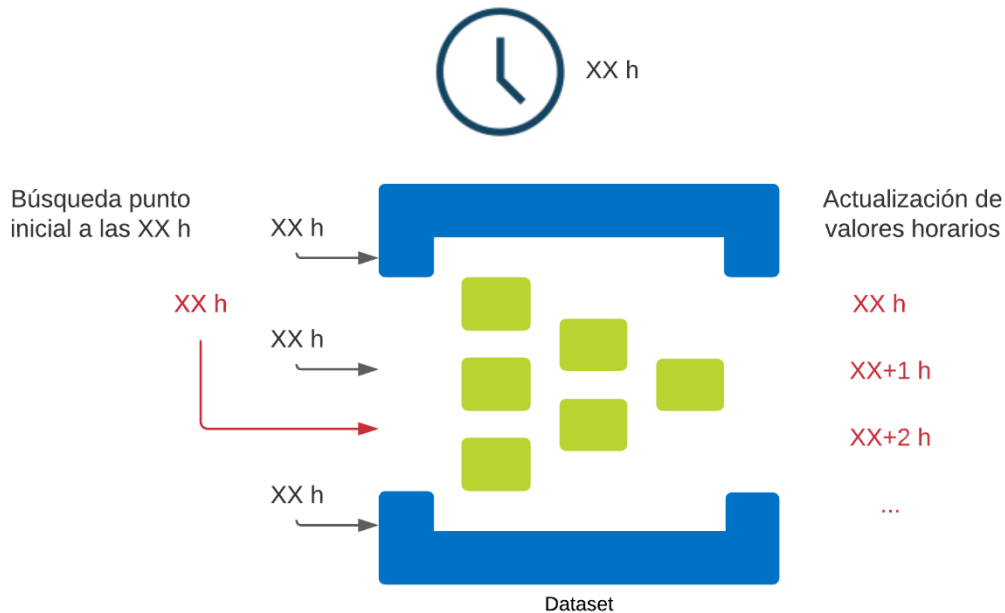


Figura 53. Funcionamiento de `adapt_data_test`

## 7.6 Resultados

Se han recogido los resultados de la ejecución del programa en un archivo csv que contiene los valores de predicción de consumo, predicción de la generación y la decisión sobre cargar o no la batería para cada momento en el que se realiza una decisión.

Si se representan en una gráfica puede comprobarse cómo el resultado es el esperado. Para que la gráfica de los resultados sea más fácil de interpretar, se ha eliminado el coeficiente de seguridad de la comparación entre el consumo y la generación. En la Figura 54 aparece el resultado del algoritmo aplicado en las horas centrales del día. No aparece el funcionamiento nocturno ya que al ser hora valle se encuentra siempre cargando y además no hay producción fotovoltaica. Se aprecia que, partiendo de un punto de menor generación, a las 9 am, la potencia fotovoltaica comienza a subir, y cuando supera la predicción del consumo, se extingue la orden de cargar la batería. Nuevamente, cuando el consumo vuelve a superar la producción, se vuelve a ordenar la carga de la batería.

Este es un pequeño ejemplo del comportamiento de la salida del algoritmo. La variable binaria de carga de la batería podría comunicarse con cualquier tipo de sistema para seguir desarrollando el producto.

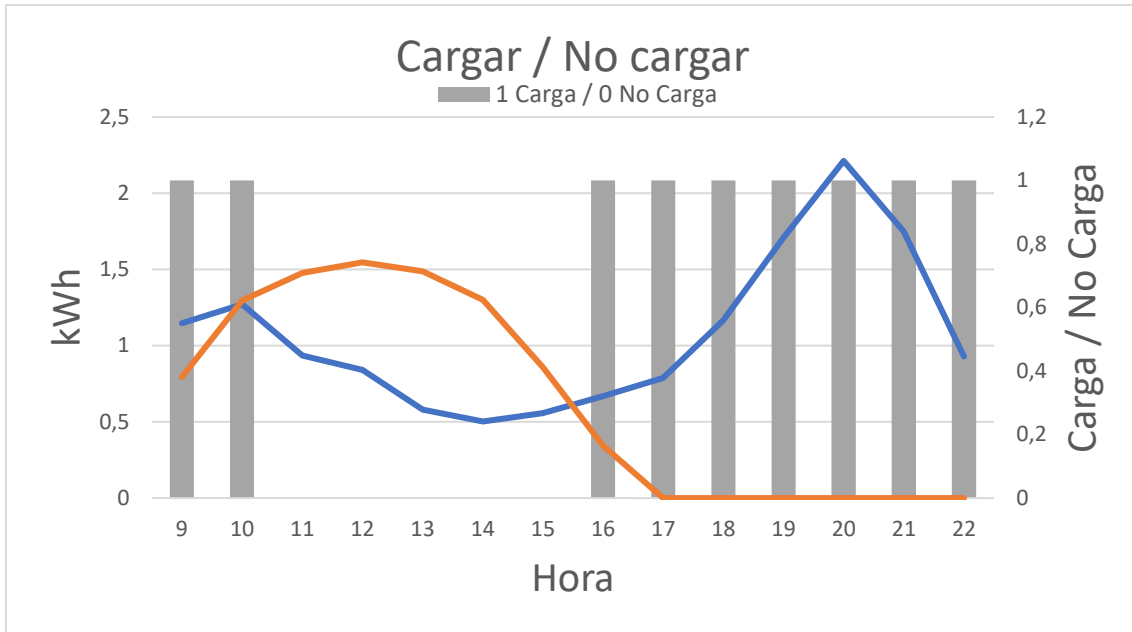


Figura 54. Decisión de cargar o no la batería (1 o 0) frente a consumo y generación fotovoltaica

Restaría probar el funcionamiento con una batería real, ya que por software es complicado modelar el comportamiento de una batería y no entra dentro del alcance de este TFM. A modo de conclusión, se recuerda que todos los parámetros relacionados con la batería son modificables por software, lo que facilita la posterior integración de este software en un producto más completo.

El valor de este producto radica en disponer de los datos a futuro tanto de la predicción como de la generación. Compararlos y definir en base a la diferencia si cargar o no la batería es solo un ejemplo de lo que se puede hacer con esta tecnología. Como línea futura, mayor investigación y experimentación con baterías reales podrían mejorar el algoritmo aplicando reglas de decisión más complejas.

Con esto se comprueba que tanto la predicción del consumo como el uso de la API de generación funcionan correctamente, así como el algoritmo que realiza la comparación de forma periódica.

## 8 CONCLUSIÓN Y LÍNEAS FUTURAS

Para finalizar el proyecto, se pueden extraer varias conclusiones del trabajo realizado:

En primer lugar, la inteligencia artificial ha llegado para quedarse. Las posibilidades que ofrece, no solo en el campo de la electrónica de potencia, sino en todos los ámbitos, son casi infinitas. El creciente interés por los datos y la tendencia a medir todo de los últimos años ha favorecido enormemente a su desarrollo. Si bien en este trabajo las precisiones alcanzadas por el modelo desarrollado distan mucho de ser perfectas, quiero remarcar que esta ha sido mi primera toma de contacto con esta tecnología y todo el desarrollo se ha apoyado en herramientas open source. Existen soluciones comerciales, como la *cloud* de Amazon, Google o IBM, que facilitan todo el proceso, desde el inicio de la minería de datos hasta el lanzamiento del modelo de IA a producción, incluso sin escribir líneas de código. Se podría haber estimado su uso para este proyecto, seguramente con mejores resultados, pero la finalidad de este trabajo era por un lado aprender de esta tecnología (y qué mejor que desarrollando todo el producto en código) y por otro lado visibilizar el potencial de la inteligencia artificial, tanto en los sectores tradicionales que todavía deben adaptar su uso como en la universidad.

En segundo lugar, se ha conseguido desarrollar con éxito todo el programa usando un mismo lenguaje: Python. Después de programar durante años en lenguajes de bajo nivel como C, se puede concluir que, si la velocidad del programa no es una de las prioridades, se puede entender por qué Python es uno de los lenguajes de programación más usados en la actualidad y uno de los más demandados profesionalmente. La cantidad de librerías open source de las que dispone permiten desarrollar cualquier tipo de proyecto, por no hablar del potencial que tiene para desarrollo de aplicaciones de IA y análisis de datos.

En tercer lugar, se ha dejado ver el potencial de la metodología *Data-driven decisión-making* que usan importantes compañías de software. En lugar de emplear metodologías genéricas, las mejores decisiones se toman en base a los hechos o datos de cada problema particular. De esta manera, como ejemplo en este proyecto, se ha podido dimensionar tanto el array fotovoltaico como la capacidad de las baterías, todo en base a los datos de consumo y generación anteriores. Esto permite adaptar el problema a cada caso particular y asegurar que las decisiones tomadas tienen fundamento en la realidad.

Como líneas futuras, me gustaría dejar planteada la posibilidad de desarrollar el producto completo. Si bien no se ha empleado un hardware específico para realizar las pruebas, el haber desarrollado todo el código en Python permite implementar el producto en casi cualquier tipo de máquina. Se puede añadir valor al producto aportando funcionalidades que emplean muchos softwares comerciales similares, como una interfaz gráfica con analíticas de los datos o la posibilidad de reentrenar el modelo con los datos nuevos entrantes cada cierto tiempo. Esta última posibilidad, que es tediosa de preparar con las herramientas usadas en este TFM, es de muy sencilla aplicación con las herramientas *cloud* de terceros descritas anteriormente.

También se ha pensado en el posible modelo comercial del producto. Si bien todo el TFM se ha desarrollado para ejecutarse en un pequeño ordenador como la mencionada BeagleBone o una Raspberry Pi, otra posibilidad además de trabajar de forma completamente local, sería

desplegar un contenedor alojado en un servidor web accesible desde cualquier sitio, que se encargase de realizar todos los cálculos necesarios. Así, a través de cualquier dispositivo se podría acceder al producto, que sería capaz de gestionar las baterías de cualquier usuario a través de internet. Esto habilitaría la posibilidad de vender el producto como suscripción a un servicio web a cualquier usuario del mundo.

A título personal, me gustaría dejar constancia de la dificultad que ha supuesto este TFM al provenir de una titulación como Ingeniería Industrial. Todo este proceso ha requerido de un alto grado de iniciativa a la hora de aprender tecnologías nuevas, las cuales no se han siquiera mencionado en ninguna asignatura de la titulación y que creo que actualmente son una necesidad de mercado. Los ingenieros estamos para resolver problemas, y por suerte, se nos enseña a desarrollar una alta capacidad de adaptación. En concreto, estoy agradecido por haber cursado esta titulación, a la que pone fin este TFM, por haberme dado las herramientas para desarrollarme en cualquier tipo de sector y la formación necesaria que sirve como base para aprender cualquier tipo de tecnología.

