Trabajo Fin de Grado

Grado en Ingeniería Eléctrica

Predicción de la demanda de energía eléctrica con aprendizaje automático

Autor: Raúl Ruiz Pérez

Tutor: Narciso Moreno Alfonso

Departamento de Ingeniería Eléctrica

Escuela Politécnica Superior

Universidad de Sevilla

Sevilla, 2023

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

RESUMEN

La demanda de energía eléctrica es un parámetro significativo en el análisis de consumos de la sociedad. Por ello, es importante conocer más detalles sobre su comportamiento y evolución temporal en diferentes sectores: residencial, edificios públicos, industria, etc. La inteligencia artificial es una herramienta muy eficiente que nos permite estudiar y analizar variables de estas características, ya que maneja de forma muy sencilla una gran cantidad de datos e información relacionada con el problema a tratar. En este proyecto se aborda el estudio de la predicción de demanda de energía eléctrica mediante la aplicación de técnicas de inteligencia artificial y Machine Learning (aprendizaje automático), enfocándose en un ejemplo práctico acerca de un consumidor concreto.

Palabras clave:

Machine Learning, Tensorflow, demanda eléctrica, predicción, redes neuronales.

ABSTRACT

The demand for electricity is a significant parameter in the analysis of society's consumption. Therefore, it is important to know more details about their behavior and evolution over time in different sectors: residential, public buildings, industry, etc. Artificial intelligence is a very efficient tool that allows us to study and analyze variables of these characteristics, since it handles a large amount of data and information related to the problem to be treated in a very simple way. This project explains the study of the prediction of electricity demand through the application of artificial intelligence and Machine Learning techniques, focusing on a practical case about a specific consumer.

Keywords:

Machine Learning, Tensorflow, Power Demand, Prediction, Neural Networks.

1. INTRODUCCIÓN

1.1. Motivación

La inteligencia artificial se encuentra al alza de forma mediática, aunque lleve muchos años establecida en procesos cotidianos de la sociedad. Esto provoca un aumento en el interés por investigar y desarrollar nuevos proyectos sobre esta tecnología.

La predicción de demanda de energía eléctrica es un concepto muy útil para el correcto funcionamiento del sistema eléctrico. Conocer valores de consumo de electricidad nos ayuda a coordinar la generación de dicha energía, gestionar el coste de mercado de la energía de forma previa, organizar y modificar los hábitos de consumo para reducir costes en la factura de la luz, entre otros.

La fusión de ambos conceptos permite aplicar tecnologías con mucho potencial de crecimiento en un campo fundamental y de estudio diario como es la energía eléctrica.

1.2. Objetivos del proyecto

El objetivo principal del proyecto es crear una herramienta que genere predicciones de valores de demanda de energía eléctrica a partir de un conjunto de datos históricos conocidos, anteriores en el tiempo a dicha predicción.

El componente básico de dicha herramienta estará formado por modelos basados en técnicas de inteligencia artificial y Machine Learning, los cuales serán programados para procesar datos de consumo reales como entrada y, tras realizar un entrenamiento con ellos, se obtenga a la salida una predicción de consumo para un instante de tiempo desconocido para el modelo.

Posteriormente, una vez obtenidos los resultados de los entrenamientos y de las predicciones generadas por los modelos, se evaluará el grado de ajuste y la fiabilidad de los valores obtenidos. Para ello, se calcularán métricas y valores de error durante los procesos de entrenamiento y prueba, los cuales nos ayudarán a determinar si el resultado que hemos conseguido es apto o, por el contrario, se debe seguir ajustando los parámetros de los modelos para obtener una precisión mayor en las predicciones.

2. ¿POR QUÉ NOS CENTRAMOS EN LA DEMANDA DE ENERGÍA ELÉCTRICA?

La energía eléctrica es una de las principales fuentes de consumo en la actualidad. Desde la industria hasta un dispositivo electrónico cotidiano necesita de energía eléctrica para poder llevar a cabo su función. Al ser un recurso tan demandado por la sociedad, realizar un buen control y una adecuada gestión sobre el mismo permite aprovechar en mayor porcentaje su potencial.

Por otra parte, el sistema eléctrico está formado por toda la infraestructura de generación, transporte y distribución de energía eléctrica. Este sistema necesita mantenerse estable para garantizar un correcto suministro eléctrico a los consumidores. Dicha estabilidad depende principalmente de compensar los valores de generación y demanda de energía eléctrica.

El concepto demanda de energía eléctrica se utiliza para conocer la cantidad de energía eléctrica que se consume. Puede ser en todo un país, en una ciudad, un polígono, hasta una vivienda o partes de ella. Dicho concepto tiene un valor económico, el cual lo hace fundamental para la sociedad. Por este motivo, monitorizar y conocer el consumo eléctrico es una tarea necesaria no sólo para las compañías eléctricas, que distribuyen dicha energía, sino para todos los consumidores.

Además, el precio de la energía en el mercado eléctrico se cierra el día anterior al que se evalúa, de forma que las compañías deben hacer sus ofertas sin conocer realmente la cantidad de energía que van a distribuir, sino basándose en una estimación. De igual forma, conocer un valor estimado de demanda eléctrica puede ayudar a organizar ciertos hábitos de consumo que influyen en el coste de la electricidad de forma negativa.

La relevancia y la necesidad de realizar estimaciones de energía eléctrica, provoca la implementación de tecnologías como la inteligencia artificial en este tipo de cuestiones. Dicha tecnología supone una mejora en los resultados obtenidos gracias a su capacidad para manejar grandes cantidades de datos además de sus algoritmos basados en aprendizaje automático con los que se logra automatizar todo el proceso de ciencia de datos.

En resumen, existen numerosas ventajas al conocer datos de energía eléctrica ya sean presentes, pasados o futuros. Toda información que se conozca sobre un concepto es objeto de estudio para su mejora en diversos aspectos. La monitorización junto con la predicción de la demanda de energía eléctrica ayuda a resolver situaciones como las descritas anteriormente de forma eficiente, por ejemplo: prevenir situaciones problemáticas al observar datos similares a los recogidos en un registro histórico durante una situación similar, conocer en tiempo real el consumo eléctrico para evitar posibles problemas en la instalación, gestionar los hábitos de consumo según el precio de mercado por horas, conocer una estimación de demanda de energía eléctrica para realizar el reparto de la generación de la misma de forma adecuada y minimizar costes, estudiar la necesidad de ampliar o mejorar la infraestructura del sistema eléctrico, etc.

3. APRENDIZAJE AUTOMÁTICO

3.1. Machine Learning

Machine Learning es “una rama de la inteligencia artificial (IA) y la informática que se centra en el uso de datos y algoritmos para imitar la forma en la que aprenden los seres humanos, con una mejora gradual de su precisión” [1].

“Es un componente importante del creciente campo de la ciencia de datos. Mediante el uso de métodos estadísticos, los algoritmos se entrenan para hacer clasificaciones o predicciones, y descubrir información clave dentro de los proyectos de minería de datos. Esta información clave facilita posteriormente la toma de decisiones dentro de las aplicaciones y las empresas, lo que afecta idealmente a las métricas de crecimiento clave. La expansión y el crecimiento de Big Data van a venir acompañados de un aumento de la demanda de científicos de datos en el mercado. Se les pedirá que identifiquen las cuestiones más relevantes para la empresa y los datos necesarios para resolverlas” [1].

3.1.1. Clasificación de métodos de ML

- El aprendizaje supervisado “se define por su uso de los conjuntos de datos etiquetados para entrenar los algoritmos para clasificar datos o predecir resultados con precisión. A medida que se introducen datos de entrada en el modelo, este adapta sus pesos hasta que se haya ajustado correctamente” [1].

  El aprendizaje supervisado se puede dividir en dos tipos de problemas cuando se extraen datos: clasificación y regresión. “Los problemas de clasificación utilizan un algoritmo para asignar con precisión datos de prueba en categorías específicas mientras que la regresión es otro tipo de método de aprendizaje supervisado que utiliza un algoritmo para comprender la relación entre variables dependientes e independientes” [2].

  Algunos métodos utilizados en el aprendizaje supervisado son las redes neuronales, la regresión logística, el bosque aleatorio y la máquina de vectores de soporte (SVM).

- El aprendizaje no supervisado “utiliza algoritmos de Machine Learning para analizar y agrupar conjuntos de datos sin etiquetar. Estos algoritmos descubren agrupaciones de datos o patrones ocultos sin necesidad de ninguna intervención humana” [1].

  Los modelos de aprendizaje no supervisados se utilizan para tres tareas principales: agrupación, asociación y reducción de dimensionalidad. “La agrupación es una técnica de minería de datos para agrupar datos sin etiquetar en función de sus similitudes o diferencias. La asociación utiliza diferentes reglas para encontrar relaciones entre variables en un conjunto de datos determinado. La reducción de dimensionalidad es una técnica de aprendizaje que se utiliza cuando la cantidad de características (o dimensiones) en un conjunto de datos determinado es demasiado alta” [2].
- El aprendizaje semisupervisado comprende un punto intermedio entre el aprendizaje supervisado y no supervisado. “Durante el entrenamiento, utiliza un conjunto de datos etiquetados más pequeño para guiar la clasificación y la extracción de características de un conjunto de datos sin etiquetar de mayor tamaño. El aprendizaje semisupervisado puede resolver el problema de no tener suficientes datos etiquetados para un algoritmo de aprendizaje supervisado. También es útil si el coste de etiquetar datos suficientes es demasiado elevado” [1].

3.1.2. Big Data

Big Data es un concepto que atiende principalmente “la gestión y análisis de enormes volúmenes de datos que no pueden ser tratados de manera convencional, ya que superan los límites y capacidades de las herramientas de software habitualmente utilizadas para la captura, gestión y procesamiento de datos” [3].

Su definición es también conocida como “las cuatro V, que representan el gran volumen de datos que debe ser capaz de tratar, la velocidad con la que puede procesar esos datos, y la variedad de formas que pueden tomar los mismos. Debemos hacer énfasis en el objetivo del Big Data añadiendo una cuarta V, la del valor que se obtiene por la información extraída de los datos” [4].

3.2. Deep Learning

Deep Learning es un concepto que surge a partir de la tecnología Machine Learning. Cómo su nombre indica, es una parte del aprendizaje automático más profundo, es decir, se trata de modelos que disponen de varias capas ocultas, las cuales otorgan un mayor grado de aprendizaje al modelo y un mayor grado de precisión al realizar predicciones.

Deep Learning "impulsa muchos servicios y aplicaciones de inteligencia artificial (IA) que mejoran la automatización, realizando tareas analíticas y físicas sin intervención humana. La tecnología de Deep Learning reside detrás de muchos productos y servicios de uso cotidiano (como los asistentes digitales, los controles de TV habilitados por voz y la detección de fraudes con tarjeta de crédito), así como de tecnologías emergentes (como los automóviles autónomos)" [5].

3.2.1. Concepto de red neuronal

Una red neuronal es un modelo de inteligencia artificial basado en la estructura interna del cerebro humano. Replica de forma artificial las conexiones entre neuronas para la transmisión de información entre ellas. Aplicado a la ciencia de datos, consigue detectar patrones entre los datos y utiliza dicha información para realizar clasificaciones, predicciones, etc.

Las redes neuronales "descomponen las entradas en capas de abstracción. Se pueden entrenar con muchos ejemplos para que reconozcan patrones de voz o en imágenes, por ejemplo, igual que el cerebro humano. Su comportamiento está definido por la forma en que se conectan sus elementos individuales, así como por la importancia (o ponderación) de dichas conexiones. Estas ponderaciones se ajustan automáticamente durante el entrenamiento de acuerdo con una regla de aprendizaje especificada hasta que la red neuronal lleva a cabo la tarea deseada correctamente" [6].

3.2.2. Tipos de redes neuronales

- Redes neuronales profundas DNN (Deep Neural Networks): son redes neuronales que constan de una capa de entrada, una capa de salida y más de una capa oculta. Las capas normalmente son de tipo denso, es decir, todas las neuronas de la capa anterior están conectadas con todas las neuronas de la siguiente.
- Redes neuronales recurrentes RNN (Recurrent Neural Networks): están formadas por capas ocultas que tienen recurrencia en el análisis de datos, es decir, tienen acceso a datos anteriores en la secuencia para así establecer y replicar ciertos patrones. Estas capas son de tipo LSTM (Long Short Term Memory) las cuales guardan datos en memoria a corto y largo plazo para ser utilizados por el modelo.
- Redes neuronales convolucionales CNN (Convolutional Neural Networks): son redes en las que “diferentes partes, se pueden entrenar para tareas diversas. Con esto se aumenta la velocidad de entrenamiento e identifican patrones de una forma más avanzada. Gracias al uso de algoritmos para redes convolucionales, se reduce el número de conexiones y de parámetros, por lo que requiere de menor entrenamiento” [7].

3.2.3. Ventajas

- “Su principal ventaja está en que son modelos de vanguardia que capturan de una forma óptima y efectiva características complejas, obteniendo resultados con una alta precisión” [8].
- “El procesado de la información es local, es decir, al estar compuesto por unidades individuales de procesamiento, dependiendo de sus entradas y pesos, y de que todas las neuronas de una capa trabajan en forma paralela, proporcionan una respuesta al mismo tiempo” [8].
- “Las neuronas son tolerantes a fallos, si parte de la red no trabaja, solo dejará de funcionar la parte para que dicha neurona sea significativa, el resto tendrá su comportamiento normal” [8].

- “Las neuronas pueden reconocer patrones que no han sido aprendidos, sólo deben tener cierto parecido con el conocimiento previo que tenga la red” [8].

3.2.4. Inconvenientes

- “Las redes neuronales necesitan un mayor preprocesamiento de los datos, siendo bastante sensibles a las distintas escalas de las variables. Suelen necesitar mayor volumen de datos para el entrenamiento del modelo y requieren de alta capacidad de recursos computacionales” [8].
- “Complejidad de aprendizaje para grandes tareas, cuanto más se necesite que aprenda una red, más complicado será enseñarle” [8].
- “Tiempo de aprendizaje elevado. Esto depende de dos factores: primero si se incrementa la cantidad de patrones a identificar o clasificar, y segundo, si se requiere mayor flexibilidad o capacidad de adaptación de la red neuronal para reconocer patrones que sean sumamente parecidos” [8].
- “No son fácilmente explicables. Conocer las reglas o motivos por los que la red devuelve esos resultados no suele ser fácil y precisa de otras analíticas” [8].

3.3. Comparativa frente a métodos estadísticos clásicos

Los métodos estadísticos clásicos “utilizan ecuaciones matemáticas para codificar información extraída de los datos. En algunos casos, las técnicas de modelado estadístico pueden proporcionar modelos adecuados de forma rápida. Incluso en el caso de problemas en los que las técnicas más flexibles de aprendizaje de las máquinas (como redes neuronales) pueden ofrecer a la postre mejores resultados, es posible usar algunos modelos estadísticos como modelos predictivos de línea base para juzgar el rendimiento de técnicas más avanzadas” [9].

Existen numerosos estudios en los que se comparan resultados entre modelos basados en inteligencia artificial y modelos basados en métodos estadísticos clásicos.

A pesar de que muchos de los modelos de inteligencia artificial son derivados de métodos estadísticos clásicos o sus algoritmos están basados en ellos, se encuentran diferencias relevantes en algunos aspectos como el grado de ajuste del modelo o la precisión a la hora de realizar predicciones.

Por ejemplo, en [10] se estudia el pronóstico de consumo de energía eléctrica residencial de corto plazo utilizando algoritmos de aprendizaje automático y profundo. Para esta investigación, se crean modelos de predicción utilizando el algoritmo de regresión lineal múltiple y una red neuronal artificial. Como resultados, se obtuvo un mejor desempeño del modelo basado en redes neuronales en cuanto a la calidad del ajuste (79,7% frente a 78,2%) y de las métricas de error.

A pesar de que el modelo basado en métodos estadísticos (RLM) obtuvo valores aceptables y cercanos a los de la red neuronal (RNA), se realizaron predicciones para los meses de julio y agosto en cuyos resultados se muestra la notable diferencia entre ambos modelos.

Tabla 4

Predicciones Julio – Agosto 2022.

Valores	Julio 2022	Agosto 2022
Valor Real Consumo (MWh)	404.066,80	391.854,90
Predicción (MWh) - RLM	350.919,29	355.635,89
Error porcentual	13,15%	9,24%
Predicción (MWh) - RNA	382.369,97	388.530,19
Error porcentual	5,37%	0,85%

Figura 1. Predicciones julio y agosto 2022 [10].

Por otro lado, en [11] se realiza una comparación entre seis modelos distintos. Los cuatro primeros están basados en métodos estadísticos para series de tiempo y los dos restantes en redes neuronales con distintas estructuras. A su vez, se estudian ocho series de tiempo diferentes.

