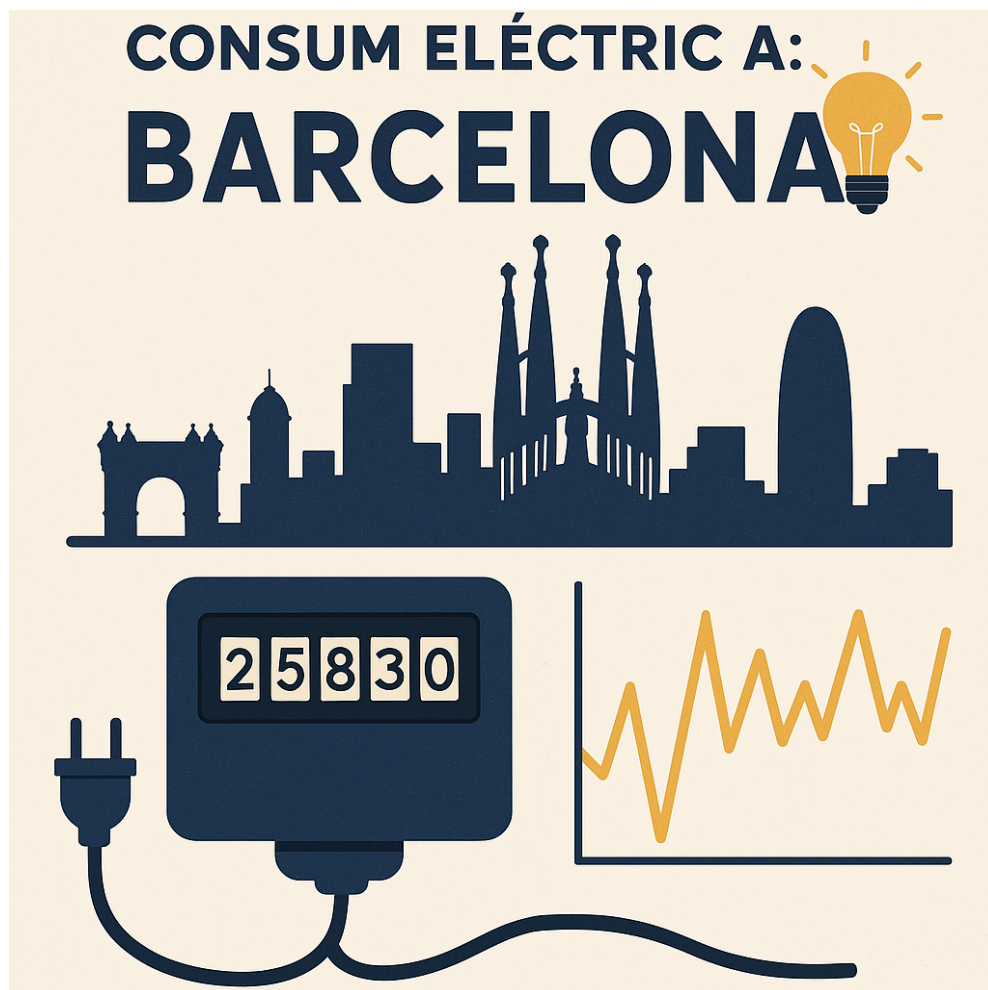


Demanda eléctrica de la ciudad de Barcelona



Adrián Yared Armas de la Nuez

Contenido

| | |
|--|----|
| 1. Introducción..... | 4 |
| 2. Breve resumen de productos o resultados obtenidos..... | 4 |
| 3. Contexto y justificación del Trabajo..... | 4 |
| 3.1 Objetivos del Trabajo..... | 5 |
| 3.2 Enfoque y metodología..... | 6 |
| 3.2.1 Enfoque..... | 6 |
| 3.2.1 metodología..... | 6 |
| 3.3 Planificación del Trabajo..... | 7 |
| 3.4. Recursos de sistemas utilizados..... | 8 |
| 4. Creación del set de datos..... | 9 |
| 4.1 Estudio de datos necesarios..... | 11 |
| 4.2 Fuente de datos..... | 12 |
| 4.3 Recolección de los datos..... | 12 |
| 4.4 Calidad e integridad de los datos..... | 12 |
| 4.5 Visualización de datos..... | 13 |
| 5. Optimización, normalización y calidad del set de datos..... | 13 |
| 6. Desarrollo del modelo predictivo..... | 14 |
| 6.1 Justificación del modelo seleccionado..... | 14 |
| 6.1.1 Comparativa..... | 14 |
| 6.1.2 Rendimiento..... | 14 |
| 6.1.3 Robustez..... | 15 |
| 6.1.4 Eficiencia computacional..... | 15 |
| 6.1.5 Justificación a nivel matemático..... | 16 |
| 6.1.6 Conclusión..... | 19 |
| 6.2 Descripción del modelo..... | 19 |
| 7. Entrenamiento y evaluación del modelo predictivo..... | 20 |
| 8. Informe de rendimiento y métricas en aula isarVdi y otros entornos..... | 21 |
| 9. Puesta en producción (API y aplicación cliente)..... | 21 |
| 10. Conclusiones y mejoras a realizar, modelos alternativos..... | 24 |
| 10.1 Conclusiones..... | 24 |
| 10.2 Mejoras..... | 26 |
| 11. Estructura completa del proyecto..... | 26 |
| 12. Código explicado..... | 27 |
| 12.1 Notebook..... | 27 |
| 12.1.1 Tabla de contenido..... | 27 |
| 12.1.1.1 Código..... | 27 |

| | |
|--|-----------|
| 12.1.2 Dependencias y librerías..... | 29 |
| 12.1.2.1 Contextualización del uso..... | 29 |
| 12.1.2.2 Instalación de librerías..... | 30 |
| 12.1.2.2.1 Código..... | 30 |
| 12.1.2.2.2 Ejecución..... | 31 |
| 12.1.2.3 Importación de librerías..... | 31 |
| 12.1.2.3.1 Código..... | 31 |
| 12.1.3 Recogida de campos necesarios..... | 32 |
| 12.1.3.1 Limpieza de datos..... | 33 |
| 12.1.3.2 Explicación..... | 34 |
| 12.1.3.3 Permisos y carga de datos..... | 35 |
| 12.1.3.4 Explicación..... | 35 |
| 12.1.3.5 Clonado y normalización..... | 36 |
| 12.1.3.6 Explicación..... | 37 |
| 12.1.3.7 Comprobación de la estructura de datos..... | 37 |
| 12.1.4 Análisis exploratorio de los datos (EDA)..... | 38 |
| 12.1.4.1 Unión de los csv..... | 38 |
| 12.1.4.2 Traducción de las columnas..... | 39 |
| 12.1.4.3 Detección de outliers..... | 39 |
| 12.1.4.4 Inserción de datos válidos en los outliers de los cuartiles... 40 | |
| 12.1.5 Entrenamiento del modelo..... | 42 |
| 12.1.5.1 Creación de atributos temporales..... | 42 |
| 12.1.5.2 Datos exploratorios junto a los atributos temporales..... | 43 |
| 12.1.5.3 División del dataset..... | 44 |
| 12.1.5.4 Definición de los atributos utilizados y la variable objetivo. 45 | |
| 12.1.5.5 Comparación de modelos..... | 46 |
| 12.1.5.6 Importancia de las variables en el entrenamiento..... | 49 |
| 12.1.6 Validación del modelo..... | 50 |
| 12.1.6.1 Gráfica..... | 50 |
| 12.1.6.2 Métrica de precisión..... | 51 |
| 12.1.7 Predicción de valores futuros..... | 52 |
| 12.1.7.1 Gráfica comparativa..... | 52 |
| 12.1.7.2 Ejemplo de predicción futura..... | 53 |
| 12.1.8 Guardado de los pesos y valor predicho..... | 54 |
| 12.1.9 Carga de los pesos..... | 55 |
| 12.2 Frontend..... | 56 |
| 12.2.1 Resumen..... | 56 |
| 12.2.2 Estilos y CSS..... | 56 |
| 12.2.3 Funcionalidad..... | 56 |

| | |
|--|----|
| 12.2.4 Visualización..... | 57 |
| 12.2.5 Conclusión..... | 57 |
| 12.2.6 Código completo..... | 57 |
| 12.3 Backend..... | 62 |
| 12.3.1 Resumen..... | 62 |
| 12.3.2 Funcionamiento..... | 62 |
| 12.3.3 Ruta Principal (/predict_range)..... | 62 |
| 12.3.4 Consideraciones implementadas..... | 63 |
| 12.3.5 Conclusión..... | 63 |
| 12.3.6 Código completo..... | 63 |
| 12.4 Instalador..... | 65 |
| 12.4.1 Resumen..... | 65 |
| 12.4.1.1 Verificación de Python..... | 65 |
| 12.4.1.2 Validación posterior a la instalación..... | 65 |
| 12.4.1.3 Gestión de pip..... | 65 |
| 12.4.1.4 Creación del entorno virtual..... | 65 |
| 12.4.1.5 Activación e instalación de dependencias..... | 65 |
| 12.4.1.6 Finalización..... | 65 |
| 12.4.2 Código completo..... | 66 |
| 12.5 Ejecutable..... | 69 |
| 12.4.1 Resumen..... | 69 |
| 12.4.2 Código completo..... | 69 |
| 13. Conclusión..... | 70 |
| 14. Bibliografía / Webgrafía..... | 71 |
| 15. Github..... | 72 |

1. Introducción.

Este informe describe el desarrollo de un sistema de análisis predictivo aplicado al consumo eléctrico en la ciudad de Barcelona. Se parte de datos históricos de generación y consumo energético con el fin de prever la demanda futura y aportar herramientas útiles para la gestión energética de la ciudad.

2. Breve resumen de productos o resultados obtenidos.

Como resultado del proyecto, se obtuvieron diversos productos y entregables clave que abarcan desde el procesamiento de datos hasta el desarrollo de herramientas aplicables en entornos reales.

En primer lugar, se realizó la limpieza y estructuración de un conjunto de datos histórico sobre consumo energético. Este proceso incluyó la depuración de registros, el tratamiento de valores atípicos y la imputación de datos faltantes, lo cual permitió conformar una base de datos robusta, coherente y lista para análisis avanzados.

Posteriormente, se llevaron a cabo visualizaciones detalladas de los patrones de consumo energético, segmentados por fecha, hora y sector económico. A través de dashboards interactivos y representaciones gráficas, se identificaron tendencias estacionales, picos de consumo y diferencias significativas entre sectores como el residencial, industrial y comercial, proporcionando información valiosa para la toma de decisiones.

Con esta base de conocimiento, se entrenó un modelo predictivo utilizando el algoritmo XGBoost, altamente eficaz en tareas de regresión. El modelo incorporó variables temporales, climáticas y sectoriales, logrando predicciones precisas del consumo energético a corto y mediano plazo.

La validez del modelo fue evaluada mediante pruebas en diferentes entornos y técnicas de validación cruzada, obteniendo métricas de rendimiento satisfactorias (RMSE, MAE, R^2) y superando con claridad los modelos de referencia utilizados como línea base.

Finalmente, se desarrolló una API RESTful que expone las predicciones generadas por el modelo, así como una aplicación cliente que permite a los usuarios consultar y visualizar dicha información de forma clara e intuitiva. Esta solución integral está orientada a facilitar la toma de decisiones informadas tanto en el ámbito empresarial como en la gestión pública del consumo energético.

3. Contexto y justificación del Trabajo.

En un contexto global marcado por el crecimiento urbano, el cambio climático y la transición hacia modelos energéticos sostenibles, el consumo energético en las ciudades se ha

convertido en un factor crítico para la planificación y gestión de los recursos. Barcelona, como una metrópolis con una alta densidad poblacional y una intensa actividad económica y social, enfrenta una creciente demanda energética que requiere respuestas eficaces y fundamentadas en datos.

La capacidad de prever la evolución del consumo eléctrico es esencial para diseñar estrategias que permitan no solo garantizar el suministro energético, sino también mejorar su eficiencia y sostenibilidad. Un sistema de predicción de la demanda energética permite optimizar la gestión de las infraestructuras existentes, anticipar necesidades futuras, reducir el riesgo de sobrecargas y, en última instancia, minimizar el impacto ambiental asociado a la generación y distribución de energía.

Este trabajo surge como respuesta a una necesidad detectada en el ámbito local: la falta de actualización del modelo de regresión utilizado por la red eléctrica de la ciudad de Barcelona desde el año 2022. En un entorno dinámico y cambiante, en el que las condiciones sociales, climáticas y tecnológicas evolucionan constantemente, mantener actualizados los modelos predictivos es fundamental para que sigan siendo útiles y precisos. La ausencia de revisiones recientes en estos modelos compromete la fiabilidad de las estimaciones actuales y puede limitar la capacidad de respuesta de las autoridades municipales y los operadores energéticos.