Finalmente, me gustaría devolverle a la comunidad *open source* todas las facilidades que ha supuesto este modelo de licencia que han permitido el desarrollo de este TFM, imposible de otra forma. Por ello, todo el trabajo puede consultarse de forma abierta en el repositorio Riunet de la UPV. Por su parte, el código es accesible en el repositorio GitHub del proyecto, en <https://github.com/RikiSot/smart-battery-manager>. Todo el código es *open source*, por lo que cualquiera puede hacer uso de él o realizar mejoras y aportaciones al repositorio, con la posibilidad de mejorar el proyecto con el tiempo gracias a la colaboración entre usuarios.

## BIBLIOGRAFÍA

- [1] R. E. d. España. [En línea]. Available: <https://www.ree.es/>.
- [2] Tableau, «What is data-driven decision-making?,» [En línea]. Available: <https://www.tableau.com/learn/articles/data-driven-decision-making>.
- [3] SmartCitiesWorld, «www.smartcitiesworld.net,» 2017. [En línea]. Available: <https://www.smartcitiesworld.net/news/news/huawei-set-to-accelerate-the-smart-grid-1613>.
- [4] AccuWeather, «AccuWeather API reference,» [En línea]. Available: <https://developer.accuweather.com/apis>.
- [5] UCI Machine Learning, «kaggle.com,» [En línea]. Available: <https://www.kaggle.com/uciml/electric-power-consumption-data-set>.
- [6] Digi, «Digi XBee® 3 Zigbee® RF Module,» [En línea]. Available: <https://www.digi.com/resources/documentation/digidocs/90001539/>.
- [7] C. Liechti, «Pyserial Documentation,» [En línea]. Available: <https://pythonhosted.org/pyserial/>.
- [8] V. Advani, «What is Machine Learning? How Machine Learning Works and future of it?,» [En línea]. Available: <https://www.mygreatlearning.com/blog/what-is-machine-learning/>.
- [9] «bismart.com,» [En línea]. Available: <https://blog.bismart.com/es/diferencia-machine-learning-deep-learning>.
- [10] R. Holbrook y A. Cook, «Intro to Deep Learning,» [En línea]. Available: <https://www.kaggle.com/learn/intro-to-deep-learning>.
- [11] J. Brownlee, «How to Convert a Time Series to a Supervised Learning Problem in Python,» [En línea]. Available: <https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/>.
- [12] A. Bronshtein, «Train/Test Split and Cross Validation in Python,» [En línea]. Available: <https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6>.

- [13] J. Brownlee, «A Gentle Introduction to the Rectified Linear Unit (ReLU),» [En línea]. Available: <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>.
- [14] IBM, «IBM Cloud Learn Hub. What is Overfitting?,» [En línea]. Available: <https://www.ibm.com/cloud/learn/overfitting>.
- [15] European Commission, «PVGIS,» [En línea]. Available: <https://ec.europa.eu/jrc/en/pvgis>.
- [16] A. Grönholm, «Advanced Python Scheduler,» [En línea]. Available: <https://apscheduler.readthedocs.io/en/3.x/>.

# PRESUPUESTO



Este documento muestra el presupuesto estimado del proyecto, de manera que se proporcione información suficiente sobre la dotación económica necesaria para efectuar su realización.

## 1. CUADRO MANO DE OBRA

Código	Ud.	Denominación	Precio (€)
MO.ING	h	Ing. Industrial	30

## 2. CUADRO MATERIALES

Código	Ud.	Denominación	Precio (€)	Referencia
MT.BGB	u	Beagle Bone	56,34	<a href="#">rs-online</a>
MT.XBEE	u	Kit Digi Xbee	88,36	<a href="#">mouser</a>

## 3. CUADRO MAQUINARIA Y SOFTWARE

Código	Ud.	Denominación	Precio anual (€)	Precio (€)
M.ORD	h	Ordenador Portátil	750	0,43
M.MSOFF	h	Microsoft Office	69	0,04
M.API	año	AccuWeather API	25	25

Para determinar el precio por hora del ordenador, de 1500€, se ha considerado un periodo de amortización de 2 años. Estimando que en un año se trabajan 11 meses, con aproximadamente 20 días de trabajo al mes con jornadas de 8 horas, se obtienen 1760 h/año.

Para calcular el precio del software utilizado se ha empleado la misma aproximación de horas al año, considerando el precio de la suscripción anual. El precio de la suscripción de la API de AccuWeather se obtiene directamente de su web.

## 4. CUADRO PRECIOS DESCOMPUESTOS

### Capítulo 1: Etapa de diseño

#### UO1.1 **Análisis de los datos y programación del código**

Descripción: Diseño, programación y redacción de documentos realizados para definir el producto

Código	Ud.	Denominación	Precio	Rendimiento	Importe
MO.GITI	h	Ing. Industrial	30,00 €	300	9.000,00 €
M.ORD	h	Ordenador Portátil	0,43 €	300	129,00 €
M.MSOFF	h	Microsoft Office	0,04 €	150	6,00 €
M.API	año	AccuWeather API	25,00 €	2	50,00 €
CDC	%	Costes directos complementarios	91,85 €	3	275,55 €
Total unidad de obra					9.460,55 €

### Capítulo 2: Construcción del prototipo

#### UO2.1 **Implantación**

Descripción: Carga del código y verificación del buen funcionamiento

Código	Ud.	Denominación	Precio	Rendimiento	Importe
MT.XBEE	u	Digi Xbee3 Mesh Kit	88,36 €	1	88,36 €
MT.BGB	u	Beagle Bone Black	56,34 €	1	56,34 €
MO.GITI	h	Ing. Industrial	30,00 €	50	1.500,00 €
M.ORD	h	Ordenador Portátil	0,43 €	50	21,50 €
CDC	%	Costes directos complementarios	16,66 €	3	49,99 €
Total unidad de obra					1.716,19 €

## 5. PRESUPUESTO TOTAL

Código	Ud.	Denominación	Medición	Precio	Importe
UO1.1	u	Programación del algoritmo	1	9.460,55 €	9.460,55 €
UO2.1	u	Implantación del código	1	1.716,19 €	1.716,19 €
<b>Presupuesto de ejecución material</b>					<b>11.176,74 €</b>
Gastos generales 12%					1.341,21 €
Beneficio industrial 6%					670,60 €
<b>Presupuesto de ejecución por contrata</b>					<b>13.188,55 €</b>
I.V.A 21%					2.769,60 €
<b>Presupuesto base de licitación</b>					<b>15.958,14 €</b>

Asciende el presente presupuesto a la expresada cantidad de:

**QUINCE MIL NOVECIENTOS CINCUENTA Y OCHO CON CATORCE CÉNTIMOS**

# ANEXO

El presente documento contiene todo el código que se ha empleado en la realización de este proyecto.

Si bien se puede consultar la totalidad del código aquí, se recomienda hacerlo en el repositorio web del proyecto, alojado en <https://github.com/RikiSot/smart-battery-manager> en la popular red social de desarrolladores GitHub. GitHub dispone de herramientas y formato adaptado para consultar código, tarea que puede hacerse algo tediosa a través de un documento escrito.



## Smart Battery Manager

Data driven decisions and deep learning in order  
to improve energy management and battery life

### Jupyter Notebooks

Este apartado contiene el código empleado para visualizar los datos, analizarlos y crear el modelo de inteligencia artificial. También incluye el análisis del tamaño de la batería y los paneles.

#### [build\\_model.ipynb](#)

#### LSTM PREDICCIÓN DE CONSUMOS

Import de las librerías necesarias

```
import itertools
import pickle

import keras
import matplotlib.pyplot as plt
import numpy as np # Linear algebra
import pandas as pd
import seaborn as sns
import sweetviz as sv
from keras.callbacks import EarlyStopping
from keras.layers import Dense
```

```

from keras.layers import Dropout
from keras.layers import LSTM
from keras.models import Sequential
from keras.utils import np_utils
from sklearn import metrics # for the check the error and accuracy of
    the model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split # to split the d
    ata into two parts
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler # for normalization
  
```

Análisis Exploratorio de Datos

*#Lectura del archivo*

```

df = pd.read_csv('household_power_consumption.txt', sep=';',
                parse_dates={'Fecha' : ['Date', 'Time']},
                infer_datetime_format=True,
                low_memory=False,
                na_values=['nan','?'],
                index_col='Fecha')
  
```

*#Tratamiento de NaN*

```

df = df.interpolate()
df.isnull().sum()
  
```

```

Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
  
```

*#Eliminar voltaje y energía reactiva*

```

df = df.drop(['Voltage', 'Global_reactive_power'], axis=1)
  
```

*#Cambio de unidades a kWh*

```

df['Sub_metering_1'] = df['Sub_metering_1']/1000
df['Sub_metering_2'] = df['Sub_metering_2']/1000
df['Sub_metering_3'] = df['Sub_metering_3']/1000
  
```

*#Cambio de potencia a energía (kWh)*

```

df['Energía activa global'] = df['Global_active_power']/60
#df['Energía reactiva global'] = df['Global_reactive_power']/60
  
```

*#Medidor 4*

```

df['Sub_metering_4'] = df['Energía activa global']-(df['Sub_metering_1
'] + df['Sub_metering_2'] + df['Sub_metering_3'])
  
```

*#Creación del nuevo dataframe*

```

df_mean = df[['Global_intensity']].copy()
df_sum = df.drop(['Global_intensity', 'Global_active_power'], axis=1)
  
```

```
#Resampling del nuevo dataframe
```

```
df_mean = df_mean.resample('h').mean()
```

```
df_sum = df_sum.resample('h').sum()
```

```
df1 = df_mean.merge(df_sum, left_index=True, right_index=True)
```

```
#Mover energia activa a la ultima posicion
```

```
brb = df1.pop('Energia activa global') # remove column b and store it in brb
```