Tras la aplicación de los distintos modelos a las series de tiempo, se concluye que “los modelos de redes neuronales multicapa permiten analizar y obtener pronósticos precisos por sus capacidades auto-adaptativas, generalización, aprendizaje y representación de funciones no lineales en las series de tiempo con patrón de tendencia fluctuante en el tiempo” [11].

De igual forma, se observa que los modelos de redes neuronales, a pesar de obtener menores valores de error porcentual absoluto para todos los casos de series temporales, no se ajustan de igual forma a todas las series con respecto a otras métricas de error y, por tanto, no realizan predicciones igual de precisas para todos los casos. Se puede concluir que es necesario adaptar el modelo a la serie de tiempo que se estudia y, en el caso de las redes neuronales, adaptar su topología.

Por último, en [12] se tiene como objetivo comparar la precisión de las predicciones de los modelos basados en métodos estadísticos y en Machine Learning para series de tiempo multivariadas.

Se determina que “no existe un modelo que domine a los otros en todos los conjuntos de datos. No obstante, los modelos estadísticos vector autorregresivo (VAR) y modelo de corrección de errores vectoriales (VECM) junto con sus versiones con dummies para considerar la estacionalidad, muestran una ligera superioridad sobre los modelos de Machine Learning en la precisión de sus predicciones” [12].

Además, “en los casos donde las series multivariadas contaban con mayor número de variables, el desempeño de los modelos de Machine Learning fue mejor según las métricas de error. Mientras que, en el resto de series, donde se contaba con tres y cuatro variables, el desempeño fue mejor en los modelos estadísticos en la mayoría de los casos” [12].

Se deduce de nuevo que, en el estudio de diversos conjuntos de datos con características diferentes, a pesar de obtener mejores resultados generales y la gran adaptabilidad que poseen los modelos de inteligencia artificial, se pueden conseguir mejores resultados con un método estadístico aplicado a un caso concreto.

Por otro lado, se aprecia la capacidad de los modelos basados en Machine Learning de reconocer patrones en los datos y relaciones entre variables que los métodos estadísticos clásicos no alcanzan a detectar. Por ello, como se menciona anteriormente, al ampliar el número de variables y, por tanto, el número de dependencias entre datos, se deducen mejores resultados en los modelos de inteligencia artificial.

Tras la revisión bibliográfica realizada a diversos estudios en los que se comparan diversos modelos estadísticos frente a modelos de Machine Learning se concluye:

- En aspectos generales destaca la gran versatilidad del Machine Learning, destacando las redes neuronales, las cuales alcanzan resultados aceptables para la gran mayoría de situaciones.
- En contraposición, los modelos estadísticos pueden arrojar mejores predicciones para ciertas situaciones en las que se adapten mejor a las características de los datos.
- Los modelos de inteligencia artificial trabajan mejor con grandes cantidades de datos y con mayor número de variables. Además, detectan patrones y relaciones complejas para los modelos clásicos.
- El coste computacional es mayor cuando se trata de modelos de Machine Learning.
- Resulta fundamental estudiar las características de los datos y las variables a estudiar y adaptar el modelo a las mismas. Se puede realizar aplicando varios tipos de modelos diferentes o modificando la estructura y los parámetros hasta conseguir una mejor adaptación y, por tanto, un mejor ajuste del modelo.

4. PYTHON

4.1. Características básicas

Python es “un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador. Es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Python ha ido ganando adeptos gracias a su sencillez y a sus amplias posibilidades, sobre todo en los últimos años, ya que facilita trabajar con inteligencia artificial, Big Data, Machine Learning y ciencia de datos, entre muchos otros campos en auge” [13].

Figura 4.1. Logo de Python [2].

4.2. Ventajas e inconvenientes

4.2.1. Ventajas de Python.

- Lenguaje de alto nivel: “Python es un lenguaje de alto nivel, por lo que es más fácil de usar que los de bajo nivel, puesto que estos últimos no tienen mucha abstracción de lenguaje de máquina. Para programar con Python se pueden usar elementos del lenguaje natural, ya que tiene una sintaxis similar al inglés, por lo que es fácil de leer, escribir y aprender” [14].
- Polivalencia: “al ser un lenguaje de propósito general, se puede usar para diversos propósitos. Es una gran opción para el desarrollo de software, ya que permite a los desarrolladores utilizar grandes frameworks como Django y Flask. Además, se puede utilizar para scripts web, desarrollo de GUI de escritorio o data science” [2].

- Librerías: “el mayor beneficio es que tiene una amplia colección de bibliotecas y frameworks. La biblioteca estándar de Python es muy extensa, puesto que contiene muchos módulos integrados. Además, los usuarios de Python también pueden encontrar bibliotecas adicionales disponibles en PyPI (índice de paquetes de Python)” [2].
- Portabilidad: “es compatible con todos los sistemas operativos (macOS, Linux, UNIX y Windows), y los programadores solo necesitan escribir código una vez y luego podrá ejecutarse en todas partes” [2].
- Baja curva de aprendizaje: “la sencillez de la sintaxis de Python permite escribir programas totalmente funcionales en pocas líneas de código, por lo que su curva de aprendizaje es muy baja” [2].
- Comunidad: “el hecho de que sea gratuito y de código abierto contribuye a crear una comunidad sólida. Los programadores de Python pueden descargar el código fuente, modificarlo y distribuirlo como deseen” [2].

4.2.2. Desventajas de Python

- Lentitud: “la lentitud de Python se debe principalmente a su naturaleza dinámica y versatilidad. No obstante, hay formas de optimizar las aplicaciones de Python aprovechando la sincronización, entendiendo las herramientas de creación de perfiles y considerando el uso de múltiples intérpretes” [2].
- Consumo de memoria: “en el caso de que una tarea requiera mucha memoria, Python no es la mejor opción. El consumo de memoria de Python es muy alto, y esto se debe a la flexibilidad de los tipos de datos” [2].
- Desarrollo móvil: “Python es ideal para plataformas de escritorio y servidor, pero para el desarrollo móvil no es un lenguaje muy adecuado. Por este motivo, apenas vemos aplicaciones móviles desarrolladas con Python” [2].

4.3. Librerías

4.3.1. Pandas

Pandas es una librería “para realizar análisis de datos prácticos del mundo real en Python. Algunas de sus funcionalidades son: objeto DataFrame rápido y eficiente para la manipulación de datos, leer y escribir datos entre estructuras y diferentes formatos, manejo de datos faltantes, operaciones de división-aplicación-combinación de datos, manejo de series temporales. Python con pandas se utiliza en una amplia variedad de dominios académicos y comerciales, como las finanzas, la neurociencia, la economía, la estadística, la publicidad, la analítica web y más” [15].

Figura 2. Logo de pandas [15].

4.3.2. NumPy

NumPy es una librería de Python que “proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en matrices, incluidas matemáticas, lógicas, manipulación de formas, clasificación, selección, transformadas de Fourier, álgebra lineal básica, operaciones estadísticas y simulación aleatoria” [16].

Figura 3. Logo de Numpy [16].

4.3.3. Matplotlib

Matplotlib es una “librería de Python especializada en la creación de gráficos en dos dimensiones. Permite crear y personalizar los tipos de gráficos más comunes, entre ellos: diagramas de barras, histograma, diagramas de sectores, diagramas de caja y bigotes, diagramas de dispersión o puntos, diagramas de líneas, diagramas de áreas, mapas de color y combinaciones de todos ellos” [17].

Figura 4. Logo de Matplotlib [17].

4.3.4. Seaborn

Seaborn es una "biblioteca de visualización de datos de Python basada en Matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos" [18].

Figura 5. Logo de Seaborn [18].

4.3.5. TensorFlow y Keras

TensorFlow es una “plataforma de extremo a extremo que facilita tanto la creación como la implementación de modelos de aprendizaje automático. TensorFlow ofrece varios niveles de abstracción para que puedas elegir el que se adecue a tus necesidades. Crea y entrena modelos mediante la API de alto nivel de Keras, que ayuda a que los primeros pasos con TensorFlow y el aprendizaje automático sean sencillos” [19] [20] .

Figura 6. Logo de TensorFlow [19].

Figura 7. Logo de Keras [20].

4.3.6. Scikit-Learn

Scikit-Learn es una biblioteca de Python que ofrece “herramientas sencillas y eficientes para el análisis predictivo de datos, basado en NumPy, SciPy y matplotlib. Algunas de sus funciones son: predicción de un atributo de valor continuo asociado a un objeto, agrupación automática de objetos similares en conjuntos, extracción y normalización de características, comparación, validación y elección de parámetros y modelos” [21].

Figura 8. Logo de Scikit-Learn [21].

4.4. Motivo de elección

Python es un lenguaje de programación que cuenta con una gran versatilidad en cuanto a herramientas de programación se refiere. A la hora de realizar un proyecto que engloba procesos de diversas características, es fundamental manejar instrumentos que nos permitan realizar todos los requerimientos que sean necesarios de forma adecuada.

Cuenta con una gran cantidad de librerías especializadas en infinidad de funciones, las cuales se mantienen en constante desarrollo para ofrecer a sus usuarios una funcionalidad actualizada y con la menor cantidad de errores posible.

Este proyecto alberga procesos desde programación básica hasta implementación de modelos de inteligencia artificial, pasando por análisis estadísticos, ciencia de datos, creación de gráficos, entre otros.

Python permite trabajar todos estos ámbitos en un mismo lenguaje y de forma muy sencilla e intuitiva. Por ello, es uno de los lenguajes de programación más conocidos y usados en la actualidad, y se trata de la herramienta principal con la que se desarrolla el caso práctico.

5. CASO PRÁCTICO

5.1. Definición del problema y metodología

5.1.1. Características del consumidor

Existen estudios e investigaciones que abordan problemas de demanda de energía eléctrica de diversas formas, desde un enfoque más general hasta casos muy específicos y localizados.

En este proyecto, como se ha mencionado anteriormente, se trabajará con datos de consumo sobre un consumidor en concreto. Dicho consumidor se trata del edificio de la antigua Escuela de Peritos Industriales y actual Escuela Politécnica Superior que pertenece a la Universidad de Sevilla. Se encuentra en la calle Virgen de África número 7, Sevilla. Este edificio es de carácter público y está destinado a la enseñanza de alumnos y a albergar oficinas y servicios administrativos de la institución correspondiente la cual, como se ha comentado anteriormente, es la Universidad de Sevilla.

5.1.2. Metodología

Para comenzar, es importante conocer el problema que se va a afrontar. En este caso, cómo se ha mencionado anteriormente, se trata de un edificio público destinado a impartir cátedra a alumnos y a servicios de oficina de la Universidad de Sevilla.

Dicha información ayudará a interpretar si los valores de consumo de energía eléctrica que se tienen en la base de datos tienen magnitudes acordes con el ejemplo que se está estudiando.

Por otro lado, las características del sujeto de estudio, permitirán valorar si es adecuado realizar un estudio de los datos a tiempo completo, es decir, tomando todos los registros históricos existentes en la base de datos o, por el contrario, excluir ciertos intervalos de tiempo en los cuales los registros no se ajustan a la realidad o pueden perjudicar al modelo produciendo una desviación de su ajuste más óptimo.

Cuanto mayor sea la información de la que disponemos acerca del problema que se va a afrontar, mayor será el grado de comprensión de los requerimientos y necesidades del modelo a implementar, además de incrementar la capacidad de mejorar ciertas funciones del proceso al conocer el contexto de dichos requerimientos.

Una vez conocida la procedencia y las características de la variable a estudiar, se debe estudiar qué tipo de modelos son más adecuados en cada caso y posteriormente, elegir uno o varios de ellos que se ajusten a la situación que se desea examinar.

Resultando elegido el modelo que se va a crear, el primer requisito es preprocesar el set de datos con el que se trabajará durante el proyecto. En este paso se organizará la base de datos de la forma más beneficiosa para facilitar la localización y el análisis de valores durante el desarrollo del modelo.

Además, se modificarán los formatos de las variables y columnas que así lo requieran. Esto será necesario en este estudio ya que algunas librerías de Python que son utilizadas, facilitan mucho todo el trabajo de implementación de código, pero a su vez, al ser un código ya generado anteriormente, se deben adaptar las variables al formato específico requerido por los comandos existentes en la librería.

Otro apartado clave en la preparación de los datos, es el manejo de valores atípicos y datos faltantes o nulos. Cómo se ha mencionado anteriormente, los valores que no se encuentren en rangos apropiados para el problema que se estudia, pueden provocar cambios en las tendencias que marca el modelo para generar patrones en el set de datos. Esto generaría un decremento en el ajuste del modelo y, por tanto, un mayor error a la hora de realizar predicciones.

El siguiente paso en el proceso será el análisis gráfico de las variables a estudiar. Resulta fundamental conocer el contexto que engloba el problema que se va a abordar y obtener una buena base de datos, pero al tratar con una gran cantidad de valores, la forma idónea de analizar el estudio es mediante el uso de gráficos.

Representar gráficamente los valores asignados a las variables que se van a estudiar permite, de forma sencilla y a simple vista, entender cómo evoluciona el problema. Esta evolución puede ser respecto a otros valores significativos como pueden ser el tiempo u otras variables con las que guarden dependencia entre sí.

Al mismo tiempo, permite visualizar los datos definitivos con los que se va a trabajar, por tanto, es una forma de comprobar y asegurar que no existen valores atípicos o nulos.

Para continuar con el proyecto, se pasa a la creación del modelo correspondiente para el estudio del problema que se plantea. Dicho modelo, como se comenta anteriormente, resultará previamente estudiado y elegido de entre todas las posibilidades que brinda la inteligencia artificial y el aprendizaje automático.

Se comenzará por definir distintos subconjuntos o también llamados paquetes de datos. Estos tendrán las siguientes denominaciones: entrenamiento, validación y prueba. No existe una norma definida para el reparto de ellos, pero de igual forma se ha establecido un estándar debido a su notable adaptación a la gran mayoría de casuísticas: 80% entrenamiento, 10% validación y 10% prueba.

En primer lugar, el paquete de entrenamiento se trata del subconjunto que permite, como su nombre indica, poder entrenar el modelo para que detecte patrones en los distintos valores que recibe como entrada.

Durante dicho proceso, entra en juego el subconjunto de validación, el cual en cada iteración “valida” que el modelo se esté entrenando de forma correcta. De forma más concreta, este paquete de datos se utiliza para realizar evaluaciones continuas al modelo y conseguir que el ajuste del mismo sea óptimo en cada fase del entrenamiento. Con ello se consigue que el modelo aprenda en cada iteración de forma autómata a detectar patrones en los datos de entrenamiento y posteriormente, compruebe dicho aprendizaje con los datos de validación.

Por último, el set de prueba resulta ser un subconjunto desconocido para el modelo. Por este motivo, se usa para realizar predicciones y comprobar que el aprendizaje con el resto de datos ha sido próspero.

Para facilitar la convergencia del entrenamiento, es recomendable escalar los datos de entrada del modelo. Este proceso es posible gracias a distintos escaladores numéricos que transforman los valores de nuestra base de datos a unos nuevos valores comprendidos entre unos límites menores, como pueden ser entre 0 y 1. Con este método se consigue que los cambios de escala entre las distintas variables de entrada no afecten en el entrenamiento y este consiga una convergencia más rápida.

Una vez definidos y escalados los distintos subconjuntos de datos, se comienza con la implementación de los modelos. En esta etapa se crea el modelo elegido y se definen los distintos parámetros de los que depende el mismo.

Los modelos del caso práctico son ambos modelos basados en redes neuronales artificiales. Los parámetros que se definen son:

- Número de capas: define el número de capas que formarán la red neuronal. Dicha red tendrá como mínimo una capa de entrada y una de salida, además de posibles capas intermedias llamadas capas ocultas.
- Número de neuronas por capa: indica el número de neuronas que incluirá cada capa de nuestra red.
- Función de activación: “transmite la información generada por la combinación lineal de los pesos y las entradas, es decir son la manera de transmitir la información por las conexiones de salida” [22].
- Función de pérdida: métrica definida para evaluar el error que se obtiene en cada iteración del entrenamiento.
- Optimizador: “optimiza los valores de los parámetros para reducir el error cometido por la red” [23].
- Tasa de aprendizaje: establece el intervalo de cambio en los ajustes de los parámetros. Afecta a la velocidad de aprendizaje del modelo. Un valor demasiado alto conseguiría un aprendizaje realmente veloz, aunque con menor precisión, y de forma contraria, un valor demasiado pequeño generaría un tiempo de entrenamiento demasiado extenso.
- Épocas de entrenamiento: hace referencia al número de iteraciones del entrenamiento, es decir, la cantidad de veces que el modelo va a recibir y analizar los datos de entrada.

- Tamaño de lotes: también conocido como batch size, es la cantidad de datos que tendrá cada lote de entrada al modelo. Este parámetro evita que el modelo reciba el set de datos completo. La subdivisión en lotes más pequeños permite a la red trabajar de forma más veloz en cada iteración.
- Número de entradas: suma de variables que otorgan datos de entrada al modelo.
- Número de predicciones: cantidad de predicciones que generará el modelo en cada iteración del entrenamiento.