Por tanto, este proyecto tiene como objetivo desarrollar un nuevo modelo de análisis predictivo, más actualizado y adaptado a las condiciones actuales, que sirva como una herramienta de apoyo a la toma de decisiones en materia energética. La iniciativa se alinea con los principios de las smart cities y con los compromisos de sostenibilidad y eficiencia energética promovidos tanto a nivel local como europeo.

3.1 Objetivos del Trabajo.

El objetivo principal del proyecto es anticipar la demanda energética futura mediante la aplicación de modelos de predicción basados en datos históricos. Para ello, se ha recopilado y procesado un conjunto de datos provenientes de fuentes oficiales que incluyen registros de generación y consumo eléctrico, distribuidos en diferentes franjas horarias, estaciones del año y zonas geográficas de la ciudad.

El enfoque adoptado combina técnicas de análisis estadístico, aprendizaje automático y visualización de datos, con el fin de identificar patrones de comportamiento energético y establecer relaciones entre distintas variables, tales como la temperatura, la actividad económica, el calendario laboral o la densidad poblacional. Estos patrones permiten construir modelos predictivos capaces de estimar la demanda con un alto grado de precisión.

Además de su utilidad en la planificación de infraestructuras y en la optimización del uso de recursos energéticos, el sistema propuesto constituye una herramienta estratégica para los

responsables de la gestión energética municipal. Permite anticipar posibles picos de demanda, mejorar la eficiencia del sistema eléctrico y avanzar hacia un modelo más sostenible y resiliente frente a las variaciones del consumo.

Este trabajo se enmarca en los esfuerzos por promover el uso inteligente de los datos (smart data) dentro de las políticas de transformación digital y sostenibilidad urbana impulsadas por el Ayuntamiento de Barcelona.

3.2 Enfoque y metodología.

El trabajo se ha desarrollado siguiendo un enfoque estructurado de *Data Science*, el cual permite abordar problemas de predicción mediante un ciclo iterativo centrado en los datos y orientado a resultados aplicables. Este enfoque ha sido complementado con técnicas de *Machine Learning* supervisado, aplicando buenas prácticas de ingeniería de datos y evaluación de modelos.

A continuación se desglosan el enfoque adoptado y la metodología seguida.

3.2.1 Enfoque.

El enfoque adoptado para este trabajo se basa en el ciclo de vida típico de un proyecto de ciencia de datos, estructurado en varias fases clave. Primero, se realiza la comprensión del problema, definiendo el objetivo del proyecto, identificando las variables a predecir y entendiendo el impacto de las predicciones. A continuación, se lleva a cabo la adquisición y exploración de datos, donde se recopilan los datos necesarios y se examina su estructura, contenido y posibles inconsistencias.

Luego, en la fase de preparación de datos, se limpian, transforman y enriquecen los datos para garantizar su calidad y adecuación al modelado. Posteriormente, se aplica el modelado predictivo, utilizando algoritmos de Machine Learning para entrenar modelos capaces de predecir el consumo energético. La evaluación del desempeño de estos modelos se realiza mediante métricas cuantitativas y análisis de error. Finalmente, en la fase de despliegue, el modelo se integra en un entorno accesible a través de una API y una interfaz cliente que permite realizar consultas en tiempo real.

Este enfoque iterativo facilitó la validación continua de las decisiones tomadas, permitiendo ajustar tanto los datos como los modelos conforme se obtenían nuevos hallazgos.

3.2.1 metodología.

Para implementar el enfoque descrito, se adoptó una metodología práctica fundamentada en herramientas ampliamente reconocidas tanto en la comunidad científica como en la industria de datos.

En primer lugar, durante la exploración y limpieza de datos, se utilizó la biblioteca pandas para cargar, filtrar y transformar la información, eliminando duplicados, imputando valores nulos y convirtiendo campos de fecha y hora para facilitar el análisis temporal.

Posteriormente, la visualización y análisis exploratorio se apoyaron en matplotlib y seaborn, con las cuales se construyeron gráficos que permitieron identificar tendencias diarias, semanales y estacionales en el consumo energético.

La ingeniería de características incluyó la codificación de variables categóricas, como el sector económico, y la extracción de atributos relevantes a partir de las fechas, como el día de la semana o la hora del día. Para la selección y entrenamiento de modelos, se eligió XGBoost debido a su eficacia con datos tabulares y su alta precisión en tareas de regresión, validando su desempeño mediante la comparación con modelos base.

La evaluación del modelo se realizó utilizando métricas como el Error Absoluto Medio (MAE) y la Raíz del Error Cuadrático Medio (RMSE), complementadas con validación cruzada para reducir la varianza en los resultados. Finalmente, para el despliegue del modelo se desarrolló una API REST con Flask, que permite recibir datos de entrada y devolver predicciones en tiempo real, integrada a su vez con una aplicación cliente que actúa como interfaz gráfica de usuario.

Esta metodología garantizó un flujo de trabajo ordenado, reproducible y documentado, aspectos clave para asegurar la transparencia y escalabilidad del proyecto.

3.3 Planificación del Trabajo.

El desarrollo del sistema predictivo se ha estructurado en varias fases clave que permiten avanzar de forma ordenada desde la recopilación de datos hasta la implementación de una herramienta funcional para los usuarios finales.

En primer lugar, se realiza la revisión y limpieza de datos, donde se recopilan y validan los registros históricos de generación y consumo eléctrico. En esta fase, se corrigen errores, se gestionan valores faltantes y se eliminan inconsistencias para garantizar que la información sea fiable y adecuada para el análisis posterior.

A continuación, se lleva a cabo un análisis exploratorio y visualización de los datos, utilizando técnicas estadísticas y gráficos que facilitan la identificación de patrones temporales, estacionales y otras tendencias relevantes. Esta exploración permite entender mejor las variables que afectan al consumo energético y orienta la selección de características para los modelos predictivos.

Basándose en estos conocimientos, se procede a la ingeniería de características, donde se generan nuevas variables derivadas que enriquecen el conjunto de datos. Por ejemplo, se incorporan indicadores temporales como el día de la semana o festivos, así como factores externos que puedan influir en la demanda, mejorando así la capacidad predictiva del modelo.

Posteriormente, se aborda el entrenamiento y evaluación de modelos mediante la aplicación de diferentes algoritmos de aprendizaje automático. Se ajustan y validan los modelos utilizando técnicas adecuadas para asegurar su precisión y capacidad de generalización, seleccionando finalmente el más eficiente para las predicciones de consumo eléctrico.

Finalmente, para facilitar el acceso y la interacción con el sistema, se desarrolla una API junto con un cliente que permite a los usuarios consultar las predicciones y visualizar los resultados de forma intuitiva. Esta solución garantiza la integración del modelo en sistemas de gestión energética y su usabilidad para distintos perfiles de usuarios.

3.4. Recursos de sistemas utilizados.

Para el desarrollo del sistema predictivo se han seleccionado cuidadosamente diversas herramientas y recursos tecnológicos que permiten un equilibrio óptimo entre funcionalidad, facilidad de uso y eficiencia, adaptándose a las necesidades específicas del proyecto.

En cuanto a los entornos de desarrollo he optado por utilizar Visual Studio Code y Jupyter Notebooks debido a sus ventajas complementarias. Visual Studio Code es un editor ligero y altamente personalizable, que facilita la gestión del código fuente y la integración con sistemas de control de versiones como Git. Su amplia comunidad y la gran cantidad de extensiones disponibles permiten una experiencia de desarrollo fluida y adaptable a diferentes lenguajes y flujos de trabajo.

Por otro lado, Jupyter Notebooks ofrece un entorno interactivo ideal para el análisis exploratorio de datos y la experimentación con modelos. Su capacidad para combinar código, visualizaciones y documentación en un solo documento lo convierte en una herramienta especialmente útil para proyectos de ciencia de datos, donde el proceso iterativo y la claridad en la presentación son claves. Se descartaron IDEs más pesados o menos flexibles, como PyCharm, debido a que el proyecto requería mayor dinamismo y simplicidad para la etapa de prototipado.

En la parte de la infraestructura de servidor, el uso del servidor local Isard, disponible en el IES El Rincón, responde a la necesidad de contar con un entorno controlado, seguro y con recursos adecuados para procesar los datos y desplegar la solución. Se priorizó un servidor interno para evitar problemas de dependencia. Además, disponer de un entorno propio facilita la personalización y el control directo sobre las configuraciones, algo que no siempre es posible con soluciones en la nube, que pueden implicar costes adicionales y limitaciones en el acceso a datos.

Las librerías y herramientas de Python que elegí utilizar, son librerías reconocidas y ampliamente utilizadas en la comunidad de ciencia de datos para asegurar robustez, soporte y facilidad de integración:

Pandas se seleccionó por su capacidad para manejar grandes volúmenes de datos tabulares con eficiencia y flexibilidad, superando en estos aspectos a librerías más básicas o específicas.

Para la visualización, matplotlib y seaborn fueron escogidas por su complementariedad: matplotlib permite un control detallado de gráficos, mientras que seaborn facilita la creación de visualizaciones estadísticas atractivas y fáciles de interpretar. Otras opciones como Plotly o Bokeh, más orientadas a gráficos interactivos, fueron descartadas para esta fase inicial debido a la necesidad de simplicidad y rapidez en el análisis.

En cuanto a modelado, scikit-learn es una librería estándar para machine learning, que ofrece una amplia variedad de algoritmos y herramientas para evaluación, siendo ideal para comparar métodos y establecer una base sólida.

se implementaron varios modelos, de entre los que destacó xgboost, este se incorporó para mejorar la precisión en la predicción, ya que su técnica de boosting de árboles de decisión es altamente eficiente y se ha demostrado superior en numerosos retos de regresión, frente a modelos más simples o lineales.

El framework para despliegue de la aplicación, Flask fue la opción elegida por su ligereza y simplicidad para crear APIs REST. Frente a frameworks más pesados como Django, Flask ofrece mayor flexibilidad y menor curva de aprendizaje, lo que facilita iterar rápidamente y adaptar la solución a necesidades futuras.

Asimismo, se contempló la integración con un frontend web para mejorar la experiencia del usuario y la posibilidad de usar Streamlit como una herramienta rápida para crear interfaces interactivas, sin necesidad de un desarrollo web complejo. Esto permite ofrecer visualizaciones y consultas en tiempo real, adaptándose al perfil de usuarios no técnicos.

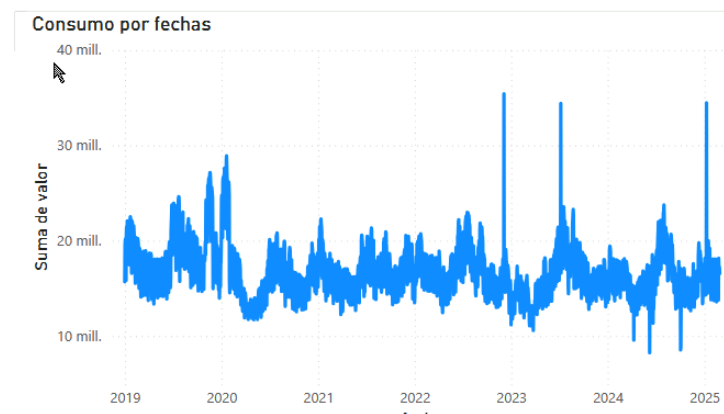
4. Creación del set de datos.

Para llevar a cabo un análisis exhaustivo del consumo energético en la ciudad de Barcelona, fue fundamental crear un set de datos robusto, limpio y representativo. Este proceso incluyó la obtención, limpieza, integración y visualización de datos provenientes de fuentes oficiales del ayuntamiento, concretamente a través de la plataforma Datadis* y el portal de datos abiertos de Barcelona*.

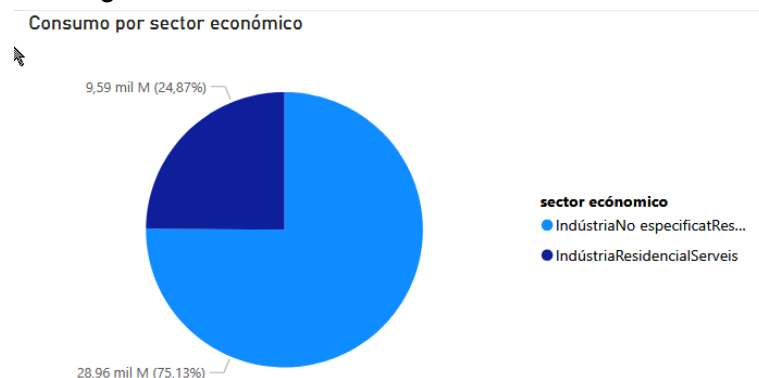
Posteriormente, realicé un análisis exploratorio de los datos en Power BI. Inicialmente, los gráficos generados se basan en datos sin limpiar. Esto me permitió visualizar de manera efectiva los *outliers* o valores atípicos, con el objetivo de identificarlos y decidir cómo filtrarlos posteriormente en el código de manera más eficiente.