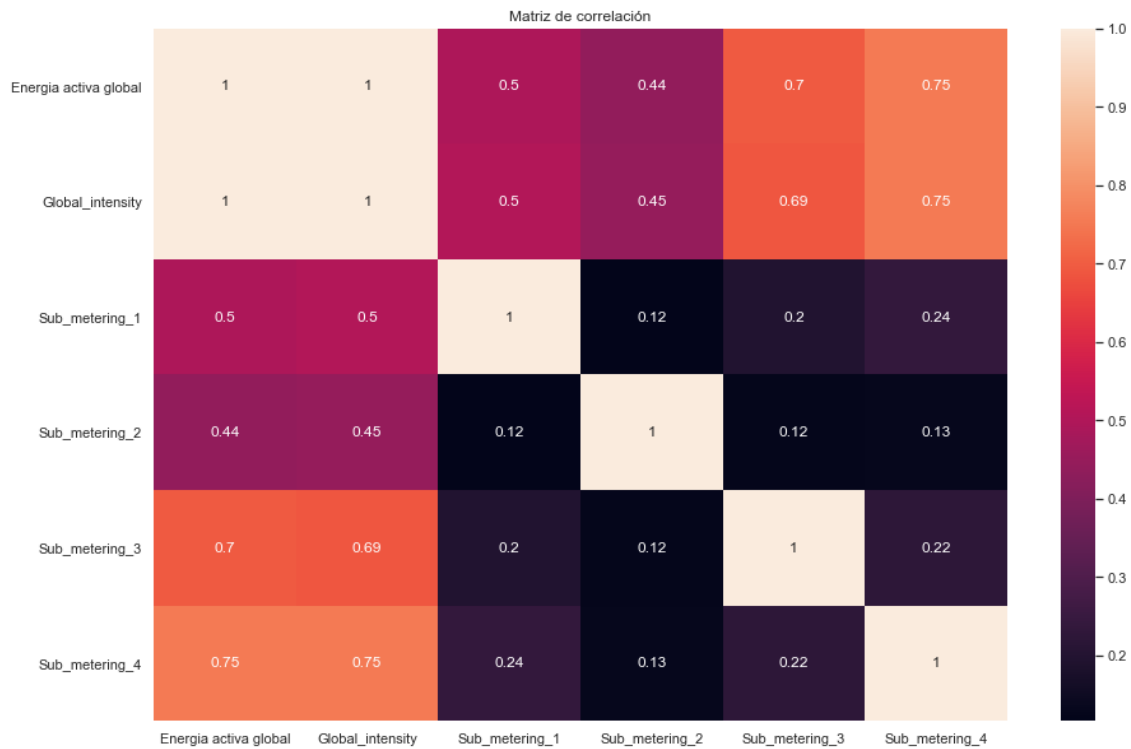
```
df1.insert(0, 'Energia activa global', brb) # en primera posicion
```

```
correlation_matrix = df1.corr()
```

```
plt.figure(figsize = (15,10))
```

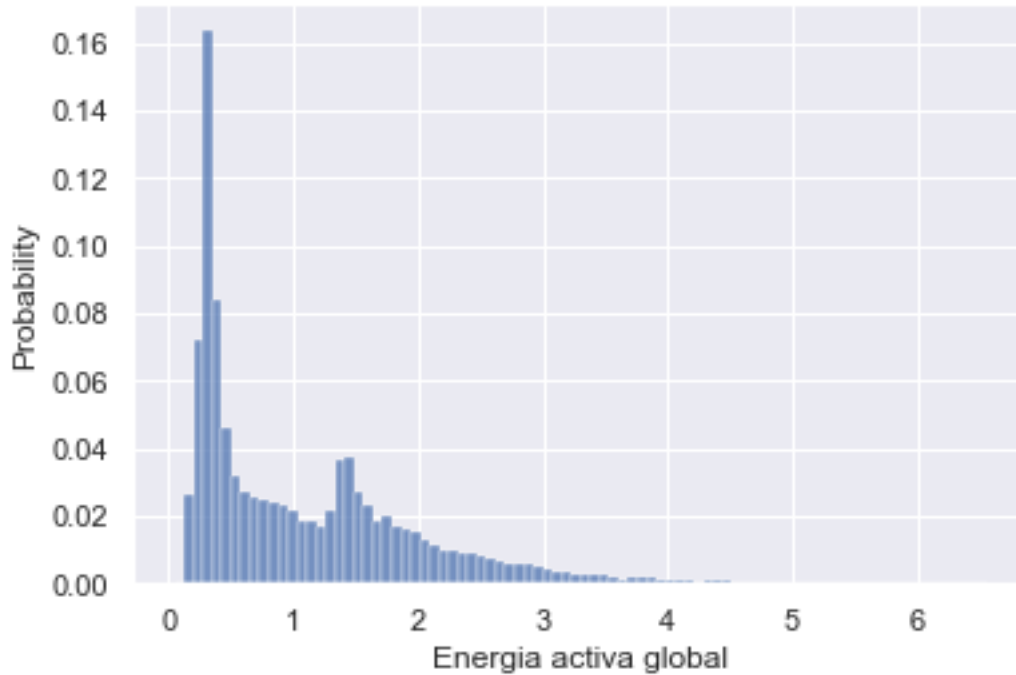
```
plt.title('Matriz de correlación')
```

```
sns.heatmap(data=correlation_matrix, annot=True)
```



```
sns.set_theme(style='darkgrid')
```

```
sns.histplot(df1['Energia activa global'], stat='probability')
```



## Construcción del modelo

```
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    dff = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(dff.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(dff.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_var
s)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Escalar los datos

```
n_hours = 24
n_features = df1.shape[1]
```

```
values = df1.values
```



```

scaler = MinMaxScaler()
scaler = scaler.fit(values)
scaled = scaler.transform(values)

#Save the scaler
pickle.dump(scaler, open('scaler.pkl','wb'))

supervised = series_to_supervised(scaled, n_hours, 1)

#Eliminamos variables que no vayan a predecirse
variables_eliminadas = supervised.columns[-(n_features-1):]
supervised.drop(variables_eliminadas, axis=1, inplace=True)

Train test split

# Dividir en entrenamiento y test
values = supervised.values

n_train_time = int(supervised.shape[0]*0.75)
train = values[:n_train_time, :]
test = values[n_train_time:, :]

# Dividir en entradas y salidas
X_train, y_train = train[:, :-1], train[:, -1]
X_test, y_test = test[:, :-1], test[:, -1]

# Redimensionar el input a 3d [muestras, secuencias, variables]
X_train = X_train.reshape((X_train.shape[0], n_hours, n_features))
X_test = X_test.reshape((X_test.shape[0], n_hours, n_features))
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

(25923, 24, 6) (25923,) (8642, 24, 6) (8642,)

Definir el modelo

early_stopping = EarlyStopping(
    min_delta=0.0005, # minimum amount of change to count as an impro
vement
    patience=10, # how many epochs to wait before stopping
    restore_best_weights=True,
)

#Preparar La red LSTM
units = 100
#n_outputs = y_train.shape[1]

model = Sequential()
model.add(LSTM(units, input_shape=(X_train.shape[1], X_train.shape
[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units, input_shape=(X_train.shape[1], X_train.shape
[2]), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
    
```

Adaptar el modelo a los datos

```
#Elegir hiperparametros
```

```
batch_size = 64
```

```
epochs = 50
```

```
# fit network
```

```
history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, y_test), verbose=1, shuffle=False, callbacks=[early_stopping])
```

```
# summarize history for loss
```

```
plt.plot(history.history['loss'])
```

```
plt.plot(history.history['val_loss'])
```

```
plt.title('model loss')
```

```
plt.ylabel('loss')
```

```
plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='upper right')
```

```
plt.show()
```

```
Epoch 1/50
```

```
406/406 [=====] - 20s 40ms/step - loss: 0.013
```

```
9 - val_loss: 0.0081
```

```
Epoch 2/50
```

```
406/406 [=====] - 15s 38ms/step - loss: 0.009
```

```
4 - val_loss: 0.0074
```

```
Epoch 3/50
```

```
406/406 [=====] - 16s 38ms/step - loss: 0.008
```

```
9 - val_loss: 0.0070
```

```
Epoch 4/50
```

```
406/406 [=====] - 16s 40ms/step - loss: 0.008
```

```
5 - val_loss: 0.0067
```

```
Epoch 5/50
```

```
406/406 [=====] - 15s 38ms/step - loss: 0.008
```

```
2 - val_loss: 0.0066
```

```
Epoch 6/50
```

```
406/406 [=====] - 15s 37ms/step - loss: 0.008
```

```
1 - val_loss: 0.0065
```

```
Epoch 7/50
```

```
406/406 [=====] - 16s 40ms/step - loss: 0.008
```

```
0 - val_loss: 0.0064
```

```
Epoch 8/50
```

```
406/406 [=====] - 16s 39ms/step - loss: 0.007
```

```
9 - val_loss: 0.0064
```

```
Epoch 9/50
```

```
406/406 [=====] - 15s 38ms/step - loss: 0.007
```

```
8 - val_loss: 0.0063
```

```
Epoch 10/50
```

```
406/406 [=====] - 16s 40ms/step - loss: 0.007
```

```
8 - val_loss: 0.0063
```

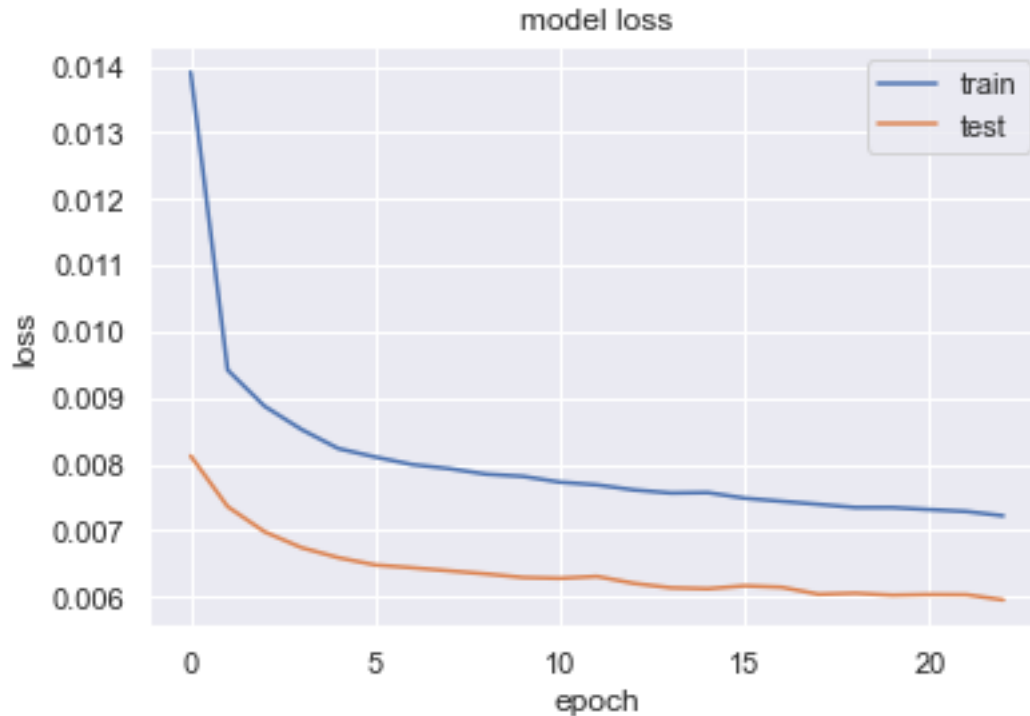
```
Epoch 11/50
```