La siguiente fase del proyecto es el entrenamiento y la validación del modelo. En este proceso se llevarán a cabo las iteraciones definidas, en las cuales el modelo irá detectando y aprendiendo patrones que se encuentran en los datos de entrenamiento de manera implícita.

A su vez, como se ha mencionado anteriormente, el subconjunto de validación realiza la función de comprobar que dicho aprendizaje es adecuado mediante evaluaciones que se realizan al modelo con este paquete de datos. De esta forma, se pretende que las futuras predicciones que se realicen con el modelo sigan los patrones extraídos del set de datos de entrenamiento.

En cada época del entrenamiento se puede analizar el grado de ajuste del modelo mediante métricas y la variable de pérdida que definimos previamente. La evolución de estos valores indica si existe convergencia en ellos o no, de forma que muestran si el entrenamiento se está desarrollando de forma correcta o se deben modificar parámetros del modelo para corregir fallos.

Dos conceptos fundamentales a evitar en el entrenamiento del modelo son el underfitting y el overfitting. “Un modelo con underfitting es aquel en donde los errores tanto de entrenamiento como de validación son similares y relativamente altos. Esto se debe a que el modelo es demasiado general. Por otra parte, en un modelo con overfitting se obtiene un error de entrenamiento relativamente bajo y uno de validación relativamente alto. En este caso, el modelo memoriza los datos de entrenamiento” [24].

Para finalizar nuestro proyecto, se realizan predicciones a partir del subconjunto de datos de prueba. Este paquete de datos hasta ahora es desconocido para el modelo, por tanto, es un método adecuado para simular una predicción futura del modelo. Comparando los valores obtenidos de la predicción junto a los valores reales del subconjunto de prueba, se dará lugar a los errores que comete el modelo durante dicho proceso. Mediante una evaluación analítica y gráfica de los resultados obtenidos, se determinará si el ajuste del modelo es adecuado o, por el contrario, debemos redefinir ciertos parámetros de nuestro modelo de forma que se consiga un porcentaje de error adecuado al proceso que se está tratando.

5.2. Análisis preliminar

En primer lugar, se decide desde qué puntos de vista se va a afrontar el problema y qué análisis se van a realizar. Estas decisiones determinarán la dirección que se debe seguir durante el desarrollo del proyecto.

El caso práctico, como se ha comentado anteriormente, se trata de un edificio público destinado principalmente a la cátedra de alumnos. Por este motivo, será relevante contemplar dos escenarios: el consumo de la semana completa y el consumo de los días lectivos, es decir, lunes a viernes.

Dado que la mayor parte de la demanda de energía eléctrica del edificio se concentra en los días que los alumnos acuden a la escuela, los valores registrados en días no lectivos, es decir, festivos y fines de semana, pueden suponer una desviación para el ajuste del modelo al ser tratados como valores atípicos.

Por otro lado, para tener en cuenta el set de datos completo, se le otorgará mayor información al modelo sobre los datos “problemáticos” de esta situación. Para ello, se crea una columna que incluya el día de la semana de cada registro para que así detecte el patrón de apertura de la escuela de cada semana. Además, mediante el uso de una función y la librería Holidays de Python, se crea una columna que muestra al modelo si el día es festivo o no. De esta forma, se prepara al modelo para contrarrestar la aparición de datos atípicos.

5.3. Procesamiento de datos

Para comenzar, cargamos y leemos el set de datos para conocer la información que contiene. Se hace uso de la función head () de la librería pandas para facilitar la visualización del conjunto de datos, ya que esta nos proporciona un resultado con un estilo gráfico tabulado más intuitivo que la clásica representación de código en consola.

```
#----------------CARGAMOS Y LEEMOS EL DATASET---------------
datos_original=pd.read_excel('/content/datosescuela.xls',header=0)
```

Figura 9. Carga y lectura del set de datos.

	Identificacion	CUPS	Fecha	Hora	Energía Activa Compra A+(kWh)	Energía Inductiva Compra Ri+(kvarh)	Energía Capacitiva Compra Rc-(kvarh)
0	US. E. U. Politécnica (Vgen Africa)	ES0031104000391001AT0F_	2017-11-01	00:15	8	1	0
1	US. E. U. Politécnica (Vgen Africa)	ES0031104000391001AT0F_	2017-11-01	00:30	9	0	0
2	US. E. U. Politécnica (Vgen Africa)	ES0031104000391001AT0F_	2017-11-01	00:45	8	0	0
3	US. E. U. Politécnica (Vgen Africa)	ES0031104000391001AT0F_	2017-11-01	01:00	8	0	0
4	US. E. U. Politécnica (Vgen Africa)	ES0031104000391001AT0F_	2017-11-01	01:15	9	0	0

Figura 10. Muestra de datos originales.

Además, con la función describe () se muestran estadísticas básicas del conjunto de datos. De esta forma, se obtiene un primer resumen sobre los datos que se van a estudiar: valor máximo y mínimo, valor medio, número de registros, percentiles, etc.

	Energía Activa Compra A+(kWh)
count	35040.000000
mean	18.309168
std	14.447904
min	5.000000
25%	8.000000
50%	10.000000
75%	30.000000
max	80.000000

Figura 11. Estadísticas del set de datos.

Examinando una pequeña muestra de los datos originales se observan los primeros pasos a implementar: eliminar las columnas que no van a ser utilizadas en el modelo, crear las columnas necesarias para dotar a la red de mayor información relevante en el proyecto y modificar las columnas que sean útiles, pero no se encuentran en el formato adecuado para su análisis.

En primer lugar, se crea una columna que indica los días de la semana de los registros del set de datos. Esta se transforma mediante el uso de un diccionario, el cual asigna un número a cada día de la semana. Como resultado se obtiene una columna la cual refleja los días de la semana de forma numérica, por ejemplo, el valor cero es el lunes, el uno corresponde al martes y así consecutivamente.

```
#------------AÑADO COLUMNA DIAS DE LA SEMANA-------------
datos_original['Día de la Semana'] = datos_original['Fecha'].dt.day_name()
mapeo = {'Monday': 0,'Tuesday': 1,'Wednesday': 2,'Thursday': 3,'Friday': 4,'Saturday': 5,'Sunday': 6}
datos_original['Día Semana Numérico'] = datos_original['Día de la Semana'].map(mapeo)   # Aplica el mapeo
```

Figura 12. Creación días de la semana.

A continuación, se define una función para corregir los valores atípicos del set de datos. Estos valores están relacionados con fallos eléctricos y, por tanto, cortes del suministro del edificio. Para la creación de dicha función, se supone que el tiempo entre el fallo eléctrico y la vuelta al funcionamiento normal no supera una hora de reloj. Por otra parte, se considera como límite de fallo, una caída en el consumo eléctrico igual o mayor a nueve kilo-vatios hora.

```
#------------ELIMINAMOS VALORES ATÍPICOS----------------
def busca_fallos(datos, columna, limite):
    nuevo_df = datos.copy()
    for i in range(1, len(datos)):
        diferencia = abs(nuevo_df.at[i,columna] - nuevo_df.at[i-1,columna])
        if diferencia >= limite:
          valor_anterior=nuevo_df.at[i-5,columna]
          valor_posterior=nuevo_df.at[i+5,columna]
          nuevo_valor=(valor_anterior+valor_posterior)/2
          nuevo_df.at[i,columna]=nuevo_valor
    return nuevo_df
data = busca_fallos(datos_original,'Energía Activa Compra A+(kWh)',9)
len(datos_original)
```

Figura 13. Función para corregir fallos eléctricos.

Por otro lado, se procede a implementar cuatro funciones para los períodos no lectivos de la Universidad de Sevilla: navidad, semana santa, feria y vacaciones de verano. Dichas funciones guardan en una lista todas las fechas que componen estos períodos y se aseguran de que los datos proporcionados sean en el formato correcto para que el modelo sea capaz de interpretarlos. Las fechas se han extraído del calendario académico de la universidad [25].

```
#---------------FUNCIONES PARA INTRODUCIR LOS FESTIVOS----------------
def navidad ():
  inicio = input('Fecha inicio navidad (formato YYYY-MM-DD): ') # Pedir al usuario fecha de inicio
  try:
    inicio = datetime.strptime(inicio, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  fin = input('Fecha fin navidad (formato YYYY-MM-DD): ')  # Pedir al usuario fecha de finalización
  try:
    fin = datetime.strptime(fin, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  if fin < inicio:  # Verificar si la fecha de finalización es posterior a la fecha de inicio
    print('La fecha de finalización debe ser posterior a la fecha de inicio. Vuelve a ejecutar el programa.')
    exit()
  lista_de_fechas = []
  fecha = inicio
  while fecha <= fin:
    lista_de_fechas.append(fecha)
    fecha += timedelta(days=1)
  return lista_de_fechas
```

Figura 14. Función para períodos no lectivos.

Además, mediante el uso de la librería “Holidays” [26], se crean listas con los días festivos en Andalucía y festivos nacionales. Agrupando dichas listas con los resultados de las funciones anteriormente definidas, se completa la información necesaria para crear una columna que refleje si el día de cada registro es festivo.

```
def es_festivo(fecha):
    return fecha in festivos_and

def es_no_lectivo(fecha):
    return fecha in periodo_no_lectivo


festivos_and_2017=holidays.ES(prov='AN',years=2017, language="es")
festivos_and_2018=holidays.ES(prov='AN',years=2018, language="es")
festivos_and=festivos_and_2017+festivos_and_2018

periodo_navidad=navidad()
periodo_ss=ss()
periodo_feria=feria()
periodo_vac_verano=vac_verano()
periodo_no_lectivo=periodo_navidad+periodo_ss+periodo_feria+periodo_vac_verano

data['Festivo'] = data['Fecha'].apply(es_festivo)
data['No lectivo'] = data['Fecha'].apply(es_no_lectivo)
```

Figura 15. Creación de columna "festivo".

Para continuar, se transforman los datos de fecha y hora de cada registro para utilizarlos como valores de entrada al modelo. Se crean columnas independientes para el día, el mes y la hora de cada registro en formato numérico. Además, los minutos se pasan a horas para ahorrar una columna en nuestro set de datos. De esta forma, se consigue que la librería TensorFlow admita estos valores como variables de entrada, ya que no es capaz de leer datos en formato “timestamp”.

```
#-------------TRANSFORMACIÓN DATOS FECHA Y HORA-------------------
data['Hora']=pd.to_datetime(data['Hora'], format='%H:%M') #convierte columna hora en datetime
data['Fecha y hora'] = data['Fecha'] +pd.to_timedelta(data['Hora'].dt.strftime('%H:%M:%S'))  #une columna de fecha y hora
data['Día'] = data['Fecha y hora'].dt.day
data['Mes'] = data['Fecha y hora'].dt.month
data['hora'] = data['Fecha y hora'].dt.hour
data['hora'] = data['hora'].astype(float)
data['minutos_en_horas'] = data['Fecha y hora'].dt.minute.astype(float) / 60.0
data['hora'] = data['hora'] + data['minutos_en_horas']
```

Figura 16. Transformación de fecha y hora.

Para finalizar, se crea una variable en la que se guardan los valores del set de datos que corresponden a los días entre semana, es decir, de lunes a viernes. Dicha variable también excluye los días festivos y períodos no lectivos, para así conseguir realizar una comparación entre el período lectivo y el set de datos completo. Al seleccionar ciertos valores del dataframe y guardarlos en una variable, se debe resetear el índice del mismo para reordenar los datos que se han elegido.

```
#------------DATOS ENTRE SEMANA---------------
datos_open = data[~data['Fecha'].dt.day_name().isin(['Saturday', 'Sunday'])]
datos_apertura = datos_open[~datos_open['Festivo'].isin([True])]
datos_apertura.reset_index(drop=True, inplace=True)
```

Figura 17. Creación de datos entre semana.

5.4. Análisis gráfico

Una vez preprocesados los datos, se puede comenzar con el análisis gráfico de la variable a estudiar. De esta forma, conseguimos que los datos, patrones, simetrías y demás conceptos que se muestran en las distintas representaciones gráficas, resulten ser los valores definitivos con los que va a trabajar el modelo.

5.4.1. Análisis diario

Para comenzar, se analizan los datos de demanda de energía eléctrica diarios. Se seleccionan dos días al azar de todo el conjunto del set de datos y se representan los valores de demanda correspondientes a cada hora del día para compararlos.

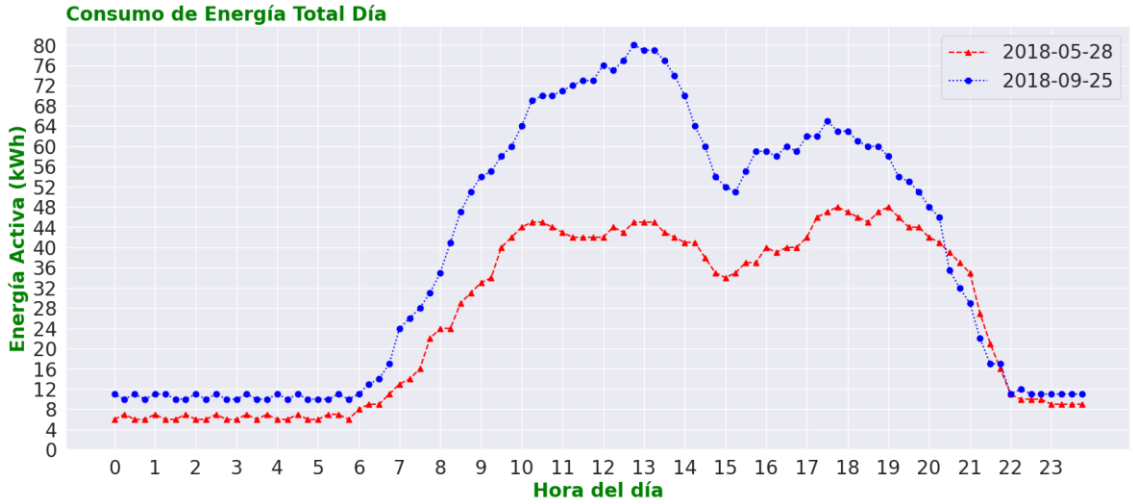

Figura 18. Gráfico de consumo diario.

A simple vista, se puede apreciar que las curvas de consumo de ambos días tienen la misma forma, a pesar de ser dos fechas muy separadas en el tiempo. Se debe a que los hábitos de consumo del edificio resultan ser muy similares durante el curso académico, aunque también se puede observar que los niveles de consumo sí que varían según el día. Este último hecho era de esperar ya que, aun siguiendo las mismas tendencias durante el día, el consumo eléctrico está condicionado por parámetros como, por ejemplo, la estación del año, la proximidad de períodos festivos o las semanas de exámenes.

Por otro lado, se puede destacar tanto el crecimiento de la demanda en las primeras horas de apertura del edificio como el descenso de la misma en las horas de cierre.

Por último, se distinguen distintas franjas horarias respecto a los niveles y tendencias que alcanzan los valores de energía. Con ello, se considera la hora del día como una variable relevante en la evolución de la demanda eléctrica y, por tanto, un valor importante para introducir al modelo como entrada.

5.4.2. Análisis semanal

A continuación, se representan los valores de demanda eléctrica que se han registrado durante una semana cualquiera del conjunto de datos. Se puede observar que la curva adopta un patrón muy similar los días lectivos de la semana, y al llegar el fin de semana, cambia. Por este motivo, resulta relevante estudiar los dos escenarios propuestos anteriormente: set de datos completo y set de datos en días lectivos.

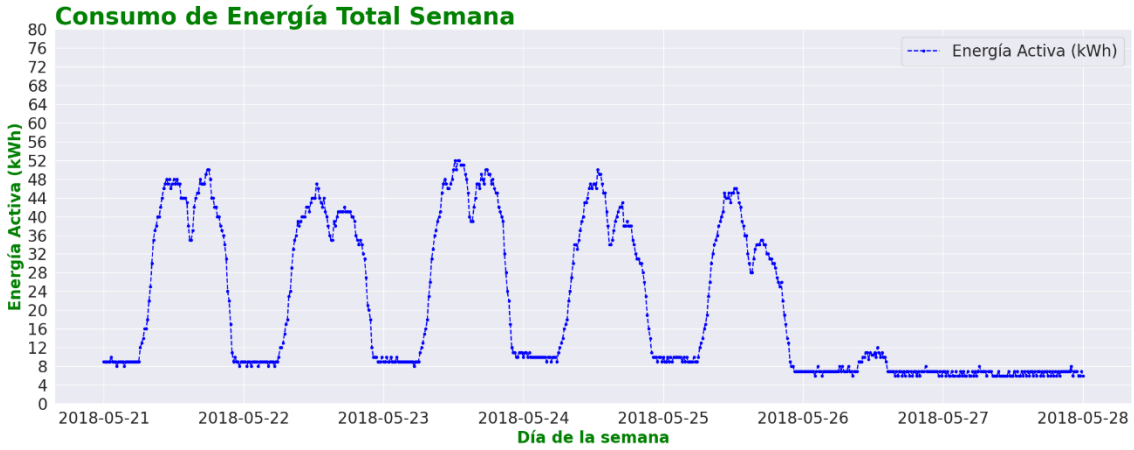

Figura 19. Gráfico de consumo semanal.