Demanda eléctrica de la ciudad de Barcelona

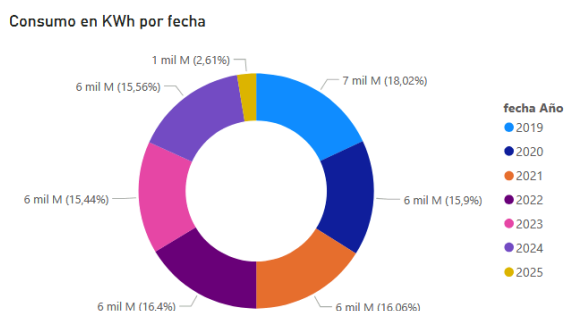
Al observar los gráficos en Power BI, se aprecia una tendencia general a la baja en el consumo energético desde el año 2020 hasta la fecha actual. Esta disminución podría deberse a diversos factores, como cambios en las políticas energéticas, mejoras en la eficiencia energética o una menor actividad industrial durante ciertos periodos, como la pandemia.



En cuanto a la distribución del consumo eléctrico, se observa que aproximadamente el 24,97% corresponde al sector industrial, mientras que el 75,14% está destinado al consumo residencial. Esto evidencia el peso significativo que tiene el ámbito doméstico en el uso de la energía eléctrica.



Adicionalmente, los datos reflejan que el consumo energético ha mostrado una mayor estabilidad a lo largo de los años, con un pico máximo registrado en 2019. Desde entonces, las variaciones han sido menos pronunciadas.





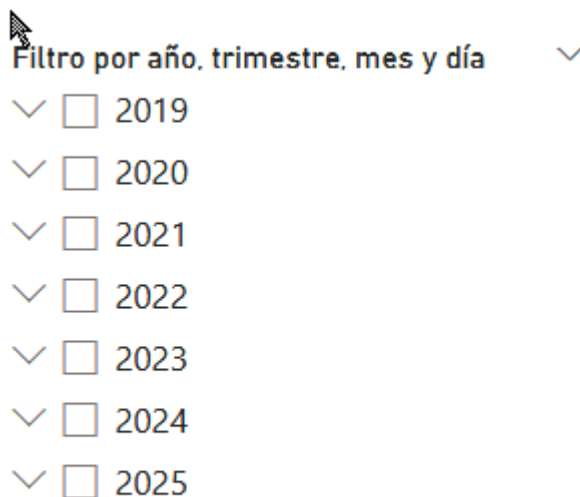
Demanda eléctrica de la ciudad de Barcelona

El total acumulado de consumo energético analizado alcanza los 38,55 mil millones de kilovatios-hora (KWh), lo cual representa un volumen considerable a considerar en términos de planificación y sostenibilidad energética.

Generación eléctrica por fecha (KWh)

I
38,55 mil M

Por último, implementé un filtro dinámico en el informe de Power BI que permite realizar consultas detalladas por día, mes, trimestre o año. Esto facilita la exploración interactiva de los datos y mejora la experiencia del usuario al permitirle identificar patrones temporales de forma más precisa.



4.1 Estudio de datos necesarios.

El primer paso consistió en definir los requerimientos del dataset. Para analizar adecuadamente los patrones de consumo energético, se identificaron las siguientes variables esenciales:

- Fecha: para segmentar y analizar tendencias temporales.
- Hora: para observar variaciones intradía y picos de demanda.
- Valor de consumo: la cantidad de energía consumida en un intervalo específico.

- Sector económico: para evaluar diferencias en consumo entre sectores como residencial, industrial, comercial, etc.

Adicionalmente, se estableció la necesidad de contar con datos en un formato que permitiera una fácil manipulación y análisis, priorizando registros que cubrieran un periodo temporal suficientemente amplio para detectar patrones estacionales y cambios a largo plazo.

4.2 Fuente de datos.

Los datos empleados provinieron del histórico energético de la ciudad de Barcelona, publicados oficialmente en el portal de datos abiertos del Ayuntamiento. Estos datos están disponibles en formato CSV, lo que facilita su acceso y manipulación.

El dataset incluye registros horarios de consumo energético detallados por sectores económicos, proporcionando un nivel de granularidad óptimo para análisis profundos. Además, al ser una fuente oficial, los datos cuentan con un respaldo institucional que garantiza su fiabilidad y precisión.

4.3 Recolección de los datos.

La recolección implicó descargar múltiples archivos CSV correspondientes a diferentes periodos y sectores. Estos archivos fueron recopilados manualmente desde la plataforma Datadis* y el portal de datos abiertos de Barcelona*.

Posteriormente, se fusionaron los diferentes archivos en una base de datos unificada, asegurando la continuidad temporal y la coherencia entre los registros. Para ello, se emplearon técnicas de manipulación de datos con herramientas como pandas en Python, que permitieron automatizar la fusión y evitar duplicados o inconsistencias.

4.4 Calidad e integridad de los datos.

Durante el proceso de revisión, se identificaron varios problemas comunes en los datos originales, tales como valores nulos en ciertos intervalos horarios o sectores, inconsistencias en los formatos de fechas y horas, así como la presencia de ruido y outliers, es decir, valores extremos o inconsistentes que podrían distorsionar el análisis. Para corregir estas fallas, se aplicaron diversas técnicas de limpieza y transformación, incluyendo la imputación o eliminación de valores nulos según su impacto, la normalización y estandarización de los formatos de fecha y hora, la detección y tratamiento de outliers mediante análisis estadísticos, y la verificación de la consistencia lógica entre columnas (por ejemplo, asegurando que los valores de consumo no fueran negativos). Gracias a estos

procedimientos, se logró obtener un dataset homogéneo, confiable y preparado para el análisis posterior.

4.5 Visualización de datos.

Una vez que los datos fueron limpiados y estructurados, se generaron diversas visualizaciones para explorar patrones y validar la calidad de la información. Para ello, se utilizaron bibliotecas de Python como seaborn y matplotlib, que permitieron crear gráficos para visualizar el consumo energético promedio por hora del día, identificando picos y horas valle; analizar el consumo diario a lo largo de semanas y meses para detectar estacionalidades o tendencias; y comparar el consumo entre distintos sectores económicos, resaltando diferencias y posibles áreas de interés. Además, para facilitar una exploración más clara e interactiva de los datos, se empleó Power BI, lo que me permitió interactuar con los gráficos y obtener insights de manera dinámica. Estas visualizaciones no solo facilitaron la interpretación de los datos, sino que también ayudaron a detectar posibles anomalías y a comunicar los hallazgos de forma clara y efectiva.

5. Optimización, normalización y calidad del set de datos.

Para preparar los datos para su análisis y modelado, se aplicaron diversas técnicas de transformación y preprocesamiento que optimizaron su estructura y formato. En primer lugar, se realizaron transformaciones temporales clave a partir de las columnas de fecha y hora, extrayendo variables como la hora del día, el día de la semana y la semana del año. Estas nuevas variables permitieron capturar patrones intradía, diferencias entre días laborables y fines de semana, así como tendencias estacionales a lo largo del año.

Además, se aplicó la normalización de las variables numéricas, especialmente en los valores de consumo energético, para escalar los datos dentro de un rango estándar, lo que facilitó un entrenamiento más eficiente de los modelos al evitar que las variables con valores absolutos mayores dominaran el proceso.

Finalmente, se realizó la codificación de las variables categóricas, como el sector económico, mediante técnicas como el one-hot encoding o label encoding, transformando estas categorías en formatos numéricos adecuados para su interpretación por los algoritmos.

Estas transformaciones aseguraron que el dataset estuviera correctamente estructurado y preparado, permitiendo que los modelos predictivos o analíticos pudieran aprovechar toda la información temporal, numérica y categórica para obtener resultados más precisos y relevantes.

6. Desarrollo del modelo predictivo.

Con el objetivo de predecir el consumo energético en distintos sectores y franjas horarias, se procedió al desarrollo de un modelo de aprendizaje automático que permitiera aprovechar las características estructuradas del dataset y capturar relaciones complejas entre las variables.

6.1 Justificación del modelo seleccionado.

6.1.1 Comparativa

Tras comparar diversos algoritmos de regresión, incluyendo regresión lineal, árboles de decisión, random forest y redes neuronales, se optó por utilizar XGBoost (Extreme Gradient Boosting) como modelo predictivo principal. Esta elección se fundamentó en una serie de ventajas que lo hacen especialmente adecuado para problemas de predicción con datos estructurados.

6.1.2 Rendimiento

En primer lugar, XGBoost ofrece un alto rendimiento en conjuntos de datos tabulares, superando a muchos otros modelos de machine learning en tareas similares. Además, posee una gran capacidad para capturar relaciones no lineales y complejas entre variables, aspecto crucial en la predicción del consumo energético, ya que este depende de múltiples factores interrelacionados como el tiempo, el sector económico o la estacionalidad. Podemos comprobar esto en el estudio científico "Data-driven ML models for accurate prediction of energy consumption in a low-energy house*6". En este estudio se compararon varios modelos de aprendizaje automático, incluyendo XGBoost, Random Forest, Árboles de Decisión y Máquinas de Vectores de Soporte, para predecir el consumo energético en una vivienda de bajo consumo en Bélgica. Los resultados mostraron que XGBoost superó a los demás modelos, logrando un coeficiente de determinación (R^2) de 0.61, un error cuadrático medio (RMSE) de 65.28, un error absoluto medio (MAE) de 29.81 y un error porcentual absoluto medio (MAPE) de 28.55 en el conjunto de prueba. Estos hallazgos destacan la capacidad de XGBoost para modelar dependencias no lineales entre las condiciones ambientales y el consumo energético total. Además, otro estudio titulado "Subway Energy Consumption Prediction based on XGBoost Model*7" aplicó XGBoost para predecir el consumo energético en el sistema de metro de Qingdao, China. Utilizando datos operativos reales, XGBoost demostró una mayor precisión en las predicciones en comparación con otros algoritmos como SVR y LSTM, acercándose más a los valores experimentales. Estos estudios respaldan la eficacia de XGBoost en la predicción del consumo energético, destacando su capacidad para manejar datos tabulares y capturar relaciones complejas entre variables.

6.1.3 Robustez

Otra ventaja significativa es su robustez ante valores atípicos y datos faltantes, gracias a su estructura basada en árboles de decisión, que permite manejar este tipo de irregularidades sin degradar el rendimiento del modelo. Asimismo, XGBoost muestra una notable flexibilidad para incorporar variables categóricas y temporales, sin requerir transformaciones complejas o costosas. Esta robustez la corroboran estudios como "Handling missing data of using the XGBoost-based multiple imputation by chained equations regression method"*⁸, este destaca que XGBoost puede predecir valores faltantes con precisión, superando las limitaciones de métodos tradicionales al manejar datos complejos. Y por la parte de la flexibilidad en el manejo de variables categóricas y temporales XGBoost también ofrece flexibilidad en la incorporación de variables categóricas y temporales. En el artículo "Data Preparation for Gradient Boosting with XGBoost in Python"*⁹, en este se explica cómo XGBoost puede manejar variables categóricas utilizando codificación one-hot y cómo puede procesar variables temporales al transformarlas adecuadamente. Estas capacidades hacen que XGBoost sea una herramienta poderosa para modelar datos con características diversas y complejas, como las que se encuentran en la predicción del consumo energético.

6.1.4 Eficiencia computacional

También, se destaca su eficiencia computacional y rapidez en el entrenamiento, incluso al trabajar con grandes volúmenes de datos, lo que facilita su aplicación en contextos reales donde la escalabilidad es un factor importante. Uno de los estudios más destacados que confirma su eficiencia computacional es "XGBoost: A Scalable Tree Boosting System"*¹⁰ de Tianqi Chen y Carlos Guestrin. En este trabajo, los autores presentan mejoras clave en la arquitectura de XGBoost, como la computación fuera de núcleo (out-of-core), el aprendizaje aproximado de árboles y la paralelización eficiente. Estas optimizaciones permiten que XGBoost maneje conjuntos de datos de hasta 1.7 mil millones de ejemplos utilizando recursos computacionales limitados, demostrando una escalabilidad lineal al aumentar el número de máquinas. Los experimentos muestran que XGBoost supera significativamente a otras bibliotecas como Spark MLlib y H2O en términos de velocidad y eficiencia en tareas de aprendizaje automático a gran escala.