```
406/406 [=====] - 16s 40ms/step - loss: 0.007
```

```
7 - val_loss: 0.0063
```

```
Epoch 12/50
```

```
406/406 [=====] - 16s 40ms/step - loss: 0.007
7 - val_loss: 0.0063
Epoch 13/50
406/406 [=====] - 16s 39ms/step - loss: 0.007
6 - val_loss: 0.0062
Epoch 14/50
406/406 [=====] - 15s 38ms/step - loss: 0.007
6 - val_loss: 0.0061
Epoch 15/50
406/406 [=====] - 15s 38ms/step - loss: 0.007
6 - val_loss: 0.0061
Epoch 16/50
406/406 [=====] - 16s 39ms/step - loss: 0.007
5 - val_loss: 0.0062
Epoch 17/50
406/406 [=====] - 16s 39ms/step - loss: 0.007
4 - val_loss: 0.0061
Epoch 18/50
406/406 [=====] - 15s 38ms/step - loss: 0.007
4 - val_loss: 0.0060
Epoch 19/50
406/406 [=====] - 16s 39ms/step - loss: 0.007
3 - val_loss: 0.0061
Epoch 20/50
406/406 [=====] - 16s 39ms/step - loss: 0.007
3 - val_loss: 0.0060
Epoch 21/50
406/406 [=====] - 16s 40ms/step - loss: 0.007
3 - val_loss: 0.0060
Epoch 22/50
406/406 [=====] - 16s 40ms/step - loss: 0.007
3 - val_loss: 0.0060
Epoch 23/50
406/406 [=====] - 16s 39ms/step - loss: 0.007
2 - val_loss: 0.0060
```



Resultados

```
# Hacer una predicción
yhat = model.predict(X_test)
X_test2 = X_test.reshape((X_test.shape[0], n_hours*n_features))
# Invertir el escalado de la predicción
inv_yhat = np.concatenate((yhat, X_test2[:, -(n_features-1):]), axis=
1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# Invertir el escalado del real
test_y = y_test.reshape((len(y_test), 1))
inv_y = np.concatenate((test_y, X_test2[:, -(n_features-1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calcular RMSE
rmse = np.sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

Test RMSE: 0.513

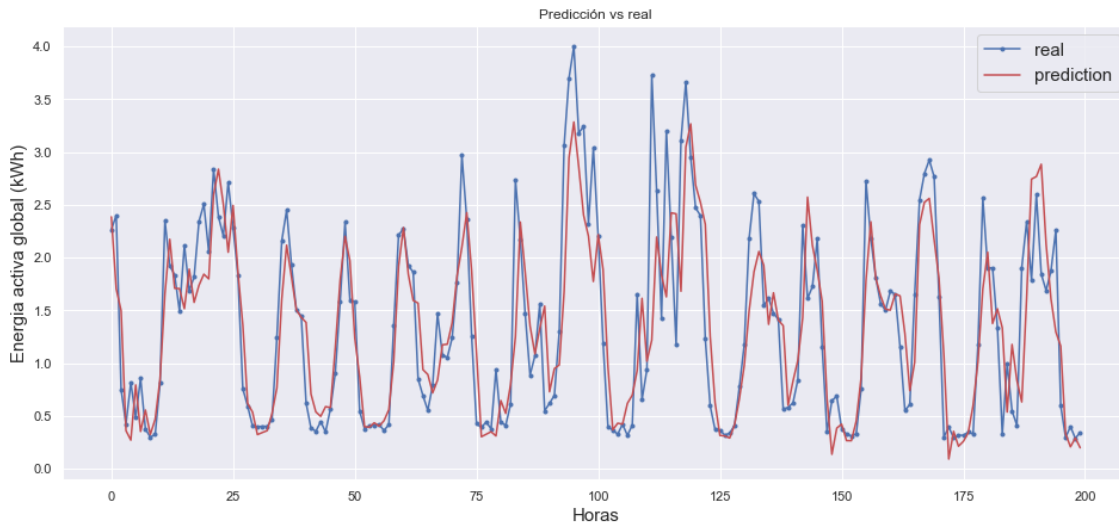
```
## time steps, every step is one hour (you can easily convert the time
step to the actual time index)
## for a demonstration purpose, I only compare the predictions in 200
hours.
```

```
hours = 200
```

```
plt.figure(figsize = (16,7))
plt.title('Predicción vs real')
```

```
aa=[x for x in range(hours)]
```

```
plt.plot(aa, inv_y[:hours], marker='.', label="real")
plt.plot(aa, inv_yhat[:hours], 'r', label="prediction")
plt.ylabel('Energia activa global (kWh)', size=15)
plt.xlabel('Horas', size=15)
plt.legend(fontsize=15)
plt.show()
```



Funciones añadidas

```
# Guardar el Modelo
model.save('consumos_model.h5')

# Recrea exactamente el mismo modelo solo desde el archivo
model = keras.models.load_model('consumos_model.h5')
```

Iterar para encontrar los mejores hiperparámetros

```
#Hiperparametros
```

```
n_units = [50,100,150,250]
n_capas = [1,2]
n_batch_size = [32, 64]
epochs = 50
i=0
best = 10000
errores = []
parametros = []
```

```
for capas in n_capas:
```

```
    for batch_size in n_batch_size:
```

```
        for units in n_units:
```

```
            # Dividir en entrenamiento y test
            values = supervised.values
```

```
            n_train_time = int(supervised.shape[0]*0.75)
```

```

train = values[:n_train_time, :]
test = values[n_train_time:, :]

# Dividir en entradas y salidas
X_train, y_train = train[:, :-1], train[:, -1]
X_test, y_test = test[:, :-1], test[:, -1]

# Redimensionar el input a 3d [muestras, secuencias, varia
bles]
X_train = X_train.reshape((X_train.shape[0], n_hours, n_fea
tures))
X_test = X_test.reshape((X_test.shape[0], n_hours, n_featu
res))

#Configurar modelo
if(capas==1):
    model = Sequential()
    model.add(LSTM(units, input_shape=(X_train.shape[1], X
_train.shape[2]), return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(loss='mse', optimizer='adam')
if(capas==2):
    model = Sequential()
    model.add(LSTM(units, input_shape=(X_train.shape[1], X
_train.shape[2]), return_sequences=True))
    model.add(Dropout(0.2))
    model.add(LSTM(units, input_shape=(X_train.shape[1], X
_train.shape[2]), return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(1))
    model.compile(loss='mse', optimizer='adam')

# fit network
history = model.fit(X_train, y_train, epochs=epochs, batch
_size=batch_size, validation_data=(X_test, y_test), verbose=1, shuffle
=False, callbacks=[early_stopping])

# Hacer una predicción
yhat = model.predict(X_test)
X_test2 = X_test.reshape((X_test.shape[0], n_hours*n_featu
res))

# Invertir el escalado de la predicción
inv_yhat = np.concatenate((yhat, X_test2[:, -(n_features-
1):]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# Invertir el escalado del real
test_y = y_test.reshape((len(y_test), 1))
inv_y = np.concatenate((test_y, X_test2[:, -(n_features-
1):]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
    
```

```

inv_y = inv_y[:,0]
# calcular RMSE
rmse = np.sqrt(mean_squared_error(inv_y, inv_yhat))
print('-----')
print('Iteración número ', i)
print('Parametros:')
print('Capas =', capas)
print('Units =', units)
print('Batch Size =', batch_size)
print('Test RMSE: %.3f' % rmse)
print('-----')
print()
current_parametros = {
    'iteracion': i,
    'units': units,
    'batch_size': batch_size,
    'capas': capas,
    'error': rmse
}
parametros.append(current_parametros)
print('-----')
errores.append(rmse)
if(rmse < best):
    best_model = model
    best = rmse
    best_parametros = current_parametros
i+=1

results=pd.DataFrame(parametros)
results.to_excel('errores.xlsx')
results.loc[results['error'].idxmin()]
# Guardar el Modelo
best_model.save('best_model.h5')

```

```

iteracion      9.000000
units          100.000000
batch_size     32.000000
capas          2.000000
error          0.507428
Name: 9, dtype: float64

```

### baterías y panel.ipynb

## Estudio de baterías y paneles

Import de las librerías necesarias

```
import matplotlib.pyplot as plt
import numpy as np # Linear algebra
import pandas as pd
import seaborn as sns
```

Datos de consumo

*#Lectura del archivo*

```
df = pd.read_csv('household_power_consumption.txt', sep=';',
                parse_dates={'Fecha' : ['Date', 'Time']},
                infer_datetime_format=True,
                low_memory=False,
                na_values=['nan', '?'],
                index_col='Fecha')
```

```
df=df['Global_active_power'].resample('h').mean()
```

```
consumo_horas_dia = df.groupby(df.index.hour).mean()
```