De igual forma, se puede distinguir un consumo menor en ciertos días lectivos, por lo que los días de la semana al que corresponde cada registro, resultan ser una variable relevante en el proyecto.

5.4.3. Análisis mensual

En este apartado, se interpretan datos de consumo eléctrico registrados durante un mes completo, por ejemplo, marzo. En el gráfico destaca principalmente el patrón que adoptan las semanas del mes, el cual se repite de forma muy similar en todas ellas. Dada la similitud de consumo entre las distintas semanas, no será destacado adoptar como variable de entrada las semanas del mes, por tanto, dicho concepto no será tratado en el proyecto.

Por otra parte, se aprecia una perturbación en el patrón de consumo a final de mes. Este acontecimiento se debe a que en dicho período transcurre la semana santa, la cual sus días son no lectivos.

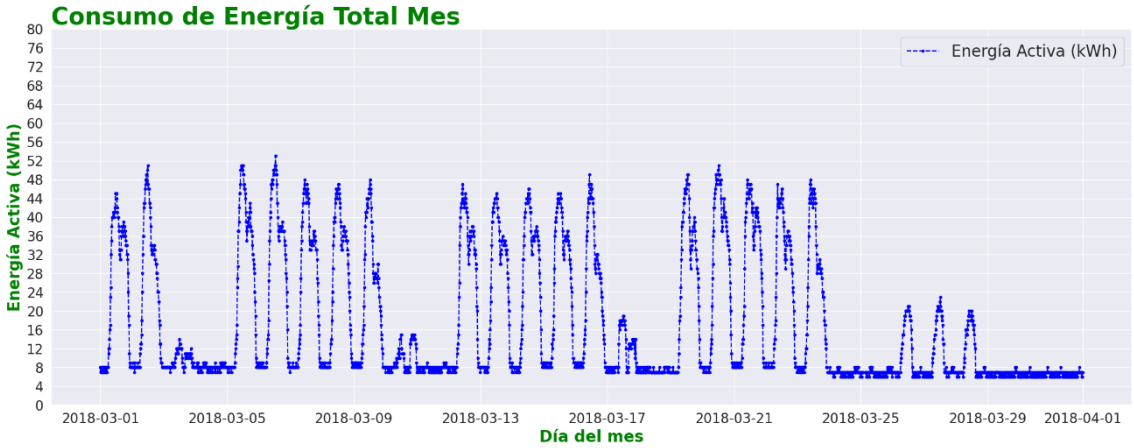

Figura 20. Gráfico de consumo mensual.

5.4.4. Análisis anual

Para finalizar el análisis gráfico, se figuran los datos de todo el año, que corresponden al set de datos completo. Esta imagen describe los acontecimientos que ocurren en el problema en su totalidad como, por ejemplo, las caídas del consumo en los períodos no lectivos, la tendencia decreciente debido a la proximidad de las vacaciones de verano o el notable incremento de la demanda eléctrica al inicio del curso escolar.

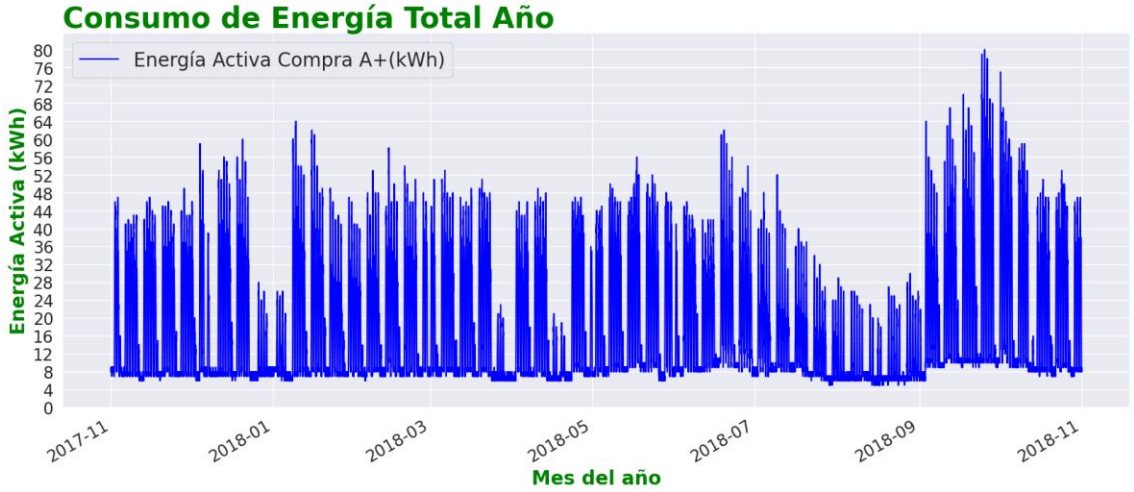

Figura 21. Gráfico de consumo anual.

La notable diferencia en los valores de consumo entre distintos meses del proyecto, convierte al mes del año en una variable a tener en cuenta como entrada del modelo.

5.5. Creación de los modelos

Una vez preparados los datos y realizado el análisis gráfico para comprender el contexto del problema, se comienza con la creación de los modelos elegidos. Se van a crear dos modelos basados en redes neuronales. El primero de ellos, es un perceptrón multicapa y el segundo una red neuronal recurrente.

La variable que se va a estudiar se trata de una serie de tiempo, es decir, los registros del set de datos corresponden a un valor que discurre y evoluciona en el tiempo, el cual se encuentra monitorizado y se mide con una cierta frecuencia.

Durante la revisión de artículos y estudios en los que se trabaja con series de tiempo, se observa que, para facilitar el aprendizaje del modelo, tratan de garantizar la continuidad de las variables de entrada. Esto se consigue, por ejemplo, transformando las variables de fecha y hora de forma que corresponda a una serie continua de valores o pasando los valores a radianes y trabajando con funciones seno y coseno. De igual forma, al fragmentar los datos en subconjuntos, se establecen paquetes de registros consecutivos.

Sin embargo, se decide que este proyecto ponga a prueba la tecnología del Machine Learning de una forma diferente a la mayoría de antecedentes. Se pretende trabajar con la gran capacidad de adaptación de las redes neuronales a distintos sets de datos, de forma que el modelo sea efectivo si recibe como entrada un conjunto de datos de características diferentes al original.

En primer lugar, las variables de fecha y hora contienen su propio valor numérico, es decir, tendrá una cierta estacionalidad, pero introduce perturbaciones al no suavizar la evolución de las variables. Por ejemplo, los valores de la hora avanzan desde cero hasta veintitrés y dicho ciclo se repite cada día, por lo tanto, en el cambio de un día al siguiente la variable hora se encuentra con un salto desde su valor máximo hasta su mínimo, lo cual en principio no favorece al modelo.

Conjuntamente, al fragmentar los paquetes de datos, se establece que la selección de datos de cada subconjunto sea aleatoria. Por ejemplo, si el total de datos es cien y el subconjunto de entrenamiento contiene el ochenta por ciento de los registros, estos no serán los primeros ochenta valores, sino que serán valores al azar de todo el conjunto. De esta forma, el modelo trabajará para encontrar patrones entre valores sin ordenar cronológicamente.

Cabe destacar que, los modelos detallados a continuación, poseen la topología más adecuada para el proyecto, es decir, con la que mejor resultado se ha obtenido.

Dicha conclusión se obtiene tras realizar combinaciones con diferentes valores para los distintos parámetros del modelo. Junto a cada modificación de un parámetro, se realiza el entrenamiento del modelo y se analizan los resultados para comprender como afecta la modificación de cada parámetro al desempeño del modelo. En el anexo se incluirán las pruebas realizadas durante el proyecto.

5.5.1. Modelo de capas densas

Para comenzar, se crea el modelo perceptrón multicapa. El apodo de dicho modelo corresponde a que está formado por capas densas o fully connected, es decir, todas las neuronas de cada capa densa están conectadas a todas las neuronas de la capa siguiente.

```
#------------CREAMOS MODELOS--------------
def modelo_0():
  model=tf.keras.Sequential([
      tf.keras.layers.Input(shape=n_entradas),
      tf.keras.layers.Dense(256,activation='relu'),
      tf.keras.layers.Dense(256,activation='relu'),
      tf.keras.layers.Dense(128,activation='relu'),
      tf.keras.layers.Dense(128,activation='relu'),
      tf.keras.layers.Dense(64,activation='relu'),
      tf.keras.layers.Dense(n_predicciones)
  ])
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)
  model.compile(loss='mse',optimizer=optimizer,metrics=['mae','mse','mape'])
  return model
```

Figura 22. Modelo de capas densas.

El modelo cero o modelo denso, está basado en el modelo secuencial de Keras que trabaja como contenedor con un conjunto lineal de capas. Está formado por una capa de entrada con tantas neuronas como número de variables de entrada, una capa de salida con una neurona ya que realizará una predicción y cinco capas ocultas de tipo denso con número de neuronas decreciente y unidad lineal rectificada (relu) como función de activación.

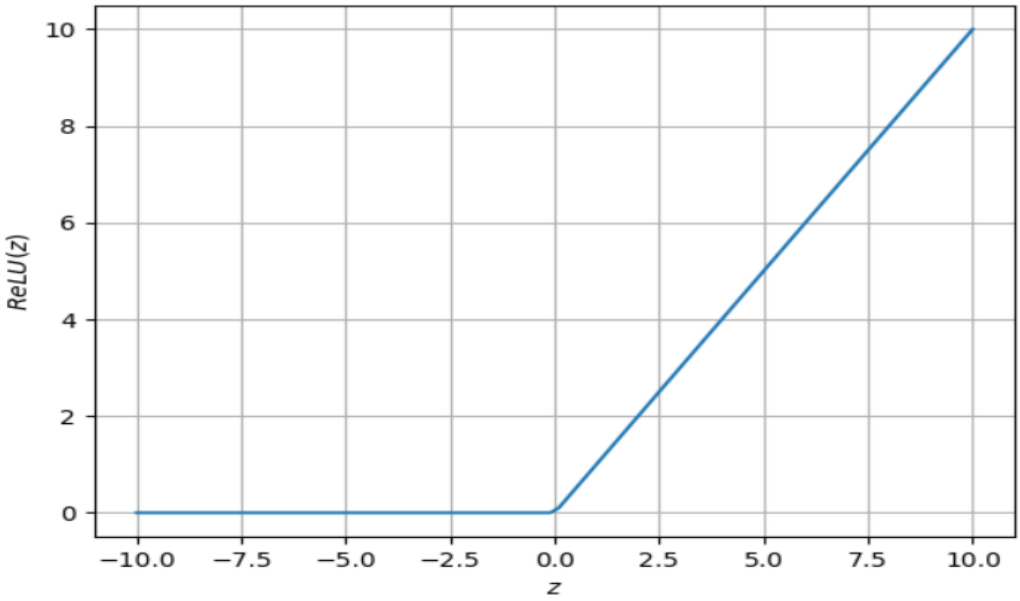

Figura 23. Función de activación "ReLU" [27].

Esta función “generará una salida igual a cero cuando la entrada (z) sea negativa, y una salida igual a la entrada cuando dicha esta última sea positiva. Además, la no existencia de saturación, hace que el algoritmo converja mucho más rápidamente, facilitando así el entrenamiento” [27].

A continuación, se define el optimizador Adam con una tasa de aprendizaje de 1 milésima. Dicho optimizador “reúne las ventajas de otros dos como son el gradiente descendente que, usa la media móvil para evitar oscilaciones excesivas en los valores de los gradientes durante el entrenamiento y el algoritmo RMSPROP, el cual incorpora un tamaño de paso variable, que permite acelerar o desacelerar el proceso de entrenamiento dependiendo de la magnitud del gradiente en cada iteración” [28].

Para finalizar con la creación del modelo, se define el valor de pérdida y las métricas de evaluación del modelo. Se define el error cuadrático medio como valor “loss”, con el cual se prevalece minimizar errores grandes en lugar de pequeñas diferencias. Además, para evaluar el ajuste del modelo en cada iteración se determinan el error absoluto medio y el error porcentual absoluto.

5.5.2. Modelo de capas LSTM

Para el modelo de red neuronal recurrente se crea un modelo formado por capas con términos de memoria a corto y largo plazo, también conocidas como Long Short Term Memory (LSTM). Como su nombre indica, las neuronas que forman estas capas tienen la capacidad de recordar valores anteriores para así detectar patrones en datos consecutivos.

Por este motivo, este tipo de modelos se adaptan muy bien a las series de tiempo, ya que en cada iteración analiza la variable a partir de la evolución que sigue dicho valor. En contraposición, este estudio trabaja con datos aleatorios cronológicamente, por tanto, no se utiliza esta característica como en los antecedentes valorados.

```
def modelo_1():
  model=tf.keras.Sequential([
      tf.keras.layers.Input(shape=(n_pasos_de_tiempo,n_entradas)),
      tf.keras.layers.LSTM(128,activation='tanh',return_sequences=True),
      tf.keras.layers.LSTM(64,activation='tanh',return_sequences=True),
      tf.keras.layers.LSTM(32,activation='tanh'),
      tf.keras.layers.Dense(n_predicciones)
  ])
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)
  model.compile(loss='mse',optimizer=optimizer,metrics=['mae','mse','mape'])
  return model
```

Figura 24. Modelo de capas LSTM.

El modelo uno o modelo LSTM guarda ciertas similitudes con respecto al modelo anterior. De esta forma, el contenedor, el optimizador, la variable de pérdida y las métricas de evaluación de la calidad del ajuste corresponden a los conceptos ya explicados anteriormente. Asimismo, las capas de entrada y salida del modelo están configuradas con la misma metodología que el modelo denso.

La diferencia se encuentra en las capas ocultas, las cuales son 3 capas LSTM con número de neuronas decreciente y función de activación tangente hiperbólica (tanh).

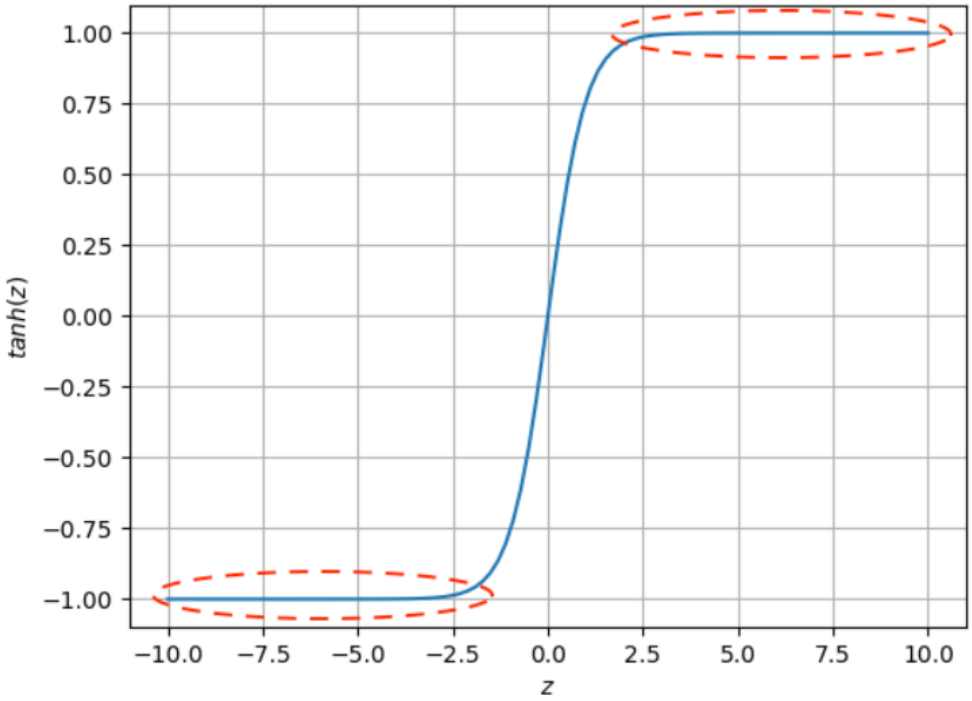

Figura 25. Función de activación "tanh".

Esta función “tiene un comportamiento muy similar a la sigmoidal, con la diferencia de que los valores de salida estarán en el rango de más menos uno. También sufre del problema de saturación, pero ofrece la ventaja de tener una salida simétrica lo cual facilita el proceso de entrenamiento” [27].