Además, el estudio "XGBoost: Scalable GPU Accelerated Learning"* de Rory Mitchell y colaboradores introduce una implementación de XGBoost que aprovecha la aceleración por GPU. Esta versión permite entrenar modelos en conjuntos de datos con 115 millones de instancias en menos de tres minutos utilizando sistemas multi-GPU. La implementación utiliza técnicas de compresión de datos y paralelismo completo en GPU para lograr una eficiencia computacional notable.

Estos estudios demuestran que XGBoost no solo es eficaz en términos de precisión predictiva, sino que también es altamente eficiente y escalable, lo que lo convierte en una herramienta valiosa para aplicaciones del mundo real que requieren procesamiento de grandes volúmenes de datos en tiempos reducidos.

6.1.5 Justificación a nivel matemático

Finalmente, a nivel matemático, para justificar el uso del modelo, voy a explicar como funciona:

Objetivo general del modelo

Dado un conjunto de datos de entrenamiento:

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$$

donde $x_i \in \mathbb{R}^m$ son los vectores de características y y_i son las etiquetas reales, el objetivo de XGBoost es encontrar una función $\hat{y}_i = \hat{f}(x_i)$ que aproxime bien los valores reales y_i minimizando una función de pérdida.

El modelo predice a través de una suma de árboles de regresión:

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

donde \mathcal{F} es el conjunto de todos los árboles de regresión posibles (también llamados *cart trees* o *regression trees*).

Función objetivo

Para establecer la función objetivo que se quiere minimizar combina:

- Una función de pérdida \mathcal{L} que mide la diferencia entre las predicciones y los valores reales.
- Un término de regularización $\Omega(f_k)$ que penaliza la complejidad del modelo.

$$\mathcal{L}(\phi) = \sum_{i=1}^n \ell(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

XGBoost usa una regularización de tipo:

$$\Omega(f) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

T : número de hojas del árbol.

w_j : valor predicho por la hoja j .

γ : penalización por el número de hojas.

λ : coeficiente de regularización L2 sobre los pesos de las hojas.

Aproximación con expansión de Taylor

Para minimizar la función objetivo de forma eficiente, XGBoost usa una expansión de segundo orden de Taylor en cada iteración (al agregar un nuevo árbol)

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^n \left[\ell \left(y_i, \hat{y}_i^{(t-1)} \right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

donde:

$$g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} \ell \left(y_i, \hat{y}_i^{(t-1)} \right) \quad (\text{primer derivada, gradiente})$$

$$h_i = \frac{\partial^2}{\partial \hat{y}_i^{(t-1)^2}} \ell \left(y_i, \hat{y}_i^{(t-1)} \right) \quad (\text{segunda derivada, Hessiano})$$

$f_t(x_i)$ valor predicho por el nuevo árbol

Construcción del árbol

Cada árbol divide el espacio de características y asigna un valor constante a cada hoja. Sea $q(x_i)$ la asignación del ejemplo x_i a una hoja j , y w_j el valor de esa hoja:

$$f_t(x_i) = w_{q(x_i)}$$

Sustituyendo en la función objetivo:

$$\mathcal{L}^{(t)} \approx \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i \right) w_j^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Donde $I_j = \{i \mid q(x_i) = j\}$ es el conjunto de instancias en la hoja j .

Solución óptima para los pesos de cada hoja

Para cada hoja, se puede obtener analíticamente el peso óptimo (peso óptimo de la hoja j):

$$w_j^* = - \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Y el valor de la función objetivo se convierte en:

$$\mathcal{L}^{(t)} = -\frac{1}{2} \sum_{j=1}^T \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T$$

Este valor es utilizado por el algoritmo para decidir cómo dividir los nodos del árbol, eligiendo las divisiones que más reducen esta pérdida.

Proceso iterativo

Se comienza con una predicción inicial (por ejemplo, la media de y_i).

En cada iteración, se agrega un nuevo árbol f_t que ajusta los errores residuales del modelo actual.

El proceso se repite durante K iteraciones o hasta que no haya mejoras significativas.

Resumen matemático del modelo

XGBoost es un algoritmo de gradient boosting que construye un modelo predictivo sumando árboles de decisión secuencialmente para minimizar una función objetivo.

Matemáticamente, en cada iteración, XGBoost agrega un nuevo árbol que corrige los errores del modelo anterior utilizando una aproximación de segundo orden de la función de pérdida, es decir, emplea tanto los gradientes como los hessianos (derivadas primera y segunda) para optimizar más eficazmente.

Además, incluye una función de regularización que penaliza la complejidad del modelo (número de hojas y magnitud de sus pesos), lo que ayuda a evitar el sobreajuste y mejora la capacidad de generalización. Gracias a su estructura, permite paralelizar el entrenamiento y aplicar optimizaciones como poda de árboles, aprendizaje acelerado con GPU y compresión de datos, haciendo que sea muy rápido y escalable.

XGBoost es un modelo matemáticamente muy eficiente por varias razones importantes. Primero, para aprender y mejorar sus predicciones, no solo mira qué tan mal va el modelo actual (esto se llama gradiente), sino que también considera qué tan rápido cambia ese error (esto es el hessiano, o segunda derivada). Esto es como tener una mejor idea de la forma del "terreno" por donde camina el modelo, permitiéndole avanzar de manera más inteligente y rápida que otros métodos que solo usan la primera información (el gradiente).

Además, XGBoost controla qué tan complejo puede ser el modelo con un sistema de penalizaciones (regularización). Esto evita que el modelo se "memorice" los datos de entrenamiento y pierda la capacidad de funcionar bien con datos nuevos, lo que se llama evitar el sobreajuste. Así, las predicciones son más confiables y estables.

Por último, la forma en que está diseñado XGBoost permite usar técnicas que aceleran mucho el entrenamiento. Puede dividir el trabajo en partes que se ejecutan al mismo tiempo

(paralelización) y usar recursos especiales como tarjetas gráficas (GPU). Esto hace que el modelo sea muy rápido y pueda manejar grandes cantidades de datos sin perder eficiencia.

En resumen, XGBoost combina una forma inteligente de aprender, controla la complejidad para no sobreajustar y está preparado para trabajar rápido y en grande, lo que lo hace muy poderoso para muchos problemas.

En definitiva, XGBoost destaca por combinar precisión en la optimización, robustez frente al sobreajuste y eficiencia computacional. Su capacidad para usar información de segundo orden y regularización clara, junto con sus optimizaciones para entrenamiento rápido y escalable, lo convierten en una herramienta ideal para problemas complejos con grandes conjuntos de datos, como la predicción de consumo energético u otras tareas tabulares donde la relación entre variables es no lineal y multidimensional. Matemáticamente, XGBoost destaca por combinar precisión en la optimización, robustez frente al sobreajuste y eficiencia computacional. Su capacidad para usar información de segundo orden y regularización clara, junto con sus optimizaciones para entrenamiento rápido y escalable, lo convierten en una herramienta ideal para problemas complejos con grandes conjuntos de datos, como la predicción de consumo energético u otras tareas tabulares donde la relación entre variables es no lineal y multidimensional.

6.1.6 Conclusión

En conclusión, el modelo XGBoost se presenta como la mejor opción para predecir el consumo energético debido a su alta precisión, robustez y eficiencia computacional. Comparado con otros algoritmos, XGBoost destaca por su capacidad para manejar datos tabulares complejos y relaciones no lineales entre variables, así como por su resistencia ante valores atípicos y datos faltantes. Su fundamento matemático, que incluye la optimización mediante gradientes de segundo orden y una función de regularización para evitar el sobreajuste, garantiza predicciones estables y confiables. Además, su diseño eficiente permite escalar y acelerar el entrenamiento con técnicas como paralelización y uso de GPU, facilitando su aplicación en contextos reales con grandes volúmenes de datos. Por estas razones, XGBoost es una herramienta poderosa y adecuada para tareas de predicción complejas como el consumo energético, donde la interacción multidimensional de variables es crucial para obtener resultados precisos y aplicables.

6.2 Descripción del modelo.

Para predecir el consumo energético, se eligió utilizar XGBoost (Extreme Gradient Boosting), un modelo de aprendizaje automático supervisado que se basa en árboles de decisión y emplea la técnica de boosting. Este algoritmo es ampliamente reconocido por su efectividad en problemas de regresión y clasificación, especialmente cuando las relaciones entre las variables son complejas y no lineales.

En este caso, el modelo se entrenó con una combinación de variables temporales (como el año, mes, semana, día de la semana y si se trata de un feriado) y variables categóricas

relacionadas con los patrones de consumo. Entre ellas se destacan: estacionalidad, para identificar patrones cíclicos ; factores de calendario, como la distinción entre días laborables y festivos; y datos históricos de consumo, que ayudan al modelo a entender la evolución del comportamiento energético a lo largo del tiempo.

XGBoost se adapta muy bien a este tipo de problemas por varias razones. Cuenta con regularización incorporada, lo que mejora su capacidad de generalización; maneja de forma eficiente los valores faltantes; permite trabajar con datos de distinto tipo (numéricos y categóricos codificados); y es altamente eficiente en términos computacionales, gracias a su diseño optimizado.

Durante el entrenamiento, se aplicó validación cruzada para afinar los principales hiperparámetros del modelo: la profundidad máxima de los árboles (`max_depth`), la tasa de aprendizaje (`learning_rate`) y el número de árboles (`n_estimators`). Esto permitió reducir los errores de predicción y mejorar el desempeño general del modelo.

Gracias a este enfoque, el modelo logró capturar tanto las tendencias generales como los patrones estacionales del consumo energético, proporcionando predicciones sólidas y confiables para los próximos años en la zona de estudio.

7. Entrenamiento y evaluación del modelo predictivo.

El modelo fue entrenado utilizando datos históricos de consumo energético disponibles hasta el año 2025. Para evaluar su desempeño, se reservó una parte de estos datos como conjunto de validación, lo que permitió medir la capacidad del modelo para predecir sobre datos no vistos, simulando así su comportamiento en un entorno real.

La evaluación del modelo se realizó mediante métricas ampliamente utilizadas en problemas de regresión: RMSE (Root Mean Squared Error) y MAE (Mean Absolute Error). La primera permite identificar cuánto se desvían, en promedio, las predicciones respecto a los valores reales, penalizando más los errores grandes. La segunda, en cambio, ofrece una medida más robusta frente a valores atípicos al calcular el promedio de los errores absolutos.

Estas métricas fueron fundamentales para comparar el rendimiento de distintas configuraciones del modelo y seleccionar aquella que ofreciera mejores resultados en términos de precisión y generalización.

En las próximas secciones del código, que se explicará en el apartado doce, se detallará paso a paso el proceso de entrenamiento, así como el cálculo e interpretación de estas métricas de evaluación.

8. Informe de rendimiento y métricas en aula isarVdi y otros entornos.

El modelo fue probado en dos entornos distintos: el entorno virtual educativo isarVdi y la plataforma en la nube Google Colab. En ambos casos se documentó el rendimiento del modelo, así como los resultados de las métricas de evaluación.

Durante las pruebas, se obtuvo un error medio absoluto (MAE) dentro de los márgenes aceptables para tareas de predicción energética, lo que confirma la validez del enfoque y la calidad del ajuste del modelo.

En términos de rendimiento computacional, el entorno isarVdi demostró ser más eficiente en comparación con la versión gratuita de Google Colab. Esta diferencia se debe principalmente a que el modelo no requiere procesamiento por GPU, y el entorno virtual de isarVdi ofreció tiempos de ejecución más rápidos y estables al estar optimizado para cargas de trabajo de CPU.

Este análisis comparativo permitió validar la portabilidad del modelo y su correcto funcionamiento en diferentes plataformas, garantizando su utilidad tanto en entornos académicos como en aplicaciones prácticas.

9. Puesta en producción (API y aplicación cliente).

Para llevar el modelo a producción y facilitar su uso por parte de usuarios finales, se desarrolló una API REST utilizando Flask. Esta API expone las funcionalidades del modelo y permite realizar predicciones de consumo energético en función de la fecha y el sector seleccionado.

Además, se implementó una aplicación cliente que se conecta a esta API. La interfaz de la aplicación es sencilla, intuitiva y pensada para facilitar la interacción, permitiendo al usuario seleccionar un intervalo de fechas y visualizar los resultados a través de una gráfica interactiva.

Aunque el código fuente y su funcionamiento detallado serán explicados en el apartado 12, a continuación se muestra una vista general de la interfaz, para ilustrar cómo se presenta la información al usuario.