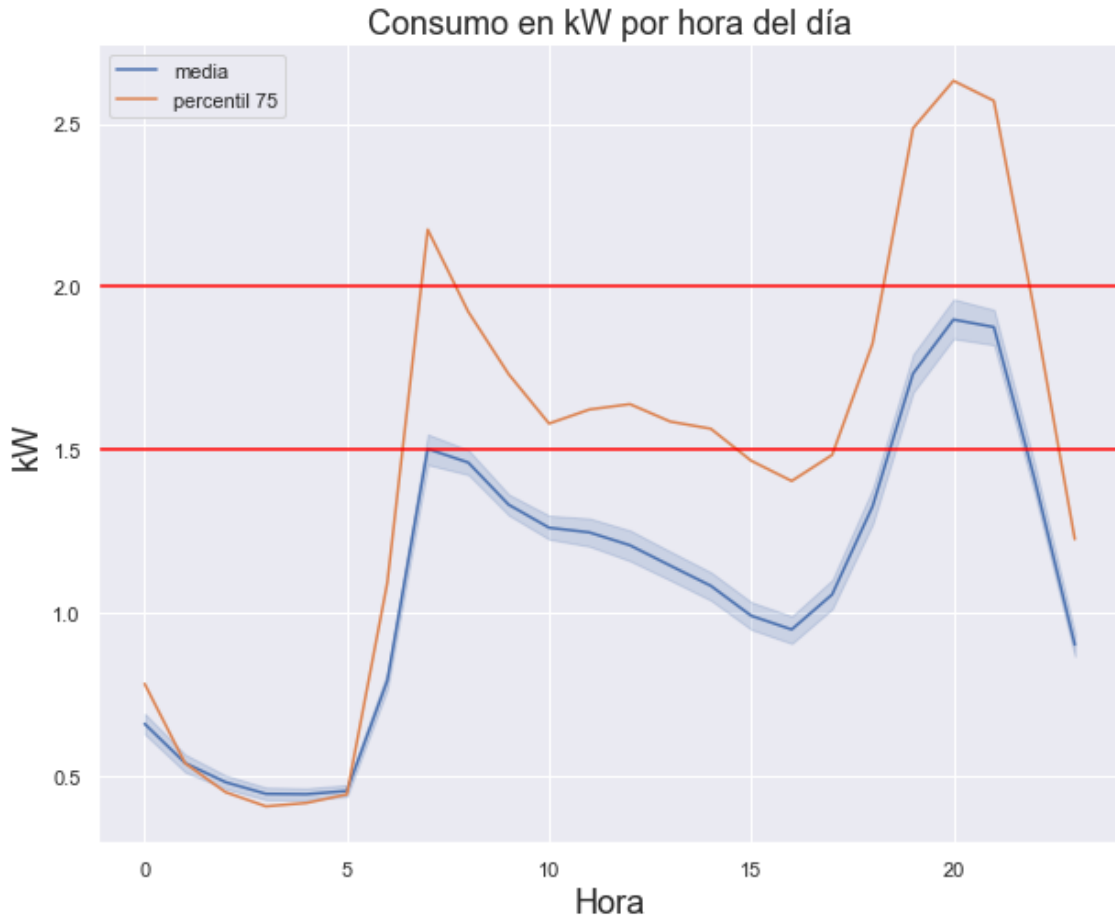
```
hours=df.index.hour
```

```
consumo_plot=pd.concat([df, pd.DataFrame(hours, index=df.index)], axis
                        = 1)
```

```
plt.figure(figsize = (10,8))
sns.set_theme(style='darkgrid')
plt.title('Consumo en kW por hora del día', fontsize=18)
plt.xlabel('Hora', fontsize=18)
plt.ylabel('kW', fontsize=18)
```

```
graph = sns.lineplot(x='Fecha', y='Global_active_power', data=consumo_
plot, label='media')
graph = sns.lineplot(data=df.groupby(df.index.hour).quantile(0.75), la
bel='percentil 75')
graph.axhline(2, color='red')
graph.axhline(1.5, color='red')
```





## Datos de generación

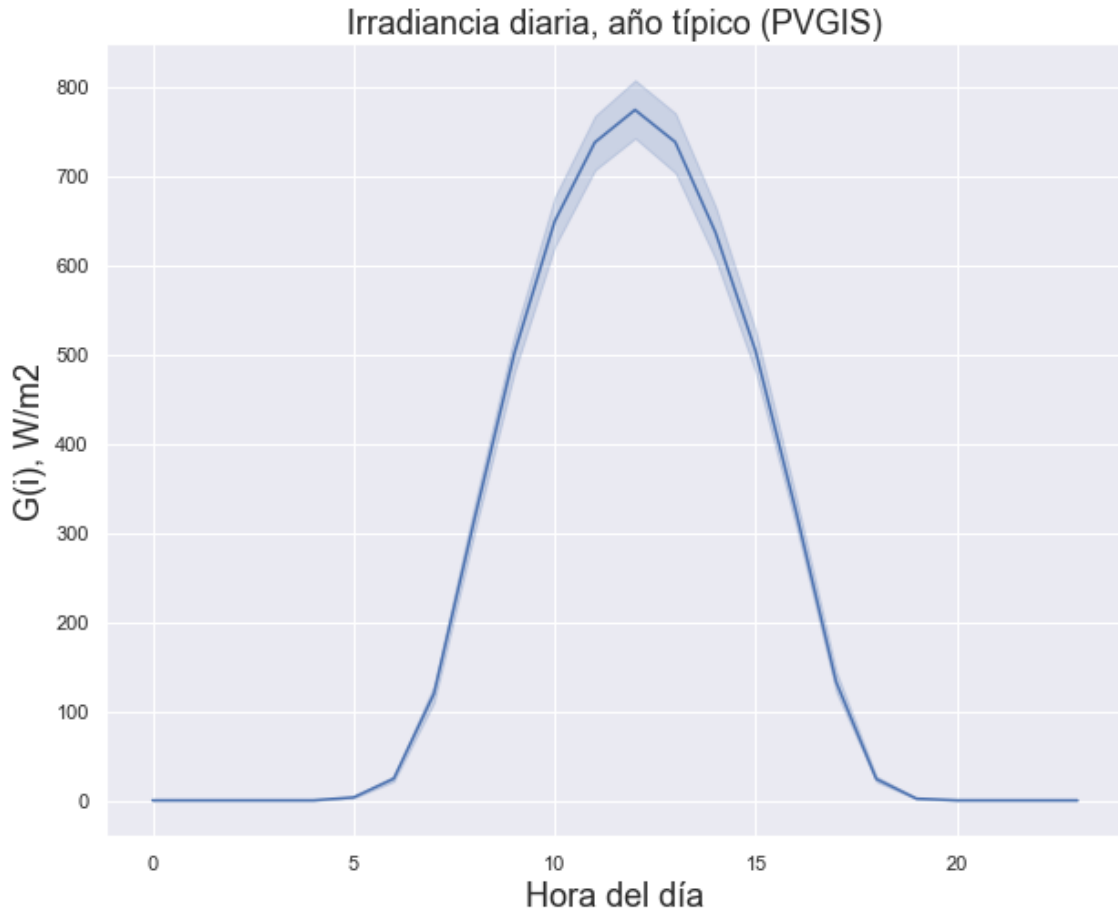
*#Lectura del archivo*

```
rad = pd.read_csv('radiationdata.csv', sep=',',
                 parse_dates={'Fecha' : ['time']},
                 infer_datetime_format=True,
                 low_memory=False,
                 na_values=['nan', '?'],
                 index_col='Fecha')

string = '%Y%m%d:%H%M'
rad.index = pd.to_datetime(rad.index, format=string)
rad['hour'] = rad.index.hour

rad=rad[['G(i)', 'hour']]

plt.figure(figsize = (10,8))
sns.set_theme(style='darkgrid')
plt.title('Irradiancia diaria, año típico (PVGIS)', fontsize=18)
plt.xlabel('Hora del día', fontsize=18)
plt.ylabel('G(i), W/m2', fontsize=18)
sns.lineplot(x='hour', y='G(i)', data=rad)
```



Datos de potencia generada

```
rendimiento = 0.15
superficie = 17
```

```
rad['potencia'] = (rad['G(i)'] * rendimiento * superficie)/1000
```

```
rad.describe()
```

```
G(i)
```

```
hour
```

```
potencia
```

```
count
```

```
8784.000000
```

```
8784.000000
```

```
8784.000000
```

```
mean
```

```
228.423913
```

```
11.500000
```

0.582481

std

331.098905

6.922581

0.844302

min

0.000000

0.000000

0.000000

25%

0.000000

5.750000

0.000000

50%

0.000000

11.500000

0.000000

75%

432.872500

17.250000

1.103825

max

1436.630000

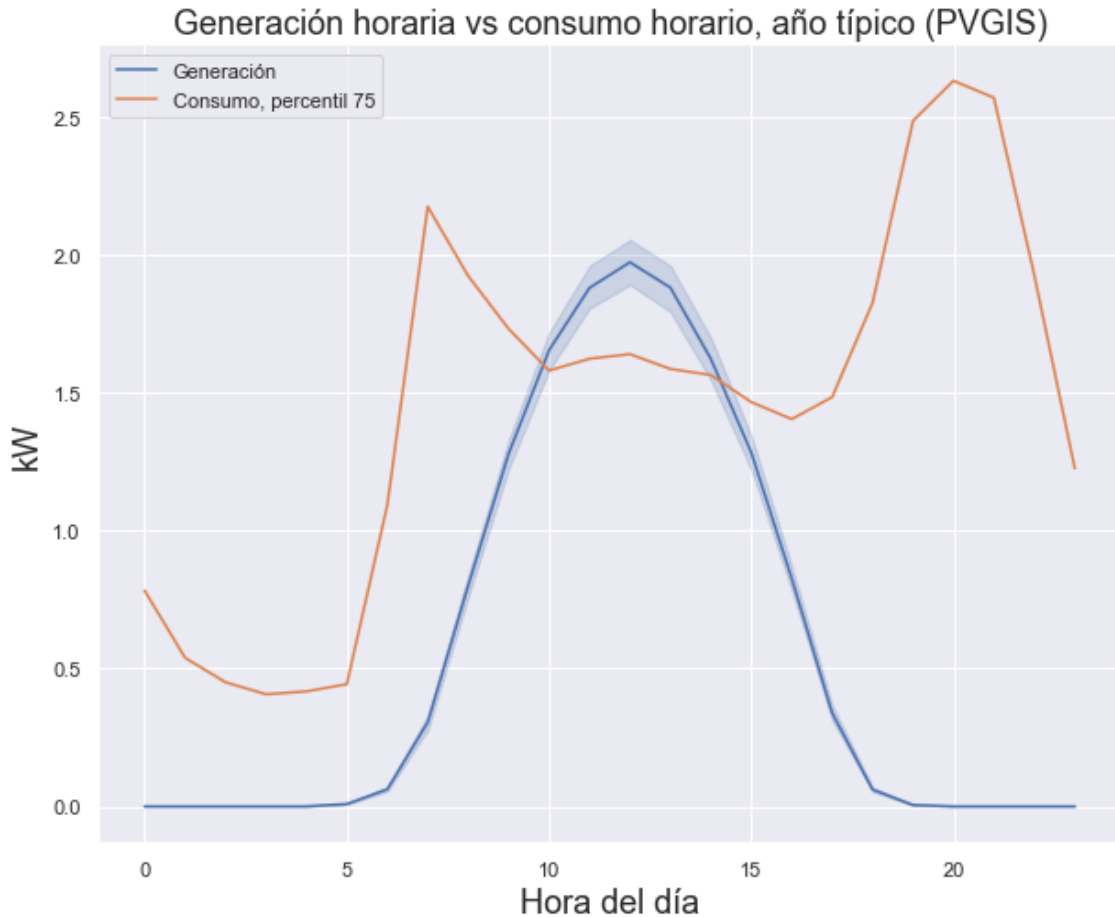
23.000000

3.663407

## Generación vs consumo

```
plt.figure(figsize = (10,8))
sns.set_theme(style='darkgrid')
plt.title('Generación horaria vs consumo horario, año típico (PVGIS)',
          fontsize=18)
plt.xlabel('Hora del día', fontsize=18)
plt.ylabel('kW', fontsize=18)
sns.lineplot(x='hour', y='potencia', data=rad, label='Generación')
```

```
#graph = sns.Lineplot(x='Fecha', y='Global_active_power', data=consumo_plot, Label='media')
graph = sns.lineplot(data=df.groupby(df.index.hour).quantile(0.75), label='Consumo, percentil 75')
```