5.6. Entrenamiento y validación

En primer lugar, se realiza la fragmentación del set de datos en los distintos subconjuntos: entrenamiento, validación y prueba. Para ello, se hace uso de la librería Scikit-Learn que, mediante la función train_test_split(), recoge dicho proceso de forma simplificada. Este proceso se realiza dos veces, la primera define los datos de prueba, y la segunda divide entrenamiento y validación. Además, se transforman los valores de entrada a una escala entre cero y uno con el escalador MaxAbsScaler() para facilitar la convergencia del proceso.

```
#-----------SEPARAMOS CONJUNTOS DE DATOS--------------
x1=df.iloc[:,1:6].values
y1=df.iloc[:,0].values

x_rest,x_test,y_rest,y_test=sk.train_test_split(x1,y1,test_size=0.075, random_state=12)

x2=x_rest
y2=y_rest

x_train,x_valid,y_train,y_valid=sk.train_test_split(x2,y2,test_size=0.08108108, random_state=10)

#-------------ESCALAMOS DATOS PARA FACILITAR CONVERGENCIA DEL PROCESO-----------------
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
x_train=scaler.fit_transform(x_train)#AJUSTA LOS DATOS Y LUEGO LOS TRANSFORMA
x_valid=scaler.transform(x_valid)
x_test=scaler.transform(x_test)  #ESTANDARIZA CENTRÁNDOSE Y ESCALANDO
```

Figura 26. Fragmentación y escalado de datos.

Por otro lado, para el modelo LSTM se deben modificar los subconjuntos de datos ya que dicho modelo trabaja con conjuntos de datos de tres dimensiones. Se define la variable n_pasos_de_tiempo que indica el número de registros que va a contener la variable de entrada en cada iteración. Conjuntamente, se definen los conjuntos de datos tridimensionales con el sufijo _3d. Los dos parámetros restantes que definen dichos conjuntos son: el número de muestras y el número de variables de entrada. Por último, se escalan los datos de entrada y se establece, en los distintos subconjuntos, un mismo número de muestras para la variable de salida, para que el modelo realice un funcionamiento correcto.

```
n_pasos_de_tiempo = 4
n_muestras_train = len(x_train) - n_pasos_de_tiempo + 1
x_train_3d = np.zeros((n_muestras_train, n_pasos_de_tiempo, 5))
n_muestras_valid = len(x_valid) - n_pasos_de_tiempo + 1
x_valid_3d = np.zeros((n_muestras_valid, n_pasos_de_tiempo, 5))
n_muestras_test = len(x_test) - n_pasos_de_tiempo + 1
x_test_3d = np.zeros((n_muestras_test, n_pasos_de_tiempo, 5))

# Rellena el tensor con datos
for i in range(n_muestras_train):
    x_train_3d[i] = x_train[i:i+n_pasos_de_tiempo]
for i in range(n_muestras_valid):
    x_valid_3d[i] = x_valid[i:i+n_pasos_de_tiempo]
for i in range(n_muestras_test):
    x_test_3d[i] = x_test[i:i+n_pasos_de_tiempo]

y_train_3d = y_train[:n_muestras_train] #Asegura que y_train tenga el mismo número de muestras
y_valid_3d = y_valid[:n_muestras_valid]
y_test_3d = y_test[:n_muestras_test]

x_train_3d = scaler.fit_transform(x_train_3d.reshape(-1, x_train_3d.shape[-1])).reshape(x_train_3d.shape)
x_valid_3d = scaler.fit_transform(x_valid_3d.reshape(-1, x_valid_3d.shape[-1])).reshape(x_valid_3d.shape)
x_test_3d = scaler.transform(x_test_3d.reshape(-1, x_test_3d.shape[-1])).reshape(x_test_3d.shape)
```

Figura 27. Transformación de datos a 3D.

Cabe señalar que, tanto el fragmentado y escalado preliminar como la posterior transformación a datos tridimensionales, se realiza para los dos escenarios propuestos: set de datos completo y días lectivos. Para diferenciar ambas propuestas, se coloca el sufijo _open a las variables definidas para los días lectivos, haciendo referencia a la apertura de la escuela. El set de datos completo se divide en 85% entrenamiento y 15% validación y prueba. Para los días lectivos, se establece un reparto 70/30 al disponer de una menor cantidad de registros.

A continuación, se entrenan ambos modelos con los sets de datos de las dos propuestas. La única diferencia a tener en cuenta es el número de variables de entrada que, en la situación del set de datos completos, aumenta su valor una unidad ya que se añade la columna de festivos. Además, para el caso de sólo días lectivos, se disminuye el batch size ya que se trata de una cantidad de datos menor.

```
#--------------ENTRENAMOS LOS MODELOS----------------
print('Comenzando entrenamiento... ')
n_entradas = 5
n_predicciones = 1
llamada_0=modelo_0()
entrenamiento_0=llamada_0.fit(x_train,y_train,epochs=100,batch_size=100, verbose=True,validation_data=(x_valid,y_valid))
print('Entrenamiento completado')
validation_results = llamada_0.evaluate(x_valid, y_valid)
print("Pérdida en el conjunto de validación:", validation_results[0])
print("Precisión en el conjunto de validación:", validation_results[1])
test_results = llamada_0.evaluate(x_test, y_test)
print("Pérdida en el conjunto de prueba:", test_results[0])
print("Precisión en el conjunto de prueba:", test_results[1])
```

Figura 28. Definición del entrenamiento.

Se establece el parámetro verbose=True para mostrar en cada iteración del entrenamiento, las métricas de evaluación de la calidad del ajuste del modelo. Estas serán referidas al desempeño de los datos de entrenamiento y validación.

```
Comenzando entrenamiento...
Epoch 1/100
298/298 [==============================] - 4s 9ms/step - loss: 140.4289 - mae: 8.3540 - mse: 140.4289 - mape: 56.5707 - val_loss: 34.2981 - val_mae: 3.7643
Epoch 2/100
298/298 [==============================] - 2s 8ms/step - loss: 29.2912 - mae: 3.4298 - mse: 29.2912 - mape: 20.6282 - val_loss: 26.6225 - val_mae: 3.0658 -
Epoch 3/100
298/298 [==============================] - 2s 6ms/step - loss: 24.9084 - mae: 3.0583 - mse: 24.9084 - mape: 17.7971 - val_loss: 22.4341 - val_mae: 2.8862 -
```

Figura 29. Muestra del desarrollo del entrenamiento.

Las variables validation_results y test_results muestran los resultados de la evaluación del modelo con los datos de validación y prueba respectivamente, al finalizar el entrenamiento.

```
Entrenamiento completado
83/83 [==============================] - 0s 2ms/step - loss: 1.7450 - mae: 0.9072 - mse: 1.7450 - mape: 5.8795
Pérdida en el conjunto de validación: 1.7450419664382935
Precisión en el conjunto de validación: 0.9071894288063049
83/83 [==============================] - 0s 2ms/step - loss: 1.8231 - mae: 0.9243 - mse: 1.8231 - mape: 6.0710
Pérdida en el conjunto de prueba: 1.8231145143508911
Precisión en el conjunto de prueba: 0.9243074059486389
```

Figura 30. Métricas de error post-entrenamiento.

Además, se crea un gráfico donde se representa la evolución del valor de pérdida para analizar la convergencia del mismo en el subconjunto de entrenamiento. Igualmente, se representa dicho valor para los datos de validación de forma que, se visualiza el comportamiento del modelo en las evaluaciones y se analiza la posibilidad de que existan problemas como los ya mencionados overfitting o underfitting.

5.7. Predicción y cálculo de errores

Una vez entrenados los modelos, se realizan predicciones sobre los datos de prueba y se comparan con los valores reales para evaluar la calidad del ajuste del modelo.

```
#------------------PREDICCION-------------------
predicciones_0 = llamada_0.predict(x_test,batch_size=100)
```

Figura 31. Creación de predicciones.

Se crea una función para evaluar el rendimiento del modelo, la cual devuelve como resultado, métricas de error que se obtienen al comparar las predicciones con los valores reales.

```
#---------------FUNCION DE ACIERTO EN LAS PREDICCIONES----------------------
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error
def evaluar_rendimiento(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true,y_pred,squared=False)  #false para que devuelva el rmse en lugar de mse
    mape = mean_absolute_percentage_error(y_true,y_pred)
    rendimiento = {
        'MSE': mse,
        'MAE': mae,
        'RMSE': rmse,
        'MAPE': mape,
    }
    return rendimiento
rendimiento_0_full= evaluar_rendimiento(y_test, predicciones_0)
```

Figura 32. Función para evaluar el rendimiento del modelo.

Por otra parte, se crea una función para calcular los errores resultantes en cada predicción y almacenar dichos valores en un vector. Posteriormente, este se utiliza para analizar el error obtenido mediante el uso de percentiles y otras métricas como, por ejemplo, el error máximo.

```
#---------------FUNCION PARA CALCULAR EL ERROR ABSOLUTO------------
def error(y_tr, y_pr,size):
    error=y_tr-y_pr.reshape(1,size)
    error_abs=np.abs(error)
    return error_abs

error_0_full = error(y_test, predicciones_0,test_size)
error_0_full=error_0_full.flatten()
error_max_0_full=np.max(error_0_full)  #devuelve el valor máximo
```

Figura 33. Función para calcular el error en la predicción.

Para finalizar el análisis de errores, se muestran todas las métricas en una tabla para comparar el desempeño de ambos modelos con datos de los dos escenarios propuestos. Además, para obtener un resultado más intuitivo visualmente, se crea un gráfico donde se representan los valores reales de prueba y las predicciones que se realizan de los mismos.

5.8. Resultados

5.8.1. Valor de pérdida

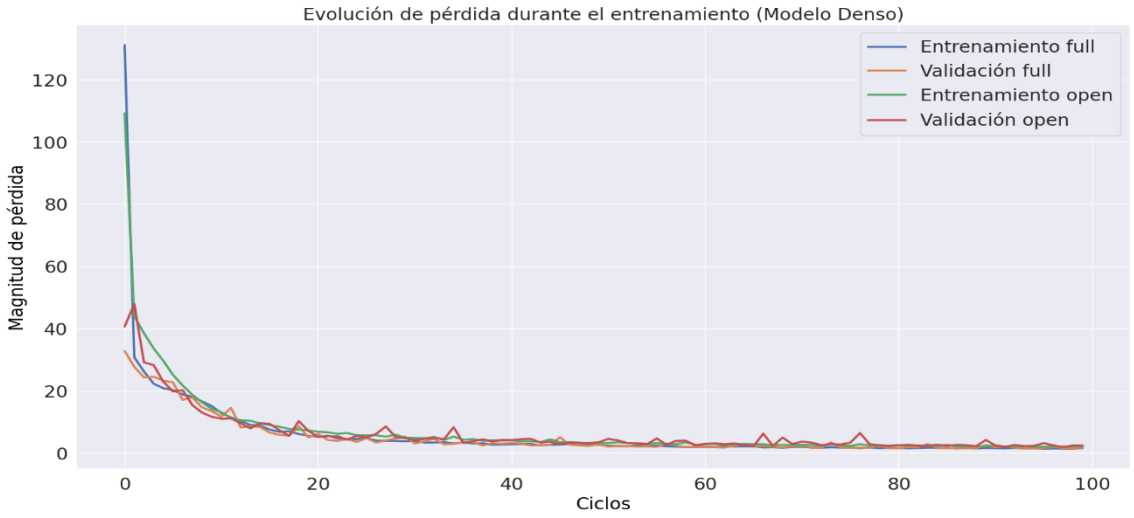

Figura 35. Gráfico del valor de pérdida en modelo denso.

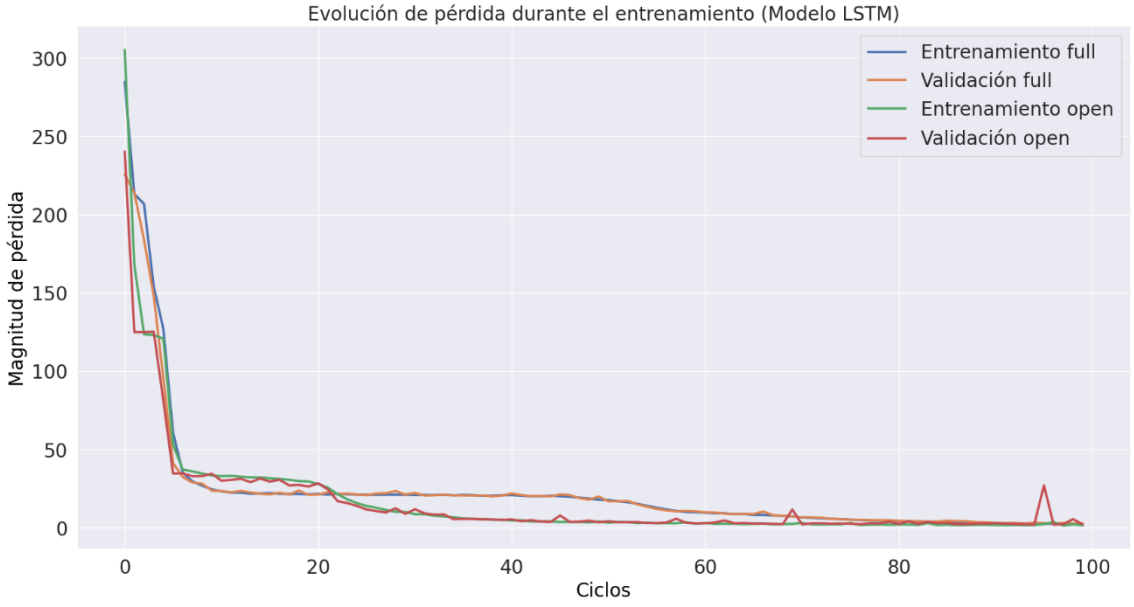

Figura 34. Gráfico del valor de pérdida en modelo LSTM.

Se observa que, en todos los casos el valor de pérdida, que corresponde al error cuadrático medio, converge de forma correcta minimizando el error del modelo y mejorando el grado de ajuste del mismo. También, destacar que, los valores correspondientes a los datos de validación, evolucionan de igual forma que la curva referente a los datos de entrenamiento. Este suceso indica que no existen problemas de overfitting ni underfitting durante la evaluación del modelo y que el ajuste es correcto.

5.8.2. Error en la predicción

Análisis de errores en datos de prueba

Métricas	Denso full	Denso open
MSE:	1.8231141754275715	2.5977043142792096
MAE:	0.9243073869876485	1.1368217251811248
RMSE:	1.3502274532194831	1.6117395305319062
MAPE(%):	6.070989016712045	5.4027870702905085
Error_25:	0.28261613845825195	0.3198509216308594
Error_50:	0.6048760414123535	0.7841997146606445
Error_75:	1.163583755493164	1.5491976737976074
Error_95	3.077468490600587	3.413072681427001
Error_max:	8.093441009521484	9.377023696899414

Figura 37. Tabla de errores en la predicción del modelo denso.

Análisis de errores en datos de prueba

Métricas	LSTM full	LSTM open
MSE:	2.4135641341395897	2.131472027764144
MAE:	1.0702444454374767	1.050661890981318
RMSE:	1.5535649758344805	1.459956173234027
MAPE(%):	7.232646003766622	5.132279288121382
Error_25:	0.3150215148925781	0.3321418762207031
Error_50:	0.6861896514892578	0.7509441375732422
Error_75:	1.4150657653808594	1.4304962158203125
Error_95	3.4671119689941357	3.118192291259765
Error_max:	7.400920867919922	8.48245620727539

Figura 36. Tabla de errores en la predicción del modelo LSTM.

Analizando los resultados obtenidos en los distintos escenarios y modelos se puede destacar que, en general, todos los casos estudiados consiguen un ajuste aceptable, siendo el error porcentual absoluto en todos ellos menor al 10%. Conjuntamente, el percentil 75 muestra que, en todas las situaciones, el 75 % de los errores absolutos están por debajo de aproximadamente 1,5 KWh.

Por otro lado, se observan valores menores en el error absoluto al utilizar el set de datos completo. A pesar de ello, se deduce que, el escenario de días lectivos, obtiene un mejor ajuste ya que el error porcentual absoluto es menor en ambos modelos.

Esta contradicción se debe a que se trabaja con dos sets de datos diferentes, por tanto, los valores reales a los que corresponden los errores reflejados en las tablas, cambian de un escenario a otro impidiendo hacer comparaciones con valores absolutos de forma directa.

5.8.3. Comparación de valor real y predicción

En los siguientes gráficos, se representan los primeros 50 valores reales del subconjunto de prueba, junto con los valores obtenidos en la predicción. En ambos escenarios, se figuran los mismos valores para comparar de forma directa la precisión de los dos modelos estudiados.

Figura 38. Gráfico valor real vs predicción en modelo denso y datos completos.