Predicción de Consumo por Intervalo de Fechas

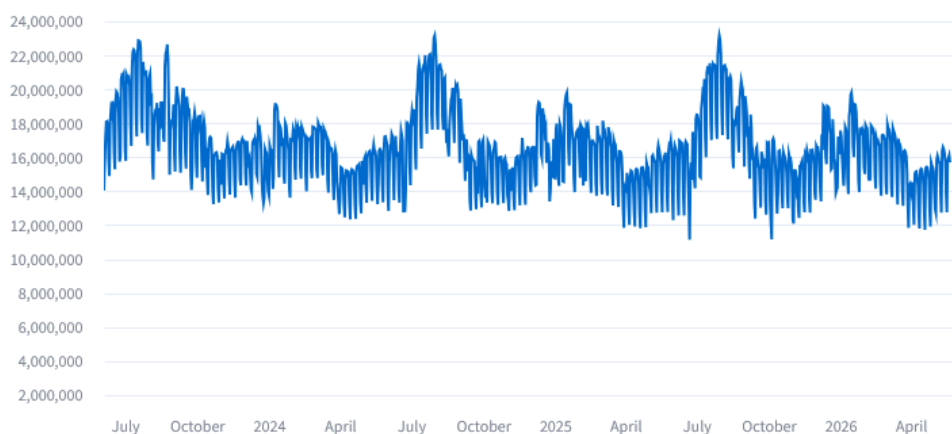
Fecha inicio

2023/05/21

Fecha fin

2026/05/21

Mostrar Predicciones



**Consumo
total estimado**

18,094,805,701.00

kWh



**Consumo
medio diario**

16,494,809.21

kWh



**Máxima
demanda
estimada**

23,159,888.00



**Mínima
demanda
estimada**

11,146,387.00

Junto a la visualización, la aplicación también calcula y muestra automáticamente diversas métricas relevantes para el análisis energético:

Consumo total estimado en el período de tiempo seleccionado (en kWh y GWh):

Como podemos ver en este ejemplo, el consumo estimado es de 18,094,805,701.00 kWh que equivale a 18,094.81 GWh:



**Consumo
total estimado**

18,094,805,701.00

kWh

(18,094.81 GWh)

Consumo medio diario, este apartado calcula la media (predicción total/n días) (en kWh y GWh).

En este ejemplo se ve que la media diaria es de 16,494,809.21 kWh que equivale a 16.4948 GWh:



Máxima demanda estimada, este apartado calcula la máxima demanda predicha (en kWh y GWh), junto con la fecha en la que se proyecta.

En este ejemplo podemos ver que en el intervalo de fechas seleccionado, el máximo valor predicho en un día es 23,159,888.00 kWh que equivale a 23.1599 GWh y su fecha, que en este caso es el día 2024-07-31:



Mínima demanda estimada, este apartado calcula la mínima demanda predicha (en kWh y GWh), junto con la fecha en la que se proyecta.


En este ejemplo podemos ver que en el intervalo de fechas seleccionado, el mínimo valor predicho en un día es 11,146,387.00 kWh que equivale a 11.1464 GWh y su fecha, que en este caso es el día 2025-06-22

 **Mínima
demanda
estimada**
**11,146,387.00
kWh**
(11.1464 GWh)
 **Día:**
2025-06-22


Estas funcionalidades permiten obtener una visión completa del comportamiento energético proyectado de forma clara y accesible.

Para facilitar su implementación, el proyecto incluye un archivo README en GitHub con instrucciones claras de instalación en español e inglés. Además, se proporcionaron herramientas específicas para usuarios de Windows, como:

installer.bat: permite una instalación automática sin necesidad de configurar manualmente el entorno

| | | | |
|--|------------------|-----------------------|------|
|  intaller | 19/05/2025 13:57 | Archivo por lotes ... | 3 KB |
|--|------------------|-----------------------|------|

ejecutable.bat: permite iniciar la aplicación con un doble clic, sin ejecutar comandos desde la terminal.

| | | | |
|--|------------------|-----------------------|------|
|  Ejecutable | 19/05/2025 14:18 | Archivo por lotes ... | 2 KB |
|--|------------------|-----------------------|------|

Este enfoque busca garantizar una experiencia accesible tanto para usuarios técnicos como no técnicos, y simplificar el despliegue del sistema en distintos entornos.

10. Conclusiones y mejoras a realizar, modelos alternativos.

10.1 Conclusiones

El desarrollo del sistema de predicción de consumo energético ha demostrado resultados muy satisfactorios, alcanzando una precisión del 96.46% con el modelo principal basado en

XGBoost. Esta precisión, respaldada por métricas sólidas de error (MAE: 559,050; RMSE: 790,375; MAPE: 3.43%), posiciona al modelo como una herramienta confiable para la estimación del consumo eléctrico en distintos escenarios y sectores económicos.

Se realizó una comparativa entre las predicciones generadas por los modelos entrenados y los datos reales de consumo proporcionados por la red eléctrica, lo que permitió validar la capacidad del sistema para aproximarse al comportamiento real del consumo energético. El modelo de XGBoost no solo presentó una alta precisión, sino que también mostró mayor consistencia frente a variaciones estacionales y horarios, superando en rendimiento a otros módulos de machine learning empleados como referencia.

A continuación, se detallan los resultados obtenidos por distintos modelos utilizados durante la fase de experimentación, ordenados según su desempeño:

| Modelo | MAE | RMSE | MAPE (%) |
|---------------------------|--------------|--------------|----------|
| Gradient Boosting | 551,130.08 | 795,303.46 | 3.39 |
| XGBoost | 559,049.98 | 790,375.00 | 3.43 |
| HistGradientBoosting | 562,917.90 | 804,712.77 | 3.44 |
| Random Forest | 563,455.67 | 825,492.06 | 3.46 |
| Elastic Net | 722,464.98 | 952,608.29 | 4.36 |
| Linear Regression | 724,494.41 | 954,024.03 | 4.37 |
| K-Nearest Neighbors (KNN) | 759,089.91 | 1,134,054.24 | 4.60 |
| Support Vector Regression | 1,786,146.68 | 2,278,986.20 | 10.91 |

Aunque Gradient Boosting logró un MAPE ligeramente inferior (3.39%) respecto a XGBoost (3.43%), la diferencia es marginal y el rendimiento general del modelo XGBoost fue superior en términos de estabilidad, escalabilidad y tiempo de entrenamiento.

10.2 Mejoras

Si bien el modelo actual basado en XGBoost ha demostrado un rendimiento sólido y una alta precisión, existen diversas oportunidades para mejorar aún más su robustez, adaptabilidad y capacidad explicativa. A continuación, se detallan algunas líneas de trabajo recomendadas:

Modelos híbridos o ensemble:

La combinación de múltiples enfoques, como XGBoost con redes neuronales profundas o modelos estadísticos clásicos, podría aumentar la resiliencia del sistema frente a eventos anómalos o condiciones atípicas no representadas en los datos históricos.

Actualización continua del modelo:

Diseñar un sistema de reentrenamiento periódico, basado en la incorporación de nuevos datos en tiempo real o por lotes, contribuiría a mantener el modelo actualizado ante cambios estructurales en los patrones de consumo energético, como nuevas políticas tarifarias o transformaciones en la demanda.

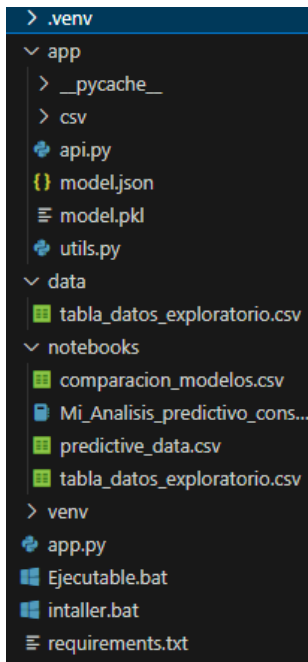
Exploración de modelos de series temporales tradicionales:

Aunque los modelos basados en árboles de decisión han superado en rendimiento a otras alternativas, una comparativa más profunda con modelos como ARIMA o Prophet podría ofrecer ventajas complementarias, especialmente en términos de interpretabilidad y análisis de tendencias a largo plazo.

En resumen, aunque el modelo XGBoost se posiciona actualmente como la solución más precisa y eficiente dentro del conjunto de métodos evaluados, la incorporación de enfoques híbridos, nuevas variables, mecanismos de actualización automática y modelos explicativos adicionales permitiría evolucionar hacia un sistema predictivo aún más completo, robusto y adaptado a contextos dinámicos.

11. Estructura completa del proyecto

A continuación adjunto una captura de la estructura de carpetas del proyecto y sus archivos:



12. Código explicado

12.1 Notebook

12.1.1 Tabla de contenido

En la parte superior del notebook se encuentra la tabla de contenido que permite ir a las secciones de código deseadas con mayor facilidad

12.1.1.1 Código

```
# Tabla de Contenidos

## 1. Preparación de los datos
- [1.1 Instalación de librerías] (#11-instalación-de-librerías)
- [1.2 Importación de librerías] (#12-importación-de-librerías)

## 2. Recogida de campos necesarios
- [2.1 Limpieza de datos] (#21-limpieza-de-datos)
- [2.2 Permisos y carga de datos] (#22-permisos-y-carga-de-datos)
- [2.3 Clonado de datos y normalización] (#23-clonado-de-datos-y-normalización)
- [2.4 Comprobación de la estructura de datos] (#24-comprobación-de-la-estructura-de-datos)
```

3. Análisis exploratorio de los datos (EDA)

- [3.1 Unión de los datasets] ([#31-uni3n-de-los-datasets](#))
- [3.2 .csv con datos actualizados a espaol] ([#32-csv-con-datos-actualizados-a-espaol](#))
- [3.3 Detección de outliers] ([#33-detección-de-outliers](#))
- [3.4 Inserción de datos válidos en los outliers de los cuartiles] ([#34-inserción-de-datos-válidos-en-los-outliers-de-los-cuartiles](#))

4. Entrenamiento del modelo

- [4.1 Creación de atributos temporales] ([#41-creación-de-atributos-temporales](#))
- [4.2 Datos exploratorio junto a los atributos temporales] ([#42-datos-exploratorio-junto-a-los-atributos-temporales](#))
- [4.3 División del dataset (70% train - 30% test)] ([#43-división-del-dataset-70-train---30-test](#))
- [4.4 Definición de los atributos utilizados (X) y la variable objetivo (Y)] ([#44-definición-de-los-atributos-utilizados-x-y-la-variable-objetivo-y](#))
- [4.5 Entrenamiento de varios modelos y métrica de precisión] ([#45-entrenamiento-de-varios-modelos-y-métrica-de-precisión](#))
- [4.6 Importancia de las variables en el entrenamiento] ([#46-importancia-de-las-variables-en-el-entrenamiento](#))

5. Validación del modelo

- [5.1 Gráfica] ([#51-gráfica](#))
- [5.2 Métricas de precisión] ([#52-métricas-de-precisión](#))

6. Predicción valores futuros

- [6.1 Comparativa valor real - valor predicho] ([#61-comparativa-valor-real---valor-predicho](#))
- [6.2 Ejemplo de predicción futura] ([#62-ejemplo-de-predicción-futura](#))

7. Guardado de los pesos y valor predicho

- [7. Guardado de los pesos y valor predicho] ([#7-guardado-de-los-pesos-y-valor-predicho](#))

8. Carga de los pesos

- [8. Carga de los pesos] ([#8-carga-de-los-pesos](#))


```
# Test 1 (Mejora de precisión)
- [Test 1 (Mejora de precisión)] (#test-1-mejora-de-precisión)

# Test 2 (Selección de modelos)
- [Test 2 (Selección de modelos)] (#test-2-selección-de-modelos)

# Test 3 (Selección de modelo y métricas de error con más modelos)
- [Test 3 (Selección de modelo y métricas de error con más
modelos)] (#test-3-selección-de-modelo-y-métricas-de-error-con-más-model
os)
```

12.1.2 Dependencias y librerías

En esta sección se ofrece una breve descripción de la utilidad de cada librería, junto con las instrucciones para la instalación de las necesarias.

12.1.2.1 Contextualización del uso

En primer lugar, se presenta una contextualización del propósito y la aplicación de las librerías instaladas, acompañada de enlaces a sus sitios web oficiales.

Lo primero que debemos hacer es importar y cargar las librerías que utilizaremos en el ejercicio.