Baterías

```
total_consumo_dia = df.groupby(df.index.hour).mean()

potencia_diaria = rad['potencia'].groupby(rad.index.hour).mean()
consumo_diario = df.groupby(df.index.hour).quantile(0.75)

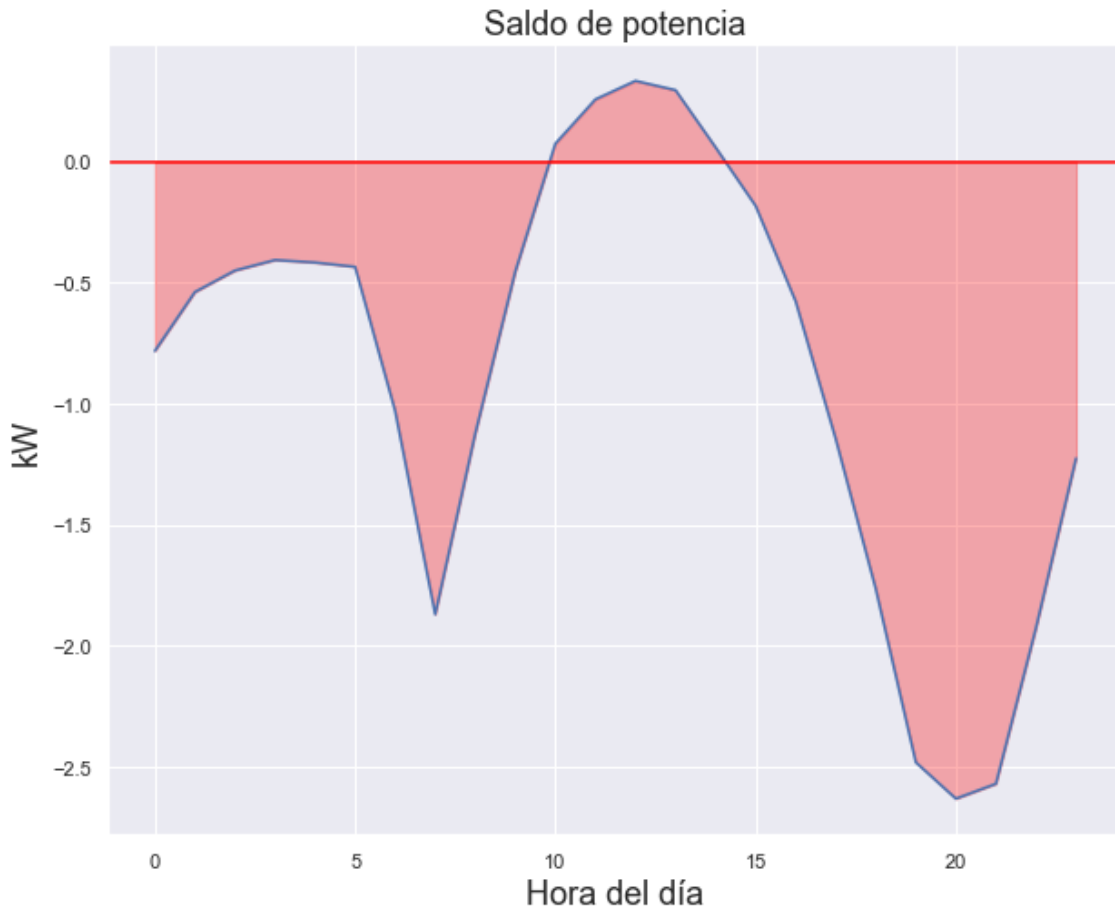
diff = potencia_diaria - consumo_diario

plt.figure(figsize = (10,8))
sns.set_theme(style='darkgrid')

plt.title('Saldo de potencia', fontsize=18)
plt.xlabel('Hora del día', fontsize=18)
plt.ylabel('kW', fontsize=18)
ax = sns.lineplot(data=diff)
l1 = ax.lines[0]

x1 = l1.get_xydata()[:, 0]
y1 = l1.get_xydata()[:, 1]
```

```
ax.fill_between(x1, y1, color="red", alpha=0.3)
ax.axhline(0, color='red')
```



```
start_valle = 0
end_valle = 8
index = list(range(start_valle+1,end_valle+1))
```

```
day_diff = diff[diff.index.difference(index)]
day_diff
```

```
Fecha
0    -0.782217
9    -0.453693
10   0.074019
11   0.256911
12   0.333388
13   0.295071
14   0.058772
15  -0.182795
16  -0.577356
17  -1.145550
18  -1.765000
19  -2.481072
20  -2.631333
21  -2.569842
22  -1.925783
```

```
23 -1.225742
dtype: float64
```

```
reindex = day_diff.index[1:].to_list()
reindex.append(day_diff.index[0])
day_diff_re = day_diff.reindex(reindex)
day_diff_re.index.values[-1] = 24
day_diff_re
```

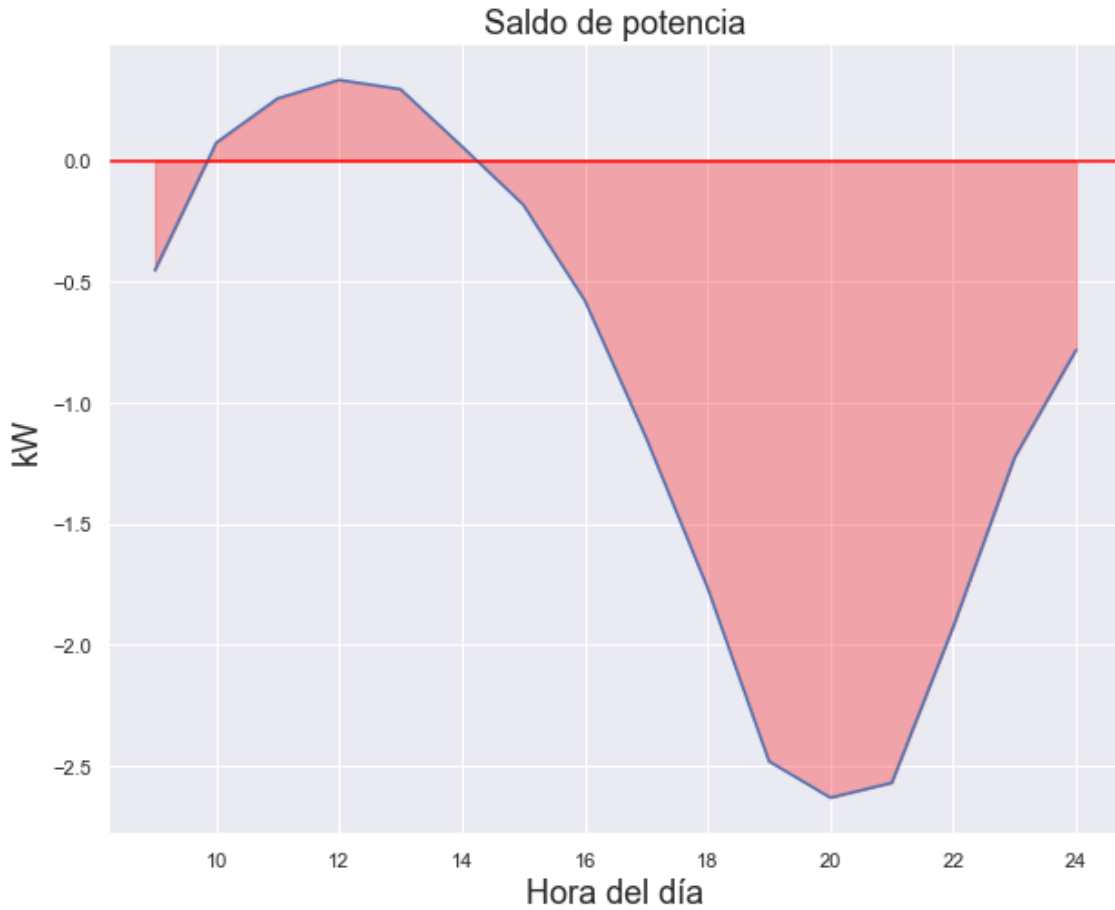
```
Fecha
9 -0.453693
10 0.074019
11 0.256911
12 0.333388
13 0.295071
14 0.058772
15 -0.182795
16 -0.577356
17 -1.145550
18 -1.765000
19 -2.481072
20 -2.631333
21 -2.569842
22 -1.925783
23 -1.225742
24 -0.782217
dtype: float64
```

```
plt.figure(figsize = (10,8))
sns.set_theme(style='darkgrid')
```

```
plt.title('Saldo de potencia', fontsize=18)
plt.xlabel('Hora del día', fontsize=18)
plt.ylabel('kW', fontsize=18)
ax = sns.lineplot(data=day_diff_re)
l1 = ax.lines[0]
```

```
x1 = l1.get_xydata()[:, 0]
y1 = l1.get_xydata()[:, 1]
```

```
ax.fill_between(x1, y1, color="red", alpha=0.3)
ax.axhline(0, color='red')
```



```
capacidad=day_diff_re.sum()
print(capacidad)
```

```
-14.722222377049185
```

```
day_diff_re
```

```
Fecha
```

```
9    -0.453693
10    0.074019
11    0.256911
12    0.333388
13    0.295071
14    0.058772
15   -0.182795
16   -0.577356
17   -1.145550
18   -1.765000
19   -2.481072
20   -2.631333
21   -2.569842
22   -1.925783
23   -1.225742
24   -0.782217
dtype: float64
```

```

energy = []
last_value = 0
values = []
for x in day_diff_re:
    value=x+last_value
    energy.append(value)
    last_value = value
print(energy[-1])
print(capacidad)

-14.722222377049183
-14.722222377049185

fig, axes = plt.subplots(2, figsize=(10, 12), sharey=False)
fig.suptitle('Energía acumulada en batería vs saldo energético', fonts
ize=22)

# Energía en batería
sns.lineplot(ax=axes[0],y=energy, x=day_diff_re.index, label='Energía'
)
axes[0].set_title('Energía acumulada en la batería', fontsize=18)
axes[0].set_xlabel('Hora del día', fontsize=18)
axes[0].set_ylabel('kWh', fontsize=18)
axes[0].axhline(0, color='red', ls='--', label='Capacidad batería 100%
')
axes[0].legend()

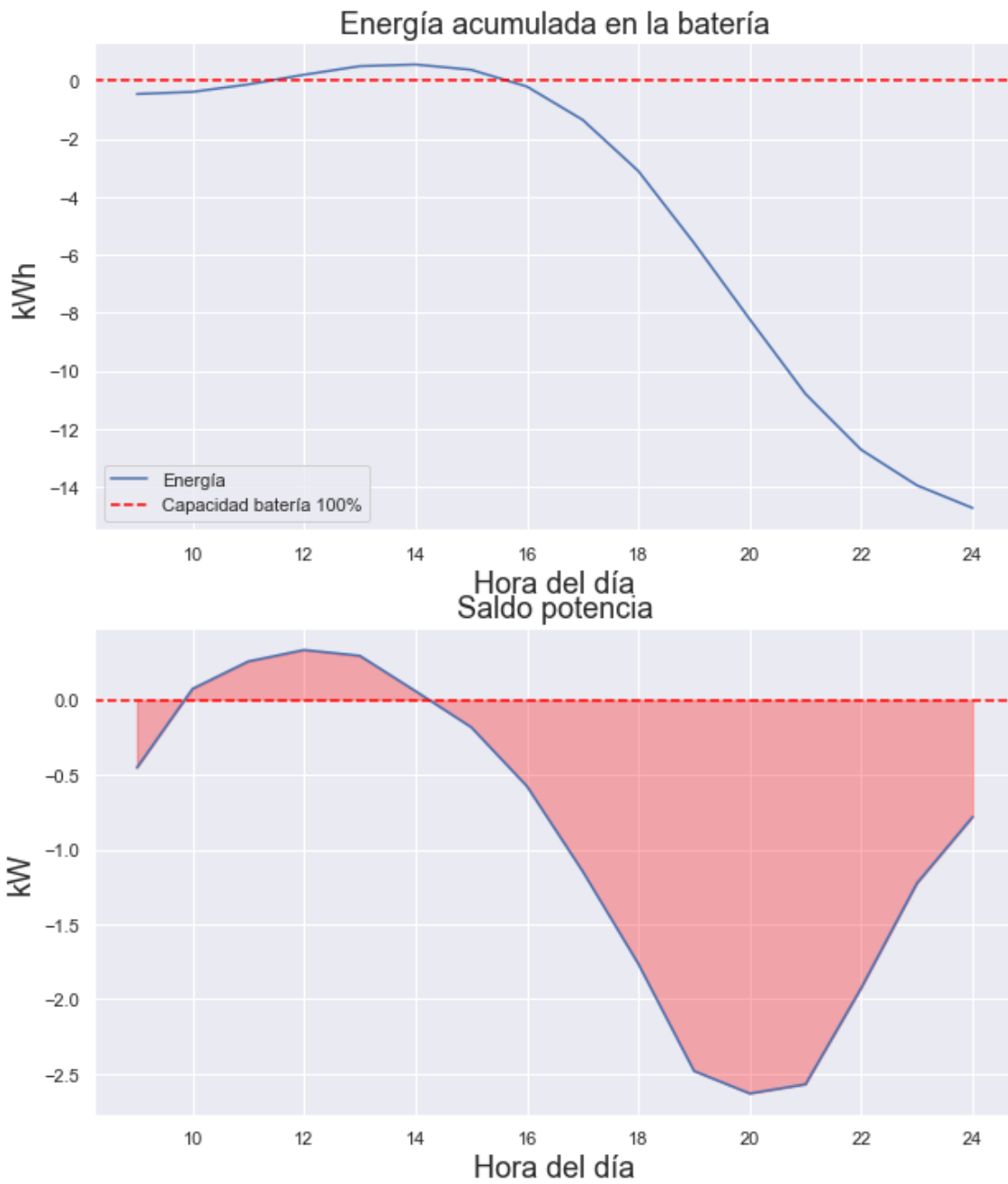
# Saldo de potencia
sns.lineplot(ax=axes[1], data=day_diff_re)
axes[1].set_title('Saldo potencia', fontsize=18)
axes[1].set_xlabel('Hora del día', fontsize=18)
axes[1].set_ylabel('kW', fontsize=18)

l1 = axes[1].lines[0]
x1 = l1.get_xydata()[:, 0]
y1 = l1.get_xydata()[:, 1]

axes[1].fill_between(x1, y1, color="red", alpha=0.3)
axes[1].axhline(0, color='red',ls='--')
    
```