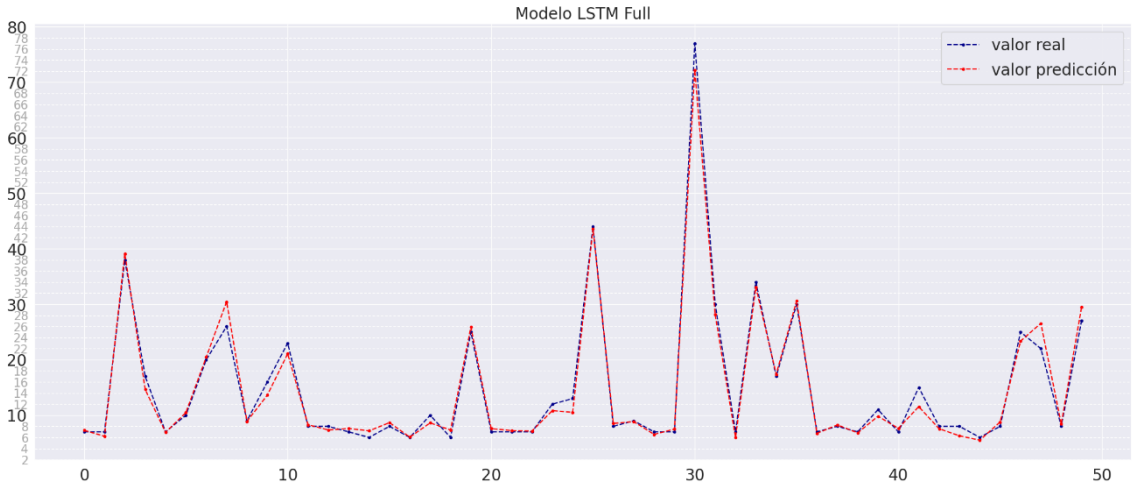

Figura 39. Gráfico valor real vs predicción en modelo LSTM y datos completos.

Figura 41. Gráfico valor real vs predicción en modelo denso y datos lectivos.

Figura 40. Gráfico valor real vs predicción en modelo LSTM y datos lectivos.

Cabe destacar como principal característica, el buen ajuste de los modelos a la hora de predecir datos de prueba, de forma que, la gráfica de valores estimados se asemeja con gran exactitud a la gráfica formada por los valores reales.

Por otra parte, se puede corroborar los resultados obtenidos en las tablas ya que, se aprecia que los errores absolutos de mayor valor son más frecuentes con el set de datos de días lectivos. Sin embargo, este caso alberga valores de consumo mayores en comparación con el set de datos completo en el que se hayan valores de demanda menores al incluir festivos y períodos no lectivos. Por ello, el error porcentual absoluto es menor y, por tanto, el ajuste del modelo es mejor.

6. CONCLUSIONES

6.1. Python

Este proyecto se comienza desde el desconocimiento sobre el lenguaje Python y finaliza con la implementación de un modelo basado en inteligencia artificial, concretamente en la parte de Machine Learning.

Después de dicho proceso, se destaca la facilidad que ofrece el lenguaje para su aprendizaje. Cuenta con una gran cantidad de contenido de calidad, aportado por la comunidad tanto de forma escrita como audiovisual además de las numerosas librerías que ahorran trabajo de codificación.

Por otra parte, la gran versatilidad que ofrece es uno de sus puntos fuertes. Este proyecto contiene codificación básica, gráficos personalizables, diversos procesos de ciencia de datos, operaciones matemáticas, estadística básica e incluso la implementación de un modelo de aprendizaje automático. La capacidad de realizar todas estas tareas tan diversas con un mismo lenguaje, ayudan a considerar Python como una herramienta fundamental en la actualidad.

6.2. Machine Learning

Tras un período breve de formación sobre aprendizaje automático, el cual comprende el desarrollo de este proyecto, resalta principalmente el potencial de desarrollo de dicha tecnología. Permite realizar estudios avanzados y extraer información de grandes cantidades de datos de forma sencilla.

Además, la capacidad de automatizar procesos complejos junto con la gran versatilidad de los modelos, hacen del Machine Learning una tecnología muy útil para analizar diversas cuestiones que afectan de manera global a la sociedad.

6.3. Ejemplo práctico

Analizando los resultados obtenidos durante el entrenamiento de los modelos y posteriormente con las predicciones realizadas, se concluye que:

- Los modelos implementados son sensibles a cambios bruscos entre valores consecutivos de demanda eléctrica. Se observa que el valor de error absoluto en una predicción aumenta cuando dicho registro se trata de un aumento o descenso notable en el consumo. Por tanto, el modelo aprende patrones de tendencias en los datos, sin adaptarse correctamente a los valores que resultan atípicos para el mismo.
- Los modelos de redes neuronales demuestran una gran capacidad para adaptarse a distintos sets de datos, consiguiendo en todas las situaciones propuestas un valor óptimo de calidad del ajuste.
- El mejor ajuste del proyecto se consigue con el modelo LSTM, el cual aumenta su rendimiento cuando se entrena con los datos de días lectivos. Este set de datos contiene una estacionalidad más definida, por tanto, el término de memoria de las capas LSTM resulta más eficiente en este caso.
- El modelo de capas densas muestra su gran versatilidad en ambos escenarios. Se resuelve que, dotando de mayor información al modelo, se mitigan los errores que podrían provocan ciertas perturbaciones en la estacionalidad de los datos. Es decir, a pesar de mantener valores atípicos de demanda como son los días no lectivos, otorgar información acerca de dichos valores ayuda a que el modelo consiga un ajuste óptimo.
- Se obtienen mejores resultados con el set de datos de días lectivos. Los modelos se adaptan mejor a dicho escenario al contener una distribución de valores más uniforme.

7. LÍNEAS FUTURAS

Para finalizar el proyecto, se recomiendan varias líneas futuras de investigación con las cuales se pueda mejorar el presente estudio o desarrollar nuevos proyectos:

- Durante las primeras etapas del proyecto, se entrenaron los modelos con los sets de datos originales. Se observó que, el valor máximo de error absoluto obtenido en las predicciones, correspondía a los registros que tenían valor cero. Por tanto, se sugiere implementar una función que agrupe los errores absolutos de mayor valor e identifique los valores reales de consumo que corresponden a dichos instantes. Sería una forma sencilla de encontrar perturbaciones eléctricas en una gran cantidad de datos de demanda para su posterior análisis.
- El set de datos original contiene datos de energía inductiva y capacitiva, los cuales no se han tenido en cuenta para el desarrollo del proyecto. Se propone desarrollar el estudio realizado, teniendo en cuenta dichos valores. De esta forma, se puede evaluar el factor de potencia del edificio para mejorar la eficiencia energética del mismo.

8. BIBLIOGRAFÍA

[1] «¿Qué es machine learning? | IBM». Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/machine-learning
[2] J. Delua, «Supervised vs. Unsupervised Learning: What's the Difference?», IBM Blog. Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://www.ibm.com/blog/supervised-vs-unsupervised-learning/
[3] elEconomista.es, «Big Data: qué es - Diccionario de Economía». Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://www.eleconomista.es/diccionario-de-economia/big-data
[4] «Big Data - Expertos en IIC», Instituto de Ingeniería del Conocimiento. Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://www.iic.uam.es/big-data/
[5] «¿Qué es Deep Learning? | IBM». Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://www.ibm.com/es-es/topics/deep-learning
[6] «¿Qué es una red neuronal?» Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://es.mathworks.com/discovery/neural-network.html
[7] J. Finance, «Tipos de redes neuronales (Clasificación)», Inteligencia-Artificial.dev. Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://inteligencia-artificial.dev/tipos-redes-neuronales/
[8] P. Royo, «Qué son las redes neuronales y cuál es su aplicación en el marketing», artyco | the data driven company. Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://artyco.com/que-son-las-redes-neuronales-y-cual-es-su-aplicacion-en-el-marketing/
[9] «IBM Documentation». Accedido: 2 de noviembre de 2023. [En línea]. Disponible en: https://www.ibm.com/docs/es/spss-modeler/saas?topic=nodes-statistical-models
[10] C. A. Y. Ramírez, «Pronóstico de consumo de energía eléctrica residencial de corto plazo utilizando algoritmos de aprendizaje automático y profundo», *Revista de investigación de Sistemas e Informática*, vol. 15, n.º 2, Art. n.º 2, dic. 2022, doi: 10.15381/risi.v15i2.23909.
[11] C. H. Menacho Chiok, «Comparación de los métodos de series de tiempo y redes neuronales», *Anales Científicos*, vol. 75, n.º 2, pp. 245-252, 2014.
[12] J. A. Mariño Villalba, «Una comparación entre modelos estadísticos y de Machine Learning para la predicción de series de tiempo multivariadas», Trabajo de grado - Maestría, Universidad Nacional de Colombia, 2023. Accedido: 11 de septiembre de 2023. [En línea]. Disponible en: https://repositorio.unal.edu.co/handle/unal/84522
[13] «Python: qué es y por qué deberías aprender a utilizarlo». Accedido: 26 de octubre de 2023. [En línea]. Disponible en: https://www.becas-santander.com/es/blog/python-que-es.html
[14] R. KeepCoding, «Ventajas y Desventajas de Python | KeepCoding Bootcamps». Accedido: 26 de octubre de 2023. [En línea]. Disponible en: https://keepcoding.io/blog/ventajas-y-desventajas-de-python/
[15] «pandas - Python Data Analysis Library». Accedido: 22 de octubre de 2023. [En línea]. Disponible en: https://pandas.pydata.org/about/
[16] «NumPy documentation — NumPy v1.26 Manual». Accedido: 21 de octubre de 2023. [En línea]. Disponible en: https://numpy.org/doc/stable/
[17] «matplotlib», Aprende Python. Accedido: 5 de octubre de 2023. [En línea]. Disponible en: https://aprendepython.es/pypi/datascience/matplotlib/

[18] M. Waskom, «seaborn: statistical data visualization», *JOSS*, vol. 6, n.º 60, p. 3021, abr. 2021, doi: 10.21105/joss.03021.
[19] «Introducción a TensorFlow», TensorFlow. Accedido: 21 de octubre de 2023. [En línea]. Disponible en: https://www.tensorflow.org/learn?hl=es-419
[20] «Keras: Deep Learning for humans». Accedido: 21 de octubre de 2023. [En línea]. Disponible en: https://keras.io/
[21] «scikit-learn: machine learning in Python — scikit-learn 1.3.1 documentation». Accedido: 21 de octubre de 2023. [En línea]. Disponible en: https://scikit-learn.org/stable/
[22] T. Tech, «¿Qué es la Función de Activación?», ¿Qué es la Función de Activación? Accedido: 26 de octubre de 2023. [En línea]. Disponible en: https://aiofthings.telefonicatech.com/recursos/datapedia/funcion-activacion
[23] «¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?», DataSmarts Español. Accedido: 26 de octubre de 2023. [En línea]. Disponible en: https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/
[24] «Underfitting y Overfitting en las Redes Neuronales», Codificando Bits. Accedido: 26 de octubre de 2023. [En línea]. Disponible en: https://www.codificandobits.com/blog/underfitting-y-overfitting/
[25] R. U. L. B. Sevilla, «Calendario Académico US 2017/2018. Residencia Universitaria La Buhaira», Residencia de Estudiantes La Buhaira. Accedido: 10 de octubre de 2023. [En línea]. Disponible en: https://residenciauniversitarialabuhaira.com/calendario-academico-us-20172018/
[26] «python-holidays — holidays documentation». Accedido: 6 de octubre de 2023. [En línea]. Disponible en: https://python-holidays.readthedocs.io/en/latest/
[27] «La Función de Activación», Codificando Bits. Accedido: 3 de noviembre de 2023. [En línea]. Disponible en: https://www.codificandobits.com/blog/funcion-de-activacion/
[28] «2.24 - Otros algoritmos de optimización: Adam», Codificando Bits. Accedido: 3 de noviembre de 2023. [En línea]. Disponible en: https://www.codificandobits.com/curso/fundamentos-deep-learning-python/redes-neuronales-24-adam/

9. ANEXOS

Pruebas para definir la red neuronal

TRAIN/TEST	ESCALA	Nº CAPA	NEURONAS	FUN_ACT	LEARN RATE	PRED	CICLOS	LOSS
80/20	NO	3	256-128-64	RELU X3	0.001	2	100	14.38
80/20	NO	4	256-256-128-64	RELU X4	0.001	2	100	13.04
80/20	NO	4	256-256-128-64	RELU X4	0.005	2	100	78.9
80/20	SI	4	256-256-128-64	RELU X4	0.001	2	100	9.96
85/15	SI	4	256-256-128-64	RELU X4	0.001	2	100	5.349
85/15	SI	4	256-256-128-64	RELU X4	0.0005	2	100	10.22
85/15	SI	4	256-256-128-64	RELU-TANH X2	0.001	1	100	28.27
85/15	SI	4	256-256-128-64	RE-TH-RE-RE	0.001	1	100	11.61
90/10	SI	4	256-256-128-64	RELU X4	0.001	1	100	16.98
90/10	NO	4	256-256-128-64	RELU X4	0.001	1	100	33.55
82.5/17.5	SI	4	256-256-128-64	RELU X4	0.001	1	100	16.27
85/15	SI	4	256-256-128-64	RELU X4	0.001	1	100	3.66

85/15	SI	4	256-256-128-64	RELU X4	0.001	2	100	7.28
85/15	SI	4	256-256-128-64	RELU X4	0.001	1	125	2.949
85/15	SI	4	256-256-128-64	RELU X4	0.001	1	200	1.75
85/15	NO	4	256-256-128-64	RELU X4	0.001	1	200	9.42
85/15	SI	4	256-256-128-64	RELU X4	0.0008	1	200	3.558
85/15	SI	4	256-256-128-128	RELU X4	0.0011+ bat128	1	200	6.75
85/15	SI	4	256-256-128-128	RELU X4	0.001+ bat100	1	200	4.6
85/15	SI	4	256-256-128-128	RELU X4	0.001+ bat10	1	200	1.9
85/15	SI	4	256-256-128-64	RELU X4	0.001+ bat10	1	200	1.93
85/15	SI	4	256-256-128-64	RELU X4	0.001+ bat20	1	200	2.34
85/15	SI	4	256-256-128-64	RELU X4	0.001	1	200	1.89
80/20	NO	4	256-256-128-64	RELU X4	0.001+ bat200	1	200	14.33
80/20	NO	4	256-256-128-64	RELU X4	0.001+ bat128	1	200	3.47

80/20	NO	4	256-256-128-64	RELU X4	0.001	1	200	8.35
80/20	SI	4	256-256-128-64	RELU X4	0.001+ bat128	1	200	4.52
80/20	SI	3	256-256-128	RELU X3	0.001+ bat20	1	200	3.88
80/20	SI	3	256-256-128	RELU X3	0.001+ bat10	1	200	7.01
80/20	SI	4	256-256-128-64	RELU X4	0.001+ bat10	1	200	2.28
80/20	SI	5	256-256-128-64-32	RELU X5	0.001+ bat10	1	200	2.166
80/20	SI	5	256-128-64-32-32	RELU X5	0.001+ bat100	1	200	2.743
80/15	SI	5	256-128-64-32-32	RELU X5	0.001+ bat10	1	200	4.635
80/15	SI	5	256-128-64-32-16	RELU X5	0.001+ bat128	1	200	10.45
85/15	SI	5	256-128-64-64-32	RELU X5	0.001+ bat100	1	200	2.988
85/15	SI	5	256-256-128-64-32	RELU X5	0.001+ bat96	1	200	2.17
85/15	SI	5	256-256-128-128-64	RELU X5	0.001+ bat96	1	200	2.267
85/15	SI	5	256-256-128-128-64	RELU X5	0.001+ bat100	1	100	1.643