- * `[pandas]` (<https://pandas.pydata.org/>) es una librería de Python especializada en la manipulación y análisis de datos.
- * `[numpy]` (<https://numpy.org/>) es una librería de Python para operaciones matemáticas avanzadas y manejo eficiente de arrays multidimensionales.
- * `[matplotlib]` (<https://matplotlib.org/>) es una librería de Python para la creación de gráficos estáticos, animados e interactivos.
- * `[seaborn]` (<https://seaborn.pydata.org/>) es una librería de Python basada en matplotlib, especializada en la visualización estadística.
- * `[xgboost]` (<https://xgboost.readthedocs.io/en/stable/>) es una librería de Python para la implementación eficiente de algoritmos de machine learning bajo el esquema de gradient boosting.
- * `[scikit-learn]` (<https://scikit-learn.org/stable/>) ofrece diversos modelos de regresión y algoritmos de machine learning, incluyendo RandomForestRegressor, GradientBoostingRegressor, ExtraTreesRegressor, HistGradientBoostingRegressor, LinearRegression, ElasticNet, SVR, y KNeighborsRegressor.

```
* [warnings] (https://docs.python.org/3/library/warnings.html) es una
librería para la gestión y control de mensajes de advertencia en
Python.
* [os] (https://docs.python.org/3/library/os.html) proporciona una
interfaz para interactuar con funcionalidades dependientes del sistema
operativo.
* [shutil] (https://docs.python.org/3/library/shutil.html) ofrece
operaciones de alto nivel para la gestión y manipulación de archivos y
directorios.
* [stat] (https://docs.python.org/3/library/stat.html) permite el manejo
de permisos y estados de archivos en el sistema operativo.
* [gitpython] (https://gitpython.readthedocs.io/en/stable/) es una
librería para interactuar con repositorios Git desde Python.
* [traceback] (https://docs.python.org/3/library/traceback.html)
facilita la captura y manejo de información de errores para depuración.
* [joblib] (https://joblib.readthedocs.io/en/latest/) es una librería
para la serialización eficiente y el guardado/carga de modelos y
objetos en Python.
```

12.1.2.2 Instalación de librerías

Procedimiento para la instalación individual de aquellas librerías que requieren una configuración previa.

12.1.2.2.1 Código

```
# Instalación de librerías necesarias
!pip install gitpython
!pip install matplotlib
!pip install seaborn
!pip install pandas
!pip install xgboost
!pip install git
!pip install sklearn
!pip install scikit-learn
```

12.1.2.2.2 Ejecución

```
Requirement already satisfied: gitpython in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (3.1.44)
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from gitpython) (4.0.12)
Requirement already satisfied: smmap<6,>=3.0.1 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from gitdb<5,>=4.0.1->gitpython) (5.0.2)
Requirement already satisfied: matplotlib in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (3.10.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied:ycler>=0.10 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: numpy>=1.23 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (2.2.6)
Requirement already satisfied: packaging>=20.0 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Requirement already satisfied: seaborn in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from seaborn) (2.2.6)
Requirement already satisfied: pandas>=1.2 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib>=3.6.1,>=3.4 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from seaborn) (3.10.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib>=3.6.1,>=3.4->seaborn) (1.3.2)
Requirement already satisfied:ycler>=0.10 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib>=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\isard\desktop\project_energy_bcn\venv\lib\site-packages (from matplotlib>=3.6.1,>=3.4->seaborn) (4.58.0)
```

12.1.2.3 Importación de librerías

A continuación se muestran las librerías importadas con una breve descripción de su utilidad y separadas por categorías de uso:

12.1.2.3.1 Código

```
# Manejo de datos
import pandas as pd #
Manipulación y análisis de datos
import numpy as np #
Operaciones matemáticas y manejo de arrays
# Visualización
import matplotlib.pyplot as plt #
Gráficos estáticos
import seaborn as sns #
Visualización estadística basada en matplotlib
# Modelado y aprendizaje automático
import xgboost as xgb #
Framework eficiente de gradient boosting
# Modelos de regresión de Scikit-learn
from sklearn.ensemble import (
    RandomForestRegressor, #
Bosques aleatorios para regresión
    GradientBoostingRegressor, #
Árboles potenciados por gradiente
    ExtraTreesRegressor, #
Árboles extra para regresión
    HistGradientBoostingRegressor #
Gradient boosting basado en histogramas
)
```

```
from sklearn.linear_model import LinearRegression, ElasticNet #  
Modelos de regresión lineal y regularizada  
from sklearn.svm import SVR #  
Máquina de soporte vectorial para regresión  
from sklearn.neighbors import KNeighborsRegressor #  
Regresión basada en vecinos cercanos  
from sklearn.metrics import mean_absolute_error, mean_squared_error #  
Métricas de evaluación  
# Control de advertencias  
import warnings #  
Gestión de advertencias  
warnings.filterwarnings('ignore') #  
Ignora todas las advertencias  
# Gestión de archivos y sistema  
import os #  
Interacción con el sistema operativo  
import shutil #  
Operaciones de alto nivel con archivos y carpetas  
import stat #  
Manejo de permisos y estados de archivos  
# Control de versiones  
import git #  
Interacción con repositorios Git  
# Utilidades adicionales  
import traceback #  
Manejo y depuración de errores  
import joblib #  
Guardado y carga de modelos
```

12.1.3 Recogida de campos necesarios

Una vez instaladas e importadas las librerías que utilizaremos, cargamos los conjuntos de datos originales. Estos conjuntos de datos son publicados por el Ayuntamiento de Barcelona:

<https://opendata-ajuntament.barcelona.cat/data/es/dataset/consum-electricitat-bcn>

Pero contienen diversas columnas que no me interesan, como el caso del código postal y el año, para ello a través del código vamos a obtener únicamente las columnas que me interesan.

| _id | Any | Data | Codi_Postal | Sector_Eco... | Tram_Horari | Valor |
|-----|------|-------------|-------------|---------------|-----------------|-------|
| 1 | 2020 | 2020-03-... | 8024 | Indústria | De 00:00:00 ... | 1558 |
| 2 | 2020 | 2020-03-... | 8024 | Indústria | De 06:00:00 ... | 2965 |
| 3 | 2020 | 2020-03-... | 8024 | Indústria | De 12:00:00 ... | 3326 |
| 4 | 2020 | 2020-03-... | 8024 | Indústria | De 18:00:00 ... | 2556 |
| 5 | 2020 | 2020-03-... | 8024 | Indústria | No consta | 0 |

12.1.3.1 Limpieza de datos

```
def sumar_consumo_por_dia_horario(csv_path):
    # Leer el CSV con las columnas necesarias
    df = pd.read_csv(csv_path, usecols=['id', 'data', 'tramo horario',
    'valor', 'sector_economic'])

    # Convertir 'data' a datetime
    df['data'] = pd.to_datetime(df['data'])

    # Extraer solo la fecha (sin la hora)
    df['fecha'] = df['data'].dt.date

    # Agrupar por fecha, sector económico y tramo horario, sumando
    'valor'
    suma_agrupada = df.groupby(['fecha', 'sector_economic', 'tramo
    horario'])['valor'].sum().reset_index()

    return suma_agrupada

if __name__ == "__main__":
    años = list(range(2019, 2026))

    for año in años:
        ruta_csv = input(f"Por favor, introduce la ruta del archivo CSV
        para el año {año}: ")
        resultado = sumar_consumo_por_dia_horario(ruta_csv)

        nombre_salida = f"{año}_consum_electricitat_bcn.csv"
        resultado.to_csv(nombre_salida, index=False)

        print(f"Datos procesados para el año {año} guardados en
        {nombre_salida}")
```

12.1.3.2 Explicación

Dentro de la función `sumar_consumo_por_dia_horario(csv_path)` nos encontramos con

```
# Leer el CSV con las columnas necesarias
df = pd.read_csv(csv_path, usecols=['id', 'data', 'tramo horario',
'valor', 'sector_economic'])
```

Que lee un archivo CSV de la ruta que le pases y carga sólo las columnas 'id', 'data', 'tramo horario', 'valor' y 'sector_economic' en un DataFrame llamado df.

Tras esto, convierte la columna data a datetime (fecha y hora):

```
# Convertir 'data' a datetime
df['data'] = pd.to_datetime(df['data'])
```

A continuación extrae la fecha sin hora:

```
# Extraer solo la fecha (sin la hora)
df['fecha'] = df['data'].dt.date
```

Crea una nueva columna 'fecha' que contiene sólo la fecha (sin la hora) extraída de la columna 'data'.

Después agrupa y suma los valores en función a la fecha, tramo horario y sector económico

```
# Agrupar por fecha, sector económico y tramo horario, sumando 'valor'
suma_agrupada = df.groupby(['fecha', 'sector_economic', 'tramo
horario'])['valor'].sum().reset_index()
```

Para cada grupo, suma los valores de la columna 'valor'.

Finalmente, `.reset_index()` convierte el índice resultante en columnas normales, para que el resultado sea un DataFrame limpio.

Tras esto lo ejecuta y define el rango de años a procesar, en nuestro caso desde el origen a la fecha actual (2019 a 2025), y guarda el nuevo valor:

```
if __name__ == "__main__":
    años = list(range(2019, 2026))

    for año in años:
        ruta_csv = input(f"Por favor, introduce la ruta del archivo CSV
para el año {año}: ")
        resultado = sumar_consumo_por_dia_horario(ruta_csv)

        nombre_salida = f"{año}_consum_electricitat_bcn.csv"
        resultado.to_csv(nombre_salida, index=False)
```

```
print(f"Datos procesados para el año {año} guardados en  
{nombre_salida}")
```

12.1.3.3 Permisos y carga de datos

Tras este filtro, subí los datos limpios a git y los recargaré desde ahí para trabajar con ellos normalizados

```
def on_rm_error(func, path, exc_info):  
    # Cambia el permiso y reintenta  
    print(f"Permiso denegado en: {path}. Cambiando permisos y  
reintentando...")  
    os.chmod(path, stat.S_IWRITE)  
    func(path)  
  
repo_path = "IaBigData"  
  
if os.path.exists(repo_path):  
    try:  
        print(f"Eliminando carpeta existente: {repo_path}")  
        shutil.rmtree(repo_path, onerror=on_rm_error)  
        print("Carpeta eliminada correctamente.")  
    except Exception as e:  
        raise RuntimeError(f"No se pudo eliminar la carpeta  
{repo_path}: {e}")  
  
Eliminando carpeta existente: IaBigData  
Permiso denegado en: IaBigData\.git\objects\pack\pack-b5b44a13d95429e1880c2250e709c8171a15a024.idx. Cambiando  
Permiso denegado en: IaBigData\.git\objects\pack\pack-b5b44a13d95429e1880c2250e709c8171a15a024.pack. Cambiando  
Permiso denegado en: IaBigData\.git\objects\pack\pack-b5b44a13d95429e1880c2250e709c8171a15a024.rev. Cambiando  
Carpeta eliminada correctamente.
```

12.1.3.4 Explicación

La función `on_rm_error` se utiliza para manejar errores que pueden ocurrir al eliminar archivos o carpetas. Si durante la eliminación se produce un error por falta de permisos (`PermissionError`), esta función cambia los permisos del archivo o carpeta para permitir la escritura y luego vuelve a intentar eliminarlo.

La variable `repo_path` define la ruta de la carpeta que se desea eliminar, en este caso "IaBigData". Primero, el código verifica si esta carpeta existe usando `os.path.exists(repo_path)`. Si la carpeta está presente, se intenta eliminarla con `shutil.rmtree(repo_path, onerror=on_rm_error)`, aprovechando la función `on_rm_error` para solucionar posibles problemas de permisos durante el proceso.

Si la eliminación es exitosa, se imprime un mensaje que confirma que la carpeta fue borrada correctamente. En caso de que ocurra algún error que no pueda ser resuelto, el programa lanza una excepción indicando el motivo del fallo.