### Energía acumulada en batería vs saldo energético



max(energy)

0.5644676038251333

## Python Scripts

### Librerías necesarias (requirements.txt)

```

1. pandas==1.2.4
2. APScheduler==3.8.1
3. keras==2.6.0
4. numpy==1.19.5
5. requests==2.25.1
6. seaborn==0.11.1
7. sweetviz==2.1.3
8. scikit-learn==0.24.2
9. matplotlib==3.4.2
    
```

### /BeagleBoneScripts

#### rw\_sensor\_data.py

```

1. import serial
2.
3.
4. def append_csv(filepath, byte_string):
5.     row=byte_string.decode('utf-8')
6.     with open(filepath, 'a', newline='\n') as f:
7.         f.write(row)
8.
9. serialPort = serial.Serial(
10.     port='COM3')
11.
12. eol = b'\r'
13. filepath = 'sensor_data.csv'
14.
15. while(1):
16.     with serialPort as ser:
17.         x = ser.read_until(eol)
18.         print(x)
19.         append_csv(filepath, x)
    
```

### /src

#### bateria.py

```

1. class Bateria():
2.
3.     def __init__(self):
4.         self.battery_level = 50
5.         self.max_level_night = 85
6.         self.cargar_bateria = 0
7.         self.max_level_day = 80
8.         self.min_level_day = 20
9.
10.     def charge_simulator(self):
    
```

```

11.     print('Charge simulator')
12.     if (self.battery_level > 0 and self.battery_level < 100):
13.         if (self.cargar_bateria == 1):
14.             self.battery_level += 1
15.         else:
16.             self.battery_level -= 1
17.
18.     def reset(self):
19.         self.battery_level = 50
20.
21.     def full_charge(self):
22.         self.battery_level = 100
23.
24.     def empty_charge(self):
25.         self.battery_level = 0
    
```

## functions.py

```

1. import requests
2. import pandas as pd
3. from datetime import datetime
4.
5.
6. def get_irradiance_next_hour():
7.     # API parametros
8.     baseurl = 'http://dataservice.accuweather.com/forecasts/v1/hourly/1hour/'
9.     location_key = '310683'
10.    apikey = 'xxxxxxxxxxxxxxxx'
11.
12.    parameters = {
13.        'apikey': apikey,
14.        'details': 'true'
15.    }
16.
17.    url = baseurl + location_key
18.    response = requests.get(url, params=parameters)
19.    try:
20.        result = response.json()[0]['SolarIrradiance']['Value']
21.    except:
22.        result = 0
23.    return result
24.
25.
26. def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
27.    n_vars = 1 if type(data) is list else data.shape[1]
28.    dff = pd.DataFrame(data)
29.    cols, names = list(), list()
30.    # input sequence (t-n, ... t-1)
31.    for i in range(n_in, 0, -1):
32.        cols.append(dff.shift(i))
33.        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
34.    # forecast sequence (t, t+1, ... t+n)
35.    for i in range(0, n_out):
36.        cols.append(dff.shift(-i))
37.        if i == 0:
38.            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
39.        else:
40.            names += [('var%d(t+%d)' % (j + 1, i)) for j in range(n_vars)]
41.    # put it all together
42.    agg = pd.concat(cols, axis=1)
43.    agg.columns = names
    
```

```

44.     # drop rows with NaN values
45.     if dropnan:
46.         agg.dropna(inplace=True)
47.     return agg
48.
49.
50. # Write to csv
51. def write_to_csv_RT(filename, names, data):
52.     print('writing onto csv file at: ', datetime.now())
53.     filepath = "../Data/" + filename
54.
55.     df_to_csv = pd.DataFrame(columns=names)
56.     ts = datetime.now()
57.
58.     new_row = pd.DataFrame([data], columns=names, index=[ts])
59.     df_to_csv = pd.concat([df_to_csv, pd.DataFrame(new_row)],
        ignore_index=False)
60.     df_to_csv.index.name = 'timeStamp'
61.     df_to_csv.to_csv(filepath, mode='a', header=False)
    
```