Código implementado

```
#----------------IMPORTAMOS LIBRERÍAS----------------
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from datetime import datetime,timedelta
import sklearn.model_selection as sk
import tensorflow as tf
import holidays

#----------------CARGAMOS Y LEEMOS EL DATASET----------------
datos_original=pd.read_excel('/content/datosescuela.xls',header=0)
datos_original.head()

#------------AÑADO COLUMNA DIAS DE LA SEMANA-------------
datos_original['Día de la Semana'] = datos_original['Fecha'].dt.day_name()
mapeo = {'Monday': 0,'Tuesday': 1,'Wednesday': 2,'Thursday': 3,'Friday':
4,'Saturday': 5,'Sunday': 6}
datos_original['Día Semana Numérico'] = datos_original['Día de la
Semana'].map(mapeo)   # Aplica el mapeo

#------------ELIMINAMOS VALORES ATÍPICOS----------------
def busca_fallos(datos, columna, limite):
    nuevo_df = datos.copy()
    for i in range(1, len(datos)):
        diferencia = abs(nuevo_df.at[i,columna] - nuevo_df.at[i-1,columna])
        if diferencia >= limite:
          valor_anterior=nuevo_df.at[i-5,columna]
          valor_posterior=nuevo_df.at[i+5,columna]
          nuevo_valor=(valor_anterior+valor_posterior)/2
          nuevo_df.at[i,columna]=nuevo_valor
    return nuevo_df
data = busca_fallos(datos_original,'Energía Activa Compra A+(kWh)',9)
len(datos_original)

#--------------TRANSFORMACIÓN DATOS FECHA Y HORA-------------------
data['Hora']=pd.to_datetime(data['Hora'], format='%H:%M') #convierte columna
hora en datetime
data['Fecha y hora'] = data['Fecha']
+pd.to_timedelta(data['Hora'].dt.strftime('%H:%M:%S'))  #une columna de fecha
y hora
data['Día'] = data['Fecha y hora'].dt.day
data['Mes'] = data['Fecha y hora'].dt.month
data['hora'] = data['Fecha y hora'].dt.hour
data['hora'] = data['hora'].astype(float)
data['minutos_en_horas'] = data['Fecha y hora'].dt.minute.astype(float) / 60.0
```

```
data['hora'] = data['hora'] + data['minutos_en_horas']
data.info() #comprueba tipo de datos

#----------FUNCIONES PARA INTRODUCIR LOS FESTIVOS----------
def navidad ():
  inicio = input('Fecha inicio navidad (formato YYYY-MM-DD): ') # Pedir al
usuario que ingrese la fecha de inicio
  try:  # Convertir la cadena de entrada en un objeto de fecha
    inicio = datetime.strptime(inicio, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  fin='2018-01-06'
  fin = input('Fecha fin navidad (formato YYYY-MM-DD): ')  # Pedir al usuario
que ingrese la fecha de finalización
  try:  # Convertir la cadena de entrada en un objeto de fecha
    fin = datetime.strptime(fin, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  if fin < inicio:  # Verificar si la fecha de finalización es posterior a la fecha de
inicio
    print('La fecha de finalización debe ser posterior a la fecha de inicio. Vuelve
a ejecutar el programa.')
    exit()
  lista_de_fechas = []
  fecha = inicio
  while fecha <= fin:
    lista_de_fechas.append(fecha)
    fecha += timedelta(days=1)
  return lista_de_fechas

def feria ():
  inicio = input('Fecha inicio feria (formato YYYY-MM-DD): ') # Pedir al usuario
que ingrese la fecha de inicio
  try:  # Convertir la cadena de entrada en un objeto de fecha
    inicio = datetime.strptime(inicio, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  fin='2018-04-21'
  fin = input('Fecha fin feria (formato YYYY-MM-DD): ')  # Pedir al usuario que
ingrese la fecha de finalización
  try:  # Convertir la cadena de entrada en un objeto de fecha
    fin = datetime.strptime(fin, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  if fin < inicio:  # Verificar si la fecha de finalización es posterior a la fecha de
inicio
```

```
    print('La fecha de finalización debe ser posterior a la fecha de inicio. Vuelve a ejecutar el programa.')
    exit()
  lista_de_fechas = []
  fecha = inicio
  while fecha <= fin:
    lista_de_fechas.append(fecha)
    fecha += timedelta(days=1)
  return lista_de_fechas

def ss ():
  inicio = input('Fecha inicio s_santa (formato YYYY-MM-DD): ') # Pedir al usuario que ingrese la fecha de inicio
  try:  # Convertir la cadena de entrada en un objeto de fecha
    inicio = datetime.strptime(inicio, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  fin='2018-04-01'
  fin = input('Fecha fin s_santa (formato YYYY-MM-DD): ')  # Pedir al usuario que ingrese la fecha de finalización
  try:  # Convertir la cadena de entrada en un objeto de fecha
    fin = datetime.strptime(fin, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  if fin < inicio:  # Verificar si la fecha de finalización es posterior a la fecha de inicio
    print('La fecha de finalización debe ser posterior a la fecha de inicio. Vuelve a ejecutar el programa.')
    exit()
  lista_de_fechas = []
  fecha = inicio
  while fecha <= fin:
    lista_de_fechas.append(fecha)
    fecha += timedelta(days=1)
  return lista_de_fechas

def vac_verano ():
  inicio = input('Fecha inicio vac_verano (formato YYYY-MM-DD): ') # Pedir al usuario que ingrese la fecha de inicio
  try:  # Convertir la cadena de entrada en un objeto de fecha
    inicio = datetime.strptime(inicio, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  fin='2018-08-31'
  fin = input('Fecha fin vac_verano (formato YYYY-MM-DD): ')  # Pedir al usuario que ingrese la fecha de finalización
  try:  # Convertir la cadena de entrada en un objeto de fecha
```

```
    fin = datetime.strptime(fin, '%Y-%m-%d')
  except ValueError:
    print('Formato de fecha incorrecto. Utilice el formato YYYY-MM-DD.')
    exit()
  if fin < inicio:  # Verificar si la fecha de finalización es posterior a la fecha de
inicio
    print('La fecha de finalización debe ser posterior a la fecha de inicio. Vuelve
a ejecutar el programa.')
    exit()
  lista_de_fechas = []
  fecha = inicio
  while fecha <= fin:
    lista_de_fechas.append(fecha)
    fecha += timedelta(days=1)
  return lista_de_fechas

def es_festivo(fecha):
    return fecha in festivos_and

def es_no_lectivo(fecha):
    return fecha in periodo_no_lectivo


festivos_and_2017=holidays.ES(prov='AN',years=2017, language="es")
festivos_and_2018=holidays.ES(prov='AN',years=2018, language="es")
festivos_and=festivos_and_2017+festivos_and_2018

periodo_navidad=navidad()
periodo_ss=ss()
periodo_feria=feria()
periodo_vac_verano=vac_verano()
periodo_no_lectivo=periodo_navidad+periodo_ss+periodo_feria+periodo_vac_v
erano

data['Festivo'] = data['Fecha'].apply(es_festivo)
data['No lectivo'] = data['Fecha'].apply(es_no_lectivo)

for i in range(0,len(data)):
  if data.at[i,'No lectivo']==True:
    data.at[i,'Festivo']=True

#------------DATOS ENTRE SEMANA---------------
datos_open = data[~data['Fecha'].dt.day_name().isin(['Saturday', 'Sunday'])]
datos_apertura = datos_open[~datos_open['Festivo'].isin([True])]
datos_apertura.reset_index(drop=True, inplace=True)
datos_entresemana=datos_apertura.drop(['Hora','CUPS','Fecha','Día de la
Semana','Festivo','No lectivo','Identificacion','minutos_en_horas','Energía
Inductiva Compra Ri+(kvarh)','Energía Capacitiva Compra Rc-(kvarh)',],axis=1)
datos_entresemana.set_index(['Fecha y hora'],inplace=True)
```

```
#----------TRANSFORMACION DATASET A NUESTRO GUSTO----------
--
df=data.drop(['Hora','CUPS','Fecha','Día de la Semana','No
lectivo','Identificacion','minutos_en_horas','Energía Inductiva Compra
Ri+(kvarh)','Energía Capacitiva Compra Rc-(kvarh)'],axis=1)  #No muestra las
columnas que no queremos
df.set_index(['Fecha y hora'],inplace=True)
df.describe()

#----------ANÁLISIS DE VALORES CONCRETOS----------
print(df.shape)
df.loc['2018-05-12'].describe() #muestra estadísticas de ese día
df.loc['2018-05-01'].tail(60).sort_index()  #muestra 10 valores random de ese
día ordenados por el índice

#----------GRÁFICO DIARIO----------
#creación gráfica día 1
day_1='2018-05-28'
df_day=df.loc[day_1] #muestra los datos de esa fecha
sns.set(rc={'figure.figsize':(20,8)}) #estilo de gráfico con seaborn
plt.plot(df_day['hora'],df_day['Energía Activa Compra
A+(kWh)'],label=day_1,marker='^',linewidth=1.5,color='red',linestyle='dashed')

#día 2
day_2='2018-09-25'
df_day=df.loc[day_2] #muestra los datos de esa fecha
plt.plot(df_day['hora'],df_day['Energía Activa Compra
A+(kWh)'],label=day_2,marker='o',linewidth=1.5,color='blue',linestyle='dotted')

#creación ejes,título,etc
plt.legend(loc='best',fontsize=20)
plt.ylabel('Energía Activa
(kWh)',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.yticks(range(0,81,4))
plt.xticks(range(0,24))
plt.xlabel('Hora del día',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.title('Consumo de Energía Total
Día',loc='left',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.tick_params(labelsize = 20)

#----------GRÁFICO SEMANAS----------
inicio_semana=pd.to_datetime('2018-05-21')
fin_semana=pd.to_datetime('2018-05-28')
df_semana=df.loc[inicio_semana:fin_semana,:] #muestra los datos de esa
fecha
sns.set(rc={'figure.figsize':(20,8)}) #estilo de gráfico con seaborn
plt.plot(df_semana.index,df_semana['Energía Activa Compra
A+(kWh)'],label='Energía Activa
(kWh)',marker='.',linewidth=1.5,color='blue',linestyle='dashed')
plt.legend(loc='best',fontsize=20)
```

```
plt.ylabel('Energía Activa
(kWh)',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.yticks(range(0,81,4))
plt.xlabel('Día de la
semana',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.title('Consumo de Energía Total
Semana',loc='left',fontdict={'fontsize':30,'fontweight':'bold','color':'green'})
plt.tight_layout()
plt.tick_params(labelsize = 20)

#---------------GRÁFICO MESES----------------
inicio_mes=pd.to_datetime('2018-03-01')
fin_mes=pd.to_datetime('2018-04-01')
df_mes=df.loc[inicio_mes:fin_mes,:] #muestra los datos de esa fecha
sns.set(rc={'figure.figsize':(20,8)}) #estilo de gráfico con seaborn
plt.plot(df_mes.index,df_mes['Energía Activa Compra A+(kWh)'],label='Energía
Activa (kWh)',marker='.',linewidth=1.5,color='blue',linestyle='dashed')
plt.legend(loc='best',fontsize=20)
plt.ylabel('Energía Activa
(kWh)',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.yticks(range(0,81,4))
plt.xlabel('Día del mes',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.title('Consumo de Energía Total
Mes',loc='left',fontdict={'fontsize':30,'fontweight':'bold','color':'green'})
plt.tight_layout()
plt.tick_params(labelsize = 17)

#------------GRÁFICO AÑO COMPLETO----------------
df['Energía Activa Compra A+(kWh)'].plot(color='blue')
sns.set(rc={'figure.figsize':(20,10)})
#creación ejes,título,etc
plt.legend(loc='best',fontsize=20)
plt.ylabel('Energía Activa
(kWh)',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.yticks(range(0,81,4))
plt.xlabel('Mes del año',fontdict={'fontsize':20,'fontweight':'bold','color':'green'})
plt.title('Consumo de Energía Total
Año',loc='left',fontdict={'fontsize':30,'fontweight':'bold','color':'green'})
plt.tick_params(labelsize = 17)

#---------------RED NEURONAL--------------------
#-----------SEPARAMOS CONJUNTOS DE DATOS FULL--------------
x1=df.iloc[:,1:6].values
y1=df.iloc[:,0].values

x_rest,x_test,y_rest,y_test=sk.train_test_split(x1,y1,test_size=0.075,
random_state=12)

x2=x_rest
y2=y_rest
```

```
x_train,x_valid,y_train,y_valid=sk.train_test_split(x2,y2,test_size=0.08108108,
random_state=10)

#-------------ESCALAMOS DATOS PARA FACILITAR CONVERGENCIA DEL
PROCESO-----------------
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
x_train=scaler.fit_transform(x_train)
x_valid=scaler.transform(x_valid)
x_test=scaler.transform(x_test)

#------------VARIABLES CON TAMAÑOS DE PAQUETES DE DATOS----------
valid_size=x_valid.shape[0]
train_size=x_train.shape[0]
test_size=x_test.shape[0]

#-----------SEPARAMOS CONJUNTOS DE DATOS OPEN----------
x1_open=datos_entresemana.iloc[:,1:5].values
y1_open=datos_entresemana.iloc[:,0].values

x_rest_open,x_test_open,y_rest_open,y_test_open=sk.train_test_split(x1_open
,y1_open,test_size=0.15, random_state=12)

x2_open=x_rest_open
y2_open=y_rest_open

x_train_open,x_valid_open,y_train_open,y_valid_open=sk.train_test_split(x2_o
pen,y2_open,test_size=0.1764706, random_state=10)

#-------------ESCALAMOS DATOS PARA FACILITAR CONVERGENCIA DEL
PROCESO-----------------
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
x_train_open=scaler.fit_transform(x_train_open)
x_valid_open=scaler.transform(x_valid_open)
x_test_open=scaler.transform(x_test_open)

#------------VARIABLES CON TAMAÑOS DE PAQUETES DE DATOS----------
valid_size_open=x_valid_open.shape[0]
train_size_open=x_train_open.shape[0]
test_size_open=x_test_open.shape[0]

#-------------MATRIZ 3D LSTM FULL-----------------
n_pasos_de_tiempo = 4
n_muestras_train = len(x_train) - n_pasos_de_tiempo + 1
x_train_3d = np.zeros((n_muestras_train, n_pasos_de_tiempo, 5))
n_muestras_valid = len(x_valid) - n_pasos_de_tiempo + 1
x_valid_3d = np.zeros((n_muestras_valid, n_pasos_de_tiempo, 5))
n_muestras_test = len(x_test) - n_pasos_de_tiempo + 1
```

```
x_test_3d = np.zeros((n_muestras_test, n_pasos_de_tiempo, 5))

# Rellena el tensor con datos
for i in range(n_muestras_train):
    x_train_3d[i] = x_train[i:i+n_pasos_de_tiempo]
for i in range(n_muestras_valid):
    x_valid_3d[i] = x_valid[i:i+n_pasos_de_tiempo]
for i in range(n_muestras_test):
    x_test_3d[i] = x_test[i:i+n_pasos_de_tiempo]

y_train_3d = y_train[:n_muestras_train] #y_train tenga el mismo número de
muestras
y_valid_3d = y_valid[:n_muestras_valid]
y_test_3d = y_test[:n_muestras_test]

x_train_3d = scaler.fit_transform(x_train_3d.reshape(-1, x_train_3d.shape[-
1])).reshape(x_train_3d.shape)
x_valid_3d = scaler.fit_transform(x_valid_3d.reshape(-1, x_valid_3d.shape[-
1])).reshape(x_valid_3d.shape)
x_test_3d = scaler.transform(x_test_3d.reshape(-1, x_test_3d.shape[-
1])).reshape(x_test_3d.shape)

#----------MATRIZ 3D LSTM OPEN----------
n_pasos_de_tiempo = 4
n_muestras_train_open = len(x_train_open) - n_pasos_de_tiempo + 1
x_train_3d_open = np.zeros((n_muestras_train_open, n_pasos_de_tiempo, 4))
n_muestras_valid_open = len(x_valid_open) - n_pasos_de_tiempo + 1
x_valid_3d_open = np.zeros((n_muestras_valid_open, n_pasos_de_tiempo, 4))
n_muestras_test_open = len(x_test_open) - n_pasos_de_tiempo + 1
x_test_3d_open = np.zeros((n_muestras_test_open, n_pasos_de_tiempo, 4))

# Rellena el tensor con datos
for i in range(n_muestras_train_open):
    x_train_3d_open[i] = x_train_open[i:i+n_pasos_de_tiempo]
for i in range(n_muestras_valid_open):
    x_valid_3d_open[i] = x_valid_open[i:i+n_pasos_de_tiempo]
for i in range(n_muestras_test_open):
    x_test_3d_open[i] = x_test_open[i:i+n_pasos_de_tiempo]

y_train_3d_open = y_train_open[:n_muestras_train_open] #y_train tenga el
mismo número de muestras
y_valid_3d_open = y_valid_open[:n_muestras_valid_open]
y_test_3d_open = y_test_open[:n_muestras_test_open]

x_train_3d_open = scaler.fit_transform(x_train_3d_open.reshape(-1,
x_train_3d_open.shape[-1])).reshape(x_train_3d_open.shape)
x_valid_3d_open = scaler.fit_transform(x_valid_3d_open.reshape(-1,