12.1.3.5 Clonado y normalización

```
# Ruta donde se va a clonar el repositorio
repo_path = "IaBigData"

# Si ya existe la carpeta, la eliminamos para sobrescribirla
if os.path.exists(repo_path):
    print(f"Eliminando carpeta existente: {repo_path}")
    shutil.rmtree(repo_path)

# Clonamos el repositorio de Github con manejo de errores
try:
    print("Clonando el repositorio...")

    git.Repo.clone_from("https://github.com/AdrianYArmas/IaBigData.git",
                        repo_path)
    print("Repositorio clonado correctamente.")
except Exception as e:
    raise RuntimeError(f"Error al clonar el repositorio: {e}")

# Ruta a la subcarpeta deseada
subfolder_path = os.path.join(repo_path, "TFG-MAED-Bcn", "dataset")

# Validar existencia de la subcarpeta
if not os.path.exists(subfolder_path):
    raise FileNotFoundError(f"La subcarpeta no existe: {subfolder_path}")

# Función para importar, agrupar y adaptar los datos
def primera(x):
    file_path = os.path.join(subfolder_path,
                              f"{x}_consum_electricitat_bcn.csv")

    if not os.path.isfile(file_path):
        raise FileNotFoundError(f"No se encontró el archivo: {file_path}")

    try:
```

```
data = pd.read_csv(file_path)
except Exception as e:
    raise RuntimeError(f"Error al leer el archivo CSV: {e}")

if 'Sector_Economic' not in data.columns or 'Data' not in
data.columns or 'Valor' not in data.columns:
    raise ValueError("El archivo CSV no contiene las columnas
necesarias: 'Sector_Economic', 'Data', 'Valor'")

data_agrupado = data.groupby(['Sector_Economic',
'Data'])['Valor'].sum().reset_index()
data_agrupado['Data'] = pd.to_datetime(data_agrupado['Data'],
errors='coerce')

if data_agrupado['Data'].isnull().any():
    raise ValueError("Se encontraron fechas inválidas en la columna
'Data'.")

return data_agrupado
```

12.1.3.6 Explicación

El código elimina una carpeta local si existe, clona un repositorio de GitHub en esa ruta, y verifica que exista una subcarpeta específica dentro del repositorio clonado. Luego define una función que, dado un identificador (como un año), busca un archivo CSV en esa subcarpeta, valida que tenga las columnas necesarias, lee y agrupa los datos por sector económico y fecha, y retorna el resultado procesado.

12.1.3.7 Comprobación de la estructura de datos

Pese a que el código anterior valida los datos, vuelvo a comprobar que la estructura de datos está correcta, esta vez de manera manual y visual.

```
# Llamado a la función para generar las tablas
data_2019 = primera(2019)
data_2020 = primera(2020)
data_2021 = primera(2021)
data_2022 = primera(2022)
data_2023 = primera(2023)
data_2024 = primera(2024)
data_2025 = primera(2025)

# Imprimimos las primeras filas de 2025
print("\ndata_2019")
```

```
print(data_2019.head())
print("\ndata_2020")
print(data_2020.head())
print("\ndata_2021")
print(data_2022.head())
print("\ndata_2022")
print(data_2022.head())
print("\ndata_2023")
print(data_2023.head())
print("\ndata_2024")
print(data_2024.head())
print("\ndata_2025")
print(data_2025.head())
```

```
data_2019
  Sector_Economic      Data      Valor
0      Industria 2019-01-01  1000881
1      Industria 2019-01-02  1702091
2      Industria 2019-01-03  1826076
3      Industria 2019-01-04  1806926
4      Industria 2019-01-05  1341548
```

```
data_2020
  Sector_Economic      Data      Valor
0      Industria 2020-01-01  1840619
1      Industria 2020-01-02  2530835
2      Industria 2020-01-03  2742286
3      Industria 2020-01-04  2230244
4      Industria 2020-01-05  2008513
```

```
data_2021
  Sector_Economic      Data      Valor
0      Industria 2022-01-01   964700
1      Industria 2022-01-02  1019860
2      Industria 2022-01-03  1393808
3      Industria 2022-01-04  1371396
4      Industria 2022-01-05  1354421
...
1      Industria 2025-01-02   895556
2      Industria 2025-01-03   970954
3      Industria 2025-01-04   774321
4      Industria 2025-01-05   707140
```

Como podemos observar, todos los csv tienen la misma estructura.

12.1.4 Análisis exploratorio de los datos (EDA)

12.1.4.1 Unión de los csv

El primer paso es unificar los datos en un solo archivo, ya que hasta ahora están en archivos separados por años. Para ello concatenamos los csv con el siguiente código:

```
# Concatenamos de forma vertical las tablas de datos obtenidas en el
punto anterior en una sola
```

```
data = pd.concat([data_2019, data_2020, data_2021, data_2022,
data_2023, data_2024, data_2025])

# Imprimimos en pantalla la principal información de la tabla de datos
resultante
data.info()
```

Tras ejecutarlo podemos ver que ahora solo nos devuelve un dataset:

```
<class 'pandas.core.frame.DataFrame'>
Index: 8426 entries, 0 to 176
Data columns (total 3 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Sector_Economic 8426 non-null   object
1   Data             8426 non-null   datetime64[ns]
2   Valor            8426 non-null   int64
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 263.3+ KB
```

12.1.4.2 Traducción de las columnas

Ya que el dataset original está en catalán, renombré las variables y algunos de sus registros, luego agrupamos los datos por consumo total diario. Finalmente, generamos un archivo con la tabla resultante en formato .csv.

```
# Modificamos el nombre de las variables (catalan -> español)
data.rename(columns = {"Data":"fecha","Valor":"valor",
"Sector_Economic":"sector_economic"}, inplace=True)

# Agrupamos la tabla de datos por consumo total por día
data = data.groupby(['fecha']).sum().reset_index()

# Guardamos la tabla de datos
data.to_csv("tabla_datos_exploratorio.csv")
```

12.1.4.3 Detección de outliers

Una vez unificadas las columnas y organizados los datos por fecha, el siguiente paso es corregir los valores atípicos para mejorar la precisión del modelo.

Para identificar numéricamente en qué fechas ocurren estos outliers, utilizaremos un método común basado en los cuartiles y el rango intercuartílico (IQR). Los cuartiles dividen los datos en partes, y el IQR representa la distancia entre el primer cuartil (Q1) y el tercer cuartil (Q3), es decir:

$$\text{IQR} = Q3 - Q1$$

Este rango mide la dispersión central de los datos, abarcando el 50% medio de los valores. Con esta información, se establecen límites para detectar valores atípicos, definidos como aquellos que se encuentran fuera del rango:

Límite inferior: $Q1 - 1.5 * \text{IQR}$

Límite superior: $Q3 + 1.5 * \text{IQR}$

Los datos que caen por debajo o por encima de estos límites son considerados outliers.

```
# Outlier cleaning
Q1 = data['valor'].quantile(0.25)
Q3 = data['valor'].quantile(0.75)
IQR = Q3 - Q1
lim_inf = Q1 - 1.5 * IQR
lim_sup = Q3 + 1.5 * IQR
outliers = data[(data['valor'] < lim_inf) | (data['valor'] > lim_sup)]

data['fecha'] = pd.to_datetime(data['fecha'])
data = data.sort_values('fecha').reset_index(drop=True)
print(outliers)
```

| | fecha | sector_economico | valor |
|-----|------------|---|----------|
| 178 | 2019-06-28 | IndústriaNo especificatResidencialServeis | 23762598 |
| 182 | 2019-07-02 | IndústriaNo especificatResidencialServeis | 23675821 |
| 183 | 2019-07-03 | IndústriaNo especificatResidencialServeis | 23855532 |
| 184 | 2019-07-04 | IndústriaNo especificatResidencialServeis | 23845642 |
| 185 | 2019-07-05 | IndústriaNo especificatResidencialServeis | 23924322 |
| 192 | 2019-07-12 | IndústriaNo especificatResidencialServeis | 23670782 |
| 202 | 2019-07-22 | IndústriaNo especificatResidencialServeis | 23821450 |
| 203 | 2019-07-23 | IndústriaNo especificatResidencialServeis | 24377531 |
| 204 | 2019-07-24 | IndústriaNo especificatResidencialServeis | 24621090 |
| 205 | 2019-07-25 | IndústriaNo especificatResidencialServeis | 24569613 |
| 206 | 2019-07-26 | IndústriaNo especificatResidencialServeis | 23801202 |
| 307 | 2019-11-04 | IndústriaNo especificatResidencialServeis | 24088180 |
| 308 | 2019-11-05 | IndústriaNo especificatResidencialServeis | 24304325 |
| 309 | 2019-11-06 | IndústriaNo especificatResidencialServeis | 24172827 |

12.1.4.4 Inserción de datos válidos en los outliers de los cuartiles

En este caso en vez de eliminar esos valores atípicos, he decidido realizar una modificación los outliers detectados con los cuartiles sustituyendo el valor por la media del registrado el día anterior y el día siguiente.

```
# Aseguramos que la columna de fecha esté en formato datetime
data['fecha'] = pd.to_datetime(data['fecha'])
```

```
data = data.sort_values('fecha').reset_index(drop=True)

# Sustituimos cada outlier por la media entre la fila anterior y
siguiente
for idx, outlier_row in outliers.iterrows():
    fecha_outlier = outlier_row['fecha']

    # Filtramos filas anteriores y siguientes
    anteriores = data[data['fecha'] < fecha_outlier]
    siguientes = data[data['fecha'] > fecha_outlier]

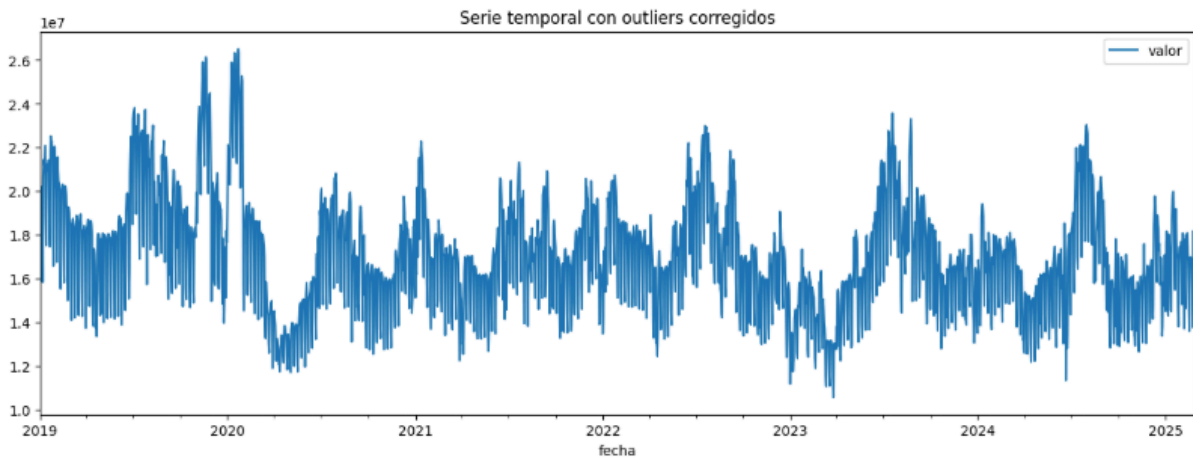
    # Verificamos que existan ambas para calcular la media
    if not anteriores.empty and not siguientes.empty:
        valor_anterior = anteriores.iloc[-1]['valor']
        valor_siguiente = siguientes.iloc[0]['valor']
        media = (valor_anterior + valor_siguiente) / 2

        # Reemplazamos el valor del outlier
        data.loc[data['fecha'] == fecha_outlier, 'valor'] = media
    else:
        print(f"No se puede reemplazar el outlier del
{fecha_outlier.date()} por falta de datos vecinos.")

# Ploteamos para verificar visualmente los cambios
data.plot(figsize=(15,5), x='fecha', y='valor', linestyle='-')
plt.title('Serie temporal con outliers corregidos')
```

El código convierte la columna de fechas al formato datetime y ordena los datos por fecha. Luego, para cada valor atípico (outlier), reemplaza su valor por la media entre el valor anterior y el siguiente en la serie temporal, siempre que existan ambos valores vecinos. Si no hay valores vecinos disponibles, no hace el reemplazo y avisa. Finalmente, grafica la serie temporal con los valores corregidos para verificar los cambios visualmente.

Y tras todo esto muestra la siguiente serie temporal sin los outliers:



12.1.5 Entrenamiento del modelo

12.1.5.1 Creación de atributos temporales

La creación de atributos temporales tiene como objetivo transformar la variable de fecha en características numéricas que permitan capturar patrones importantes en la serie temporal. Estos atributos, como el día de la semana, mes, trimestre o año, ayudan a que el modelo identifique ciclos y comportamientos recurrentes que dependen del tiempo, como variaciones semanales, estacionales o anuales.

Además, al combinar estos atributos con variables derivadas de la serie, como los valores rezagados (lags) y medias móviles, se enriquece la información disponible para el modelo. Esto facilita que pueda aprender no solo cómo varía la variable objetivo en función del tiempo calendario, sino también cómo influye su comportamiento reciente.