## smart\_algorithm.py

```

1. # Imports
2. import pickle
3. import random
4.
5. import keras
6. import numpy as np
7.
8.
9. from bateria import Bateria
10. from functions import *
11.
12.
13. class SmartAlgorithm():
14.
15.     def __init__(self):
16.         self.bateria = Bateria()
17.         # Load model
18.         self.model = keras.models.load_model('../Data/consumos_model.h5')
19.         # Load scaler
20.         self.scaler = pickle.load(open('../Data/scaler.pkl', 'rb'))
21.         # Ruta del csv
22.         self.filepath = '../Data/household_power_consumption.txt'
23.         # Numero de horas de la ventana temporal
24.         self.n_hours = 24
25.         # Superficie de paneles instalados en m2
26.         self.sup_paneles = 17
27.         # Rendimiento de los paneles en base 1
28.         self.rendimiento = 0.15
29.         # Coef seguridad en la comparacion
30.         self.x_security = 1.5
31.         # Next forecast irradiation
32.         self.irradiance = get_irradiance_next_hour()
33.
34.         # Posicion inicial de lectura en el df. Testing purposes
35.         self.is_df_randomized = 0
36.
37.     def is_valley(self):
38.         # Periodo valle. Indica que hay que cargar independientemente del resto
39.         fecha = datetime.today()
    
```

```

40.     hora = fecha.hour
41.     dia_semana = fecha.weekday()
42.
43.     if (hora >= 0 and hora < 8):
44.         return 1
45.     else:
46.         return 0
47.
48.     # Posibilidad de incluir sábados y domingos
49.
50.     # if ((hora >= 0 and hora < 8) or (dia_semana > 4 and dia_semana <= 6)):
51.     #     return 1
52.     # else:
53.     #     return 0
54.
55.     def charge_battery_night(self):
56.         # print('Checkeando si es valley a las: ',datetime.now())
57.         if (self.is_valley()):
58.             if (self.bateria.battery_level <= self.bateria.max_level_night):
59.                 # Cargar independientemente del consumo
60.                 self.bateria.cargar_bateria = 1
61.                 # Cargar hasta que el porcentaje de bateria llega al maximo
62.                 establecido
63.             else:
64.                 self.bateria.cargar_bateria = 0
65.                 write_to_csv_RT('./Data/charge_battery_data.csv',
66.                 ['charge_battery', 'consumo', 'generation'],
67.                 [self.bateria.cargar_bateria, 'NaN', 'NaN'])
68.
69.     def make_prediction(self, data):
70.         yhat = self.model.predict(data)
71.         # Invertir el escalado de la predicción
72.         try:
73.             inv_yhat = np.concatenate((yhat, self.scaled_last_row), axis=1)
74.             inv_yhat = self.scaler.inverse_transform(inv_yhat)
75.             inv_yhat = inv_yhat[:, 0]
76.             return inv_yhat[0]
77.         except:
78.             print('Called before adapt_data. No scaled data')
79.
80.     def load_data(self):
81.         # Lectura del archivo (prueba con el dataset)
82.         df = pd.read_csv(self.filepath, sep=';',
83.         parse_dates={'Fecha': ['Date', 'Time']},
84.         infer_datetime_format=True,
85.         low_memory=False,
86.         na_values=['nan', '?'],
87.         index_col='Fecha')
88.         return df
89.
90.     def adapt_data(self, df):
91.         # Tratamiento de NaN
92.         df = df.interpolate()
93.         df.isnull().sum()
94.
95.         # Eliminar voltaje y energia reactiva
96.         df = df.drop(['Voltage', 'Global_reactive_power'], axis=1)
97.
98.         # Cambio de unidades a KWh
99.         df['Sub_metering_1'] = df['Sub_metering_1'] / 1000
100.        df['Sub_metering_2'] = df['Sub_metering_2'] / 1000
101.        df['Sub_metering_3'] = df['Sub_metering_3'] / 1000
102.
103.        # Cambio de potencia a energia (kWh)
    
```

```

103.         df['Energia activa global'] = df['Global_active_power'] / 60
104.
105.         # Medidor 4
106.         df['Sub_metering_4'] = df['Energia activa global'] - (
107.             df['Sub_metering_1'] + df['Sub_metering_2'] +
            df['Sub_metering_3'])
108.
109.         # Creacion del nuevo dataframe
110.         df_mean = df[['Global_intensity']].copy()
111.         df_sum = df.drop(['Global_intensity', 'Global_active_power'],
            axis=1)
112.
113.         # Resampling del nuevo dataframe
114.         df_mean = df_mean.resample('h').mean()
115.         df_sum = df_sum.resample('h').sum()
116.
117.         df1 = df_mean.merge(df_sum, left_index=True, right_index=True)
118.
119.         # Mover energia activa a la ultima posicion
120.         brb = df1.pop('Energia activa global') # remove column b and
            store it in brb
121.         df1.insert(0, 'Energia activa global', brb) # en primera
            posicion
122.
123.         # Únicamente interesan las últimas 24 horas
124.         n_features = df1.shape[1]
125.
126.         # Seleccion de los valores de la ultimas 24h
127.         df1 = df1.tail(self.n_hours)
128.
129.         # Escalado
130.         values = df1.values
131.         scaled = self.scaler.transform(values)
132.
133.         # Para deshacer el escalado despues
134.         self.scaled_last_row = scaled[-1:, 1:]
135.
136.         # Series to supervised
137.         data_to_model = series_to_supervised(scaled, self.n_hours - 1,
            1)
138.
139.         # Redimensionar el input a 3d [muestras, secuencias, variables]
140.         data_to_model =
            data_to_model.values.reshape((data_to_model.values.shape[0],
            self.n_hours,
            n_features))
141.         return data_to_model
142.
143.         # TESTING
144.         def adapt_data_test(self, df):
145.
146.             # Tratamiento de NaN
147.             df = df.interpolate()
148.             df.isnull().sum()
149.
150.             # Eliminar voltaje y energia reactiva
151.             df = df.drop(['Voltage', 'Global_reactive_power'], axis=1)
152.
153.             # Cambio de unidades a KWh
154.             df['Sub_metering_1'] = df['Sub_metering_1'] / 1000
155.             df['Sub_metering_2'] = df['Sub_metering_2'] / 1000
156.             df['Sub_metering_3'] = df['Sub_metering_3'] / 1000
157.
158.             # Cambio de potencia a energia (kWh)
159.             df['Energia activa global'] = df['Global_active_power'] / 60
160.
    
```

```

161.         # Medidor 4
162.         df['Sub_metering_4'] = df['Energia activa global'] - (
163.             df['Sub_metering_1'] + df['Sub_metering_2'] +
            df['Sub_metering_3'])
164.
165.         # Creacion del nuevo dataframe
166.         df_mean = df[['Global_intensity']].copy()
167.         df_sum = df.drop(['Global_intensity', 'Global_active_power'],
            axis=1)
168.
169.         # Resampling del nuevo dataframe
170.         df_mean = df_mean.resample('h').mean()
171.         df_sum = df_sum.resample('h').sum()
172.
173.         df1 = df_mean.merge(df_sum, left_index=True, right_index=True)
174.
175.         # Mover energia activa a la ultima posicion
176.         brb = df1.pop('Energia activa global') # remove column b and
            store it in brb
177.         df1.insert(0, 'Energia activa global', brb) # en primera
            posicion
178.
179.         # Únicamente interesan las últimas 24 horas
180.         n_features = df1.shape[1]
181.
182.         # Seleccionar punto de partida aleatorio en el dataset. Misma
            hora, dia aleatorio
183.
184.         # Seleccion de valor random. Una vez por ejecucion
185.         if (self.is_df_randomized == 0):
186.             self.is_df_randomized = 1
187.             # Conseguir indice del dia aleatorio
188.             df2 = df1.copy()
189.             df2['hour'] = df2.index.hour
190.             df2 = df2.reset_index()
191.             # Mask de todos los puntos con la misma hora
192.             df2 = df2.loc[df2.hour == datetime.now().hour]
193.             random_first_index = random.randint(25, len(df2) - 25)
194.             # Equivalencia de indice en el df original
195.             self.starting_indice = df2.iloc[random_first_index].name
196.
197.         # Recortar dataframe al minimo requerido
198.         df1 = df1.iloc[self.starting_indice - 24:self.starting_indice]
199.
200.         # Aumentar indice para siguiente iteracion
201.         self.starting_indice += 1
202.
203.         # Escalado
204.         values = df1.values
205.         scaled = self.scaler.transform(values)
206.
207.         # Para deshacer el escalado despues
208.         self.scaled_last_row = scaled[-1:, 1:]
209.
210.         # Series to supervised
211.         data_to_model = series_to_supervised(scaled, self.n_hours - 1,
            1)
212.
213.         # Redimensionar el input a 3d [muestras, secuencias, variables]
214.         data_to_model =
            data_to_model.values.reshape((data_to_model.values.shape[0],
            self.n_hours,
            n_features))
215.         return data_to_model
216.
217.     def compare_power(self, consumo, generacion):
    
```

```

218.         # Estados independientes de la comparacion:
219.         if (self.bateria.battery_level >= self.bateria.max_level_day):
220.             self.bateria.cargar_bateria = 0
221.         elif (self.bateria.battery_level <= self.bateria.min_level_day):
222.             self.bateria.cargar_bateria = 1
223.
224.         elif (generacion < consumo * self.x_security):
225.             self.bateria.cargar_bateria = 1
226.         elif (generacion >= consumo * self.x_security):
227.             self.bateria.cargar_bateria = 0
228.
229.         def set_irradiance_next_hour(self):
230.             self.irradiance = get_irradiance_next_hour()
231.             print('Update forecasted irradiance. New value for hour ' +
str(datetime.now().hour + 1) + ' is: ' + str(
232.                 self.irradiance))
233.
234.         def power_from_irradiance(self):
235.             power = self.irradiance * self.rendimiento * (self.sup_paneles /
1000)
236.             return power
237.
238.         def bucle_normal(self):
239.             print()
240.             print('Entrando en el bucle a las: ', datetime.now())
241.             if (self.is_valley() == 0):
242.                 # 3. Lectura de datos
243.                 print('Cargando datos del csv a las: ', datetime.now())
244.                 data = self.load_data()
245.
246.                 # 4. Adaptar datos
247.                 data = self.adapt_data(data)
248.
249.                 # 5. Realizar la prediccion
250.                 print('Realizando prediccion a las: ', datetime.now())
251.                 consumo_prediction = self.make_prediction(data)
252.
253.                 # 6. Obtener el valor de la generacion y potencia en la
proxima hora
254.                 generation = self.power_from_irradiance()
255.
256.                 # . Comparar ambos valores
257.                 self.compare_power(consumo_prediction, generation)
258.                 write_to_csv_RT('../Data/charge_battery_data.csv',
['charge_battery', 'consumo', 'generation'],
259.                     [self.bateria.cargar_bateria,
consumo_prediction, generation])
260.
261.         def bucle_test(self):
262.             print()
263.             print('Entrando en el bucle a las: ', datetime.now())
264.             if (self.is_valley() == 0):
265.                 # 3. Lectura de datos
266.                 print('Cargando datos del csv a las: ', datetime.now())
267.                 data = self.load_data()
268.
269.                 # 4. Adaptar datos
270.                 data = self.adapt_data_test(data)
271.
272.                 # 5. Realizar la prediccion
273.                 print('Realizando prediccion a las: ', datetime.now())
274.                 consumo_prediction = self.make_prediction(data)
275.
276.                 # 6. Obtener el valor de la generacion y potencia en la
proxima hora
    
```



```

277.             generation = self.power_from_irradiance()
278.
279.             # . Comparar ambos valores
280.             self.compare_power(consumo_prediction, generation)
281.             write_to_csv_RT('../Data/charge_battery_data.csv',
                ['charge_battery', 'consumo', 'generation'],
282.             [self.bateria.cargar_bateria,
                consumo_prediction, generation])
    
```

## main.py

```

1. from smart_algorithmym import SmartAlgorithmym
2. from apscheduler.schedulers.background import BackgroundScheduler
3. import time
4.
5. # 1. Crear objeto algoritmo
6. al = SmartAlgorithmym()
7.
8. # 2. Scheduler
9. scheduler = BackgroundScheduler()
10. scheduler.add_job(al.bucle_normal, 'cron', hour='*')
11. scheduler.add_job(al.set_irradiance_next_hour, 'cron', hour='*', minute='58')
12. scheduler.add_job(al.charge_battery_night, 'cron', minute='*', second='30')
13. scheduler.start()
14.
15. print('Starting')
16. try:
17.     while True:
18.         time.sleep(1)
19.     except KeyboardInterrupt:
20.         pass
21.
22. scheduler.shutdown()
    
```

## test.py

```

1. from smart_algorithmym import SmartAlgorithmym
2. from apscheduler.schedulers.background import BackgroundScheduler
3. import time
4.
5. # 1. Crear objeto algoritmo
6. al = SmartAlgorithmym()
7.
8. # 2. Scheduler
9. scheduler = BackgroundScheduler()
10. scheduler.add_job(al.bucle_test, 'cron', hour='*')
11. scheduler.add_job(al.set_irradiance_next_hour, 'cron', hour='*', minute='55')
12. scheduler.add_job(al.charge_battery_night, 'interval', minutes=15)
13. scheduler.start()
14.
15. print('Starting')
16. try:
17.     while True:
18.         time.sleep(1)
19.     except KeyboardInterrupt:
20.         pass
21.
22. scheduler.shutdown()
    
```