x_valid_3d_open.shape[-1])).reshape(x_valid_3d_open.shape)
x_test_3d_open = scaler.transform(x_test_3d_open.reshape(-1,
x_test_3d_open.shape[-1])).reshape(x_test_3d_open.shape)
```

```
#------------CREAMOS MODELOS--------------
def modelo_0():
  model=tf.keras.Sequential([
      tf.keras.layers.Input(shape=n_entradas),
      tf.keras.layers.Dense(256,activation='relu'),
      tf.keras.layers.Dense(256,activation='relu'),
      tf.keras.layers.Dense(128,activation='relu'),
      tf.keras.layers.Dense(128,activation='relu'),
      tf.keras.layers.Dense(64,activation='relu'),
      tf.keras.layers.Dense(n_predicciones)
  ])
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)
  model.compile(loss='mse',optimizer=optimizer,metrics=['mae','mse','mape'])
  return model

def modelo_1():
  model=tf.keras.Sequential([
      tf.keras.layers.Input(shape=(n_pasos_de_tiempo,n_entradas)),
      tf.keras.layers.LSTM(128,activation='tanh',return_sequences=True),
      tf.keras.layers.LSTM(64,activation='tanh',return_sequences=True),
      tf.keras.layers.LSTM(32,activation='tanh'),
      tf.keras.layers.Dense(n_predicciones)
  ])
  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001)
  model.compile(loss='mse',optimizer=optimizer,metrics=['mae','mse','mape'])
  return model

#--------------ENTRENAMOS MODELO DENSO FULL----------------
print('Comenzando entrenamiento... ')
n_entradas = 5
n_predicciones = 1
llamada_0=modelo_0()
entrenamiento_0=llamada_0.fit(x_train,y_train,epochs=100,batch_size=100,
verbose=True,validation_data=(x_valid,y_valid))
print('Entrenamiento completado')
validation_results = llamada_0.evaluate(x_valid, y_valid)
print("Pérdida en el conjunto de validación:", validation_results[0])
print("Precisión en el conjunto de validación:", validation_results[1])
test_results = llamada_0.evaluate(x_test, y_test)
print("Pérdida en el conjunto de prueba:", test_results[0])
print("Precisión en el conjunto de prueba:", test_results[1])

#--------------ENTRENAMOS MODELO DENSO OPEN----------------
print('Comenzando entrenamiento... ')
n_entradas = 4
n_predicciones = 1
llamada_0_open=modelo_0()
```

```
entrenamiento_0_open=llamada_0_open.fit(x_train_open,y_train_open,epochs
=100,batch_size=10,
verbose=True,validation_data=(x_valid_open,y_valid_open))
print('Entrenamiento completado')
validation_results_open = llamada_0_open.evaluate(x_valid_open,
y_valid_open)
print("Pérdida en el conjunto de validación:", validation_results_open[0])
print("Precisión en el conjunto de validación:", validation_results_open[1])
test_results_open = llamada_0_open.evaluate(x_test_open, y_test_open)
print("Pérdida en el conjunto de prueba:", test_results_open[0])
print("Precisión en el conjunto de prueba:", test_results_open[1])

#--------------ENTRENAMOS MODELO LSTM FULL------------------
print('Comenzando entrenamiento... ')
n_entradas = 5
n_predicciones = 1
llamada_1=modelo_1()
entrenamiento_1=llamada_1.fit(x_train_3d,y_train_3d,epochs=100,batch_size=
100, verbose=True,validation_data=(x_valid_3d,y_valid_3d))
print('Entrenamiento completado')
validation_results = llamada_1.evaluate(x_valid_3d, y_valid_3d)
print("Pérdida en el conjunto de validación:", validation_results[0])
print("Precisión en el conjunto de validación:", validation_results[1])
test_results = llamada_1.evaluate(x_test_3d, y_test_3d)
print("Pérdida en el conjunto de prueba:", test_results[0])
print("Precisión en el conjunto de prueba:", test_results[1])

#--------------ENTRENAMOS MODELO LSTM OPEN------------------
print('Comenzando entrenamiento... ')
n_entradas = 4
n_predicciones = 1
llamada_1_open=modelo_1()
entrenamiento_1_open=llamada_1_open.fit(x_train_3d_open,y_train_3d_open,
epochs=100,batch_size=10,
verbose=True,validation_data=(x_valid_3d_open,y_valid_3d_open))
print('Entrenamiento completado')
validation_results_open = llamada_1_open.evaluate(x_valid_3d_open,
y_valid_3d_open)
print("Pérdida en el conjunto de validación:", validation_results_open[0])
print("Precisión en el conjunto de validación:", validation_results_open[1])
test_results_open = llamada_1_open.evaluate(x_test_3d_open,
y_test_3d_open)
print("Pérdida en el conjunto de prueba:", test_results_open[0])
print("Precisión en el conjunto de prueba:", test_results_open[1])

#----------------VISUALIZACIÓN RESULTADOS DE ENTRENAMIENTO
MODELO DENSO----------------
sns.set(rc={'figure.figsize':(20,10)})
plt.xlabel('Ciclos',fontdict={'fontsize':20,'color':'black'})
plt.ylabel('Magnitud de pérdida',fontdict={'fontsize':20,'color':'black'})
```

```
plt.title('Evolución de pérdida durante el entrenamiento (Modelo
Denso)',fontsize=20)
plt.plot(entrenamiento_0.history['loss'],label='Entrenamiento full',linewidth=2.5)
plt.plot(entrenamiento_0.history['val_loss'],label='Validación full',linewidth=2.5)
plt.plot(entrenamiento_0_open.history['loss'],label='Entrenamiento
open',linewidth=2.5)
plt.plot(entrenamiento_0_open.history['val_loss'],label='Validación
open',linewidth=2.5)
plt.legend(loc='best',fontsize=20)
plt.tick_params(labelsize = 20)
plt.show()

#----------VISUALIZACIÓN RESULTADOS DE ENTRENAMIENTO MODELO
LSTM----------
sns.set(rc={'figure.figsize':(20,10)})
plt.xlabel('Ciclos',fontdict={'fontsize':20,'color':'black'})
plt.ylabel('Magnitud de pérdida',fontdict={'fontsize':20,'color':'black'})
plt.title('Evolución de pérdida durante el entrenamiento (Modelo
LSTM)',fontsize=20)
plt.plot(entrenamiento_1.history['loss'],label='Entrenamiento full',linewidth=2.5)
plt.plot(entrenamiento_1.history['val_loss'],label='Validación full',linewidth=2.5)
plt.plot(entrenamiento_1_open.history['loss'],label='Entrenamiento
open',linewidth=2.5)
plt.plot(entrenamiento_1_open.history['val_loss'],label='Validación
open',linewidth=2.5)
plt.legend(loc='best',fontsize=20)
plt.tick_params(labelsize = 20)
plt.show()

#----------PREDICCION MODELO DENSO FULL----------
predicciones_0 = llamada_0.predict(x_test,batch_size=100)
print('valor test:\n', y_test)
print('valor pred:\n', predicciones_0,'\n\n')


#----------FUNCION DE ACIERTO EN LAS PREDICCIONES----------
-
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error
def evaluar_rendimiento(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true,y_pred,squared=False)  #false para que
devuelva el rmse en lugar de mse
    mape = mean_absolute_percentage_error(y_true,y_pred)
    rendimiento = {
        'MSE': mse,
        'MAE': mae,
        'RMSE': rmse,
        'MAPE': mape,
```

```
    }
    return rendimiento
rendimiento_0_full= evaluar_rendimiento(y_test, predicciones_0)

#---------------FUNCION PARA CALCULAR EL ERROR ABSOLUTO----------------
------
def error(y_tr, y_pr,size):
    error=y_tr-y_pr.reshape(1,size)
    error_abs=np.abs(error)
    return error_abs

error_0_full = error(y_test, predicciones_0,test_size)
error_0_full=error_0_full.flatten()
error_max_0_full=np.max(error_0_full)  #devuelve el valor máximo

#-------------ESTADÍSTICAS RELEVANTES DEL ERROR DENSO FULL-----------
---
error_25_0_full=np.percentile(error_0_full,25)
error_50_0_full=np.percentile(error_0_full,50)
error_75_0_full=np.percentile(error_0_full,75)
xx='95'
error_xx_0_full=np.percentile(error_0_full,int(xx))

#----------------PREDICCION MODELO DENSO OPEN-------------------
predicciones_0_open = llamada_0_open.predict(x_test_open,batch_size=100)
print('valor test:\n', y_test_open)
print('valor pred:\n', predicciones_0_open,'\n\n')


rendimiento_0_open= evaluar_rendimiento(y_test_open, predicciones_0_open)

error_0_open = error(y_test_open, predicciones_0_open,test_size_open)
error_0_open = error_0_open.flatten()
error_max_0_open=np.max(error_0_open)  #devuelve el valor máximo

#-------------ESTADÍSTICAS RELEVANTES DEL ERROR DENSO OPEN----------
----
error_25_0_open=np.percentile(error_0_open,25)
error_50_0_open=np.percentile(error_0_open,50)
error_75_0_open=np.percentile(error_0_open,75)
xx='95'
error_xx_0_open=np.percentile(error_0_open,int(xx))

# Imprimir las métricas de rendimiento modelo denso
from rich.table import Table
from rich.console import Console
tabla=Table(title='Análisis de errores en datos de prueba',title_justify='center')
tabla.add_column('Métricas',header_style='red')
tabla.add_column('[blue]Denso full')
tabla.add_column('[blue]Denso open')
```

```
tabla.add_row("MSE:", str(rendimiento_0_full['MSE']),
str(rendimiento_0_open['MSE'])) #no admite diccionarios ni numeros, solo
cadenas
tabla.add_row("MAE:", str(rendimiento_0_full['MAE']),
str(rendimiento_0_open['MAE']))
tabla.add_row("RMSE:", str(rendimiento_0_full['RMSE']),
str(rendimiento_0_open['RMSE']))
tabla.add_row("MAPE(%):", str(rendimiento_0_full['MAPE']*100),
str(rendimiento_0_open['MAPE']*100))
tabla.add_row("Error_25:", str(error_25_0_full), str(error_25_0_open))
tabla.add_row("Error_50:", str(error_50_0_full), str(error_50_0_open))
tabla.add_row("Error_75:", str(error_75_0_full), str(error_75_0_open))
tabla.add_row("Error_"+xx, str(error_xx_0_full), str(error_xx_0_open))
tabla.add_row("Error_max:", str(error_max_0_full), str(error_max_0_open))
console=Console()
console.print(tabla)

#----------------PREDICCION MODELO LSTM FULL-------------------
predicciones_1=llamada_1.predict(x_test_3d,verbose=False)
print(predicciones_1.shape)

rendimiento_1_full=evaluar_rendimiento(y_test_3d,predicciones_1)
error_1_full=error(y_test_3d,predicciones_1,n_muestras_test)
error_1_full=error_1_full.flatten()
error_max_1_full=np.max(error_1_full)

#-------------ESTADÍSTICAS RELEVANTES DEL ERROR LSTM FULL-------------
-
error_25_1_full=np.percentile(error_1_full,25)
error_50_1_full=np.percentile(error_1_full,50)
error_75_1_full=np.percentile(error_1_full,75)
xx='95'
error_xx_1_full=np.percentile(error_1_full,int(xx))

#----------------PREDICCION MODELO LSTM OPEN-------------------
predicciones_1_open=llamada_1_open.predict(x_test_3d_open,verbose=False
)
print(predicciones_1_open.shape)

rendimiento_1_open=evaluar_rendimiento(y_test_3d_open,predicciones_1_op
en)
error_1_open=error(y_test_3d_open,predicciones_1_open,n_muestras_test_op
en)
error_1_open=error_1_open.flatten()
error_max_1_open=np.max(error_1_open)

#-------------ESTADÍSTICAS RELEVANTES DEL ERROR LSTM OPEN------------
--
error_25_1_open=np.percentile(error_1_open,25)
error_50_1_open=np.percentile(error_1_open,50)
```

```
error_75_1_open=np.percentile(error_1_open,75)
xx='95'
error_xx_1_open=np.percentile(error_1_open,int(xx))

# Imprimir las métricas de rendimiento modelo LSTM
from rich.table import Table
from rich.console import Console
tabla=Table(title='Análisis de errores en datos de prueba',title_justify='center')
tabla.add_column('Métricas',header_style='red')
tabla.add_column('[blue]LSTM full')
tabla.add_column('[blue]LSTM open')
tabla.add_row("MSE:", str(rendimiento_1_full['MSE']),
str(rendimiento_1_open['MSE'])) #no admite diccionarios ni numeros, solo
cadenas
tabla.add_row("MAE:", str(rendimiento_1_full['MAE']),
str(rendimiento_1_open['MAE']))
tabla.add_row("RMSE:", str(rendimiento_1_full['RMSE']),
str(rendimiento_1_open['RMSE']))
tabla.add_row("MAPE(%):", str(rendimiento_1_full['MAPE']*100),
str(rendimiento_1_open['MAPE']*100))
tabla.add_row("Error_25:", str(error_25_1_full), str(error_25_1_open))
tabla.add_row("Error_50:", str(error_50_1_full), str(error_50_1_open))
tabla.add_row("Error_75:", str(error_75_1_full), str(error_75_1_open))
tabla.add_row("Error_"+xx, str(error_xx_1_full), str(error_xx_1_open))
tabla.add_row("Error_max:", str(error_max_1_full), str(error_max_1_open))
console=Console()
console.print(tabla)

#----------GRAFICO REAL VS PREDICCION MODELO DENSO FULL----------
from matplotlib.ticker import MultipleLocator
ini=0
fin=50

fig, ax = plt.subplots(figsize=(25, 10))
ax.plot(y_test[ini:fin],label='valor
real',color='darkblue',marker='.',linewidth=1.5,linestyle='dashed')
ax.plot(predicciones_0[ini:fin],label='valor
predicción',color='red',marker='.',linewidth=1.5,linestyle='dashed')
ax.legend(loc='best',fontsize=20)
plt.title('Modelo Denso Full',fontsize=20)
ax.yaxis.set_minor_locator(MultipleLocator(2))  #sirve para la escala del eje
ax.yaxis.grid(which='minor', linestyle='dashed', color='white')
ax.yaxis.set_minor_formatter('{x:.0f}') #cambia el formato de los ejes con
numero de decimales y lo muestra
ax.tick_params(axis='y', which='minor', labelsize=15, labelcolor='darkgrey')
ax.tick_params(labelsize = 20)

#----------GRAFICO REAL VS PREDICCION MODELO DENSO OPEN----------
-
from matplotlib.ticker import MultipleLocator
```

```
ini=0
fin=50

fig, ax = plt.subplots(figsize=(25, 10))
ax.plot(y_test_open[ini:fin],label='valor
real',color='darkblue',marker='.',linewidth=1.5,linestyle='dashed')
ax.plot(predicciones_0_open[ini:fin],label='valor
predicción',color='red',marker='.',linewidth=1.5,linestyle='dashed')
ax.legend(loc='best',fontsize=20)
plt.title('Modelo Denso Open',fontsize=20)
ax.yaxis.set_minor_locator(MultipleLocator(2))  #sirve para la escala del eje
ax.yaxis.grid(which='minor', linestyle='dashed', color='white')
ax.yaxis.set_minor_formatter('{x:.0f}') #cambia el formato de los ejes con
numero de decimales y lo muestra
ax.tick_params(axis='y', which='minor', labelsize=15, labelcolor='darkgrey')
ax.tick_params(labelsize = 20)

#----------GRAFICO REAL VS PREDICCION MODELO LSTM FULL----------
from matplotlib.ticker import MultipleLocator
ini=0
fin=50

fig, ax = plt.subplots(figsize=(25, 10))
ax.plot(y_test_3d[ini:fin],label='valor
real',color='darkblue',marker='.',linewidth=1.5,linestyle='dashed')
ax.plot(predicciones_1[ini:fin],label='valor
predicción',color='red',marker='.',linewidth=1.5,linestyle='dashed')
ax.legend(loc='best',fontsize=20)
plt.title('Modelo LSTM Full',fontsize=20)
ax.yaxis.set_minor_locator(MultipleLocator(2))  #sirve para la escala del eje
ax.yaxis.grid(which='minor', linestyle='dashed', color='white')
ax.yaxis.set_minor_formatter('{x:.0f}') #cambia el formato de los ejes con
numero de decimales y lo muestra
ax.tick_params(axis='y', which='minor', labelsize=15, labelcolor='darkgrey')
ax.tick_params(labelsize = 20)

#----------GRAFICO REAL VS PREDICCION MODELO LSTM OPEN----------
from matplotlib.ticker import MultipleLocator
ini=0
fin=50

fig, ax = plt.subplots(figsize=(25, 10))
ax.plot(y_test_3d_open[ini:fin],label='valor
real',color='darkblue',marker='.',linewidth=1.5,linestyle='dashed')
ax.plot(predicciones_1_open[ini:fin],label='valor
predicción',color='red',marker='.',linewidth=1.5,linestyle='dashed')
ax.legend(loc='best',fontsize=20)
plt.title('Modelo LSTM Open',fontsize=20)
ax.yaxis.set_minor_locator(MultipleLocator(2))  #sirve para la escala del eje
ax.yaxis.grid(which='minor', linestyle='dashed', color='white')
```

```
ax.yaxis.set_minor_formatter('{x:.0f}') #cambia el formato de los ejes con numero de decimales y lo muestra
ax.tick_params(axis='y', which='minor', labelsize=15, labelcolor='darkgrey')
ax.tick_params(labelsize = 20)
```