En resumen, la creación de atributos temporales permite aprovechar la dimensión temporal de los datos para mejorar la capacidad predictiva y la interpretación de los modelos de machine learning aplicados a series temporales.

```
# La variable fecha la convertimos al índice de la tabla de datos
data = data.set_index('fecha')
```

Para ello, la línea `data = data.set_index('fecha')` convierte la columna 'fecha' en el índice del DataFrame. Esto significa que la fecha deja de ser solo una columna más y pasa a ser la referencia principal para identificar cada fila. Al establecer la fecha como índice, se facilita el manejo de la información temporal, permitiendo extraer fácilmente atributos como día, mes o año y aplicar operaciones propias de series temporales. Además, este cambio es fundamental para organizar y trabajar con los datos de manera cronológica, lo que mejora la eficiencia en análisis y modelado.

```
# Definimos una función para crear nuevos atributos a partir del índice de fecha
def creacion_atributos(df):
    df['dia_semana'] = df.index.dayofweek      # Día de la semana (0 = lunes, 6 = domingo)
    df['dia_año'] = df.index.dayofyear         # Día del año (1 a 365/366)
    df['trimestre'] = df.index.quarter         # Trimestre del año (1 a 4)
    df['mes'] = df.index.month                 # Mes del año (1 a 12)
    df['año'] = df.index.year                  # Año (por ejemplo, 2025)
    return df

# Aplicamos la función al DataFrame para añadir los nuevos atributos basados en la fecha
data_total = creacion_atributos(data)
```

Posteriormente la función `creacion_atributos` se encarga de generar nuevos atributos a partir del índice de fecha del DataFrame. Al estar la fecha como índice, esta función extrae información relevante como el día de la semana, el día del año, el trimestre, el mes y el año, y añade estas características como columnas nuevas en el DataFrame. Esto enriquece los datos con variables temporales que facilitan la identificación de patrones estacionales y cíclicos. Finalmente, al aplicar esta función, el DataFrame `data_total` queda con estos nuevos atributos incorporados, listos para ser usados en el análisis y modelado.

12.1.5.2 Datos exploratorios junto a los atributos temporales

En esta parte se crean variables históricas que ayudan al modelo a capturar patrones temporales: se añaden valores rezagados (`valor_lag_1` y `valor_lag_7`) y medias móviles de 7 y 30 días para suavizar la serie. Luego, se eliminan las filas con datos faltantes generados por estos cálculos. Finalmente, se realizan gráficos exploratorios tipo boxplot para visualizar cómo varía la variable objetivo según diferentes atributos temporales, facilitando la identificación de patrones estacionales.

```
# Añadir lags y medias móviles
data_total["valor_lag_1"] = data_total["valor"].shift(1)
data_total["valor_lag_7"] = data_total["valor"].shift(7)
data_total["media_7"] = data_total["valor"].rolling(window=7).mean()
data_total["media_30"] = data_total["valor"].rolling(window=30).mean()

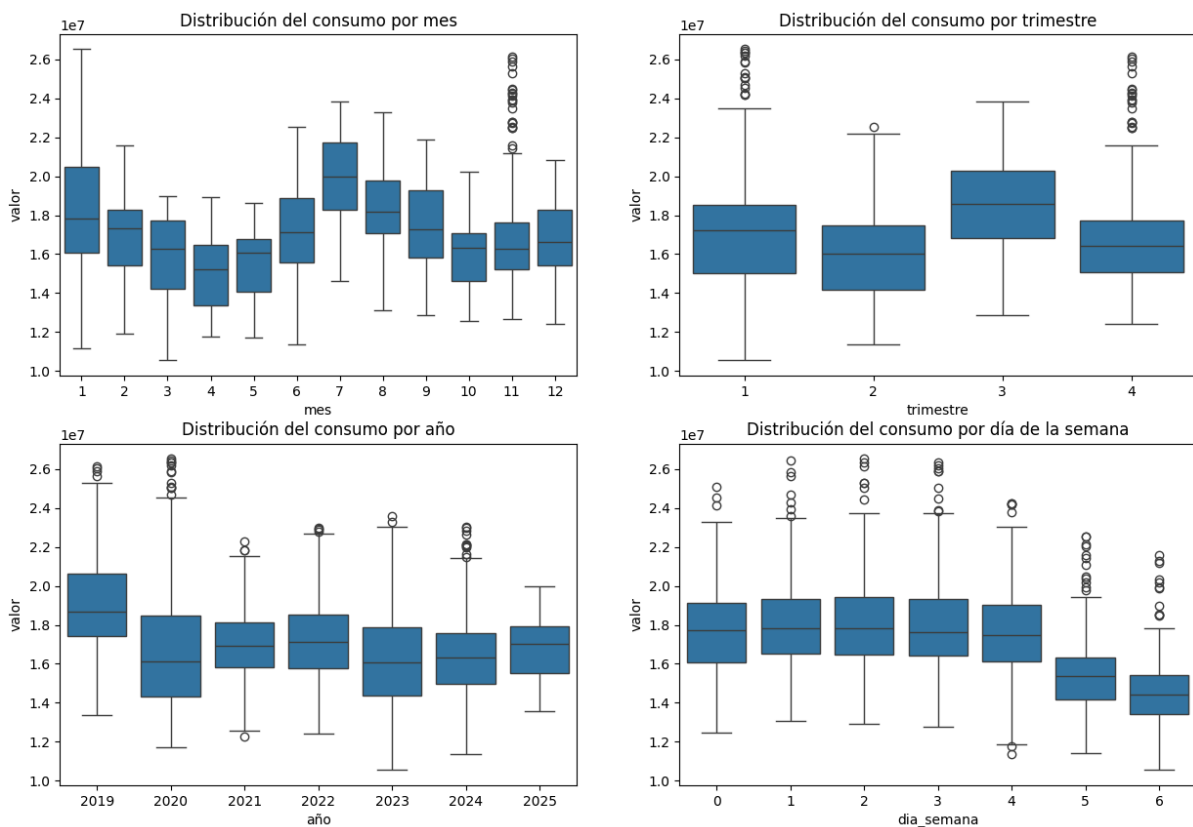
# Eliminar filas con NaN
data_total = data_total.dropna()

# Visualización exploratoria
```



```
fig, axes = plt.subplots(2,2, figsize=(15,10))
sns.boxplot(data=data_total, x="mes", y="valor", ax=axes[0,0])
sns.boxplot(data=data_total, x="trimestre", y="valor", ax=axes[0,1])
sns.boxplot(data=data_total, x="año", y="valor", ax=axes[1,0])
sns.boxplot(data=data_total, x="dia_semana", y="valor", ax=axes[1,1])
plt.tight_layout()
plt.show()
```

En estos gráficos podemos observar las distribuciones de consumo por día de la semana, mes, trimestre y año.



Podemos observar varias tendencias importantes: julio es el mes con mayor consumo, mientras que los fines de semana el consumo disminuye. El tercer trimestre, que incluye a julio, es el período con el mayor gasto energético. Además, la tendencia promedio de consumo ha ido disminuyendo desde 2019, lo cual refleja el decrecimiento de la industria y su impacto en el consumo energético total. Un valor no tan estimable, ya que el consumo a nivel personal, fuera del ámbito profesional, aumentó en 2020, coincidiendo con el período de la pandemia de COVID-19. artículos como el de <http://energética21.com>*¹⁶ lo avalan.

12.1.5.3 División del dataset

División del dataset en 70% train y 30% test, todo esto con una estimación gráfica del equivalente al 70 y 30.

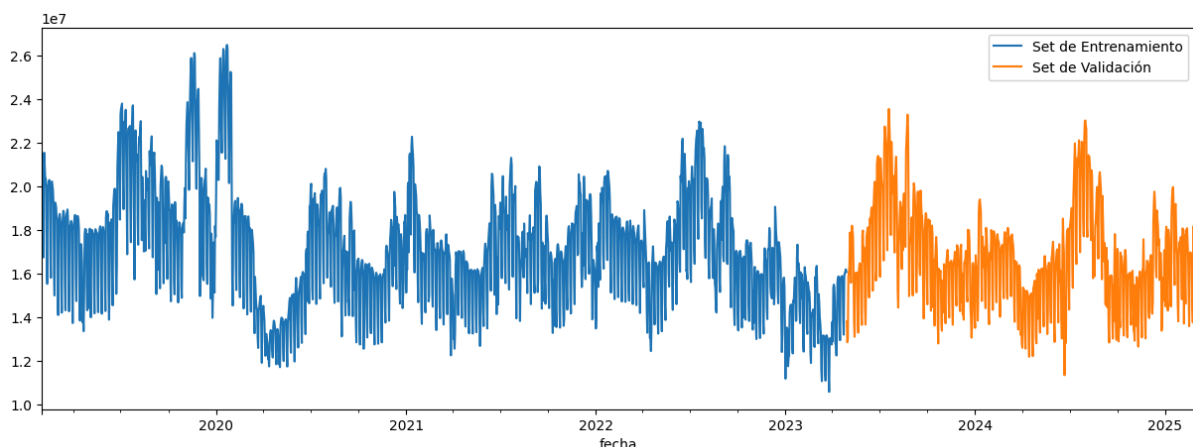
Para ello definí la división del 70 y 30% manualmente simplemente para ver su equivalencia en un modo de prueba:

```
# División de datos
trainTest = data_total.loc[data_total.index <
pd.to_datetime('2023-04-29')]
testTest = data_total.loc[data_total.index >=
pd.to_datetime('2023-04-29')]
```

Y lo mostré de manera gráfica:

```
# Ploteamos los dos subsets creados anteriormente en la misma serie
temporal
fig, ax = plt.subplots(figsize=(15,5))
trainTest.plot(ax=ax, y='valor', label='Set de Entrenamiento')
testTest.plot(ax=ax, y='valor', label='Set de Validación')
```

El fin de este paso previo es interpretar de manera gráfica si la división de datos es suficiente o excesivo.



En mi caso me pareció correcto así que hice la selección de la división de datos aleatoria pero respetando ese 70%-30%:

```
# División de datos
train, test = train_test_split(data_total, test_size=0.3,
random_state=42, shuffle=True)
```

12.1.5.4 Definición de los atributos utilizados y la variable objetivo

A continuación, definí los atributos de entrada (X) y la variable objetivo (Y). Para ello, utilicé los datos visualizados previamente como las variables predictoras (X) y la columna `valor` como la variable a predecir (Y).

```
# Creacion de una lista con las variables atributos y la variable
objetivo
atributos = [
    'dia_semana',    # día semana ciclo
    'mes',           # mes calendario anual
    'año',           # año calendario largo
    'dia_año',       # día año ordinal
    'trimestre',     # trimestre periodo anual
    'valor_lag_1',   # valor día-anterior
    'valor_lag_7',   # valor semana-anterior
    'media_7',       # media última-semana
    'media_30'       # media último-mes
]

objetivo = 'valor'  # dato a predecir
```

Y posteriormente preparé los datos para que el modelo pueda recibir las entradas (X_train) y las salidas/etiquetas (y_train) para entrenar, y luego usar las entradas de test (X_test) y las etiquetas reales (y_test) para evaluar cómo le fue:

```
# Dividimos los sets de entrenamiento y validación según las listas
creadas anteriormente
X_train = train[atributos]
y_train = train[objetivo]
X_test = test[atributos]
y_test = test[objetivo]
```

12.1.5.5 Comparación de modelos

Este apartado, muestra estadísticas descriptivas básicas de la variable objetivo (y_test), que es el conjunto de valores reales para el test, e imprime el valor promedio, mínimo y máximo de la variable objetivo, para entender la escala y rango de los datos que queremos predecir.

```
# Mostrar escala de los datos
print(f"Valor promedio: {y_test.mean():,.0f}")
print(f"Valor mínimo: {y_test.min():,.0f}")
print(f"Valor máximo: {y_test.max():,.0f}")

Valor promedio: 16,774,815
Valor mínimo: 11,349,835
Valor máximo: 23,568,214
```

Tras esto, define la batería de modelos que va a entrenar a través de un diccionario con varios modelos de regresión para probar cuál funciona mejor, posteriormente, cada modelo tiene sus hiperparámetros configurados (por ejemplo, número de árboles, profundidad máxima, tasa de aprendizaje, etc.).

Modelos usados: XGBoost, RandomForest, GradientBoosting, LinearRegression, ExtraTrees, HistGradientBoosting, SVR, KNeighbors y ElasticNet.

```
# Modelos
modelos = {
    "XGBoost": xgb.XGBRegressor(
        n_estimators=100,
        early_stopping_rounds=100,
        learning_rate=0.3,
        max_depth=6,
        min_child_weight=1,
        gamma=0,
        colsample_bytree=1,
        subsample=1,
        objective="reg:squarederror",
        random_state=42
    ),
    "RandomForest": RandomForestRegressor(n_estimators=200,
max_depth=10, random_state=42),
    "GradientBoosting": GradientBoostingRegressor(n_estimators=300,
learning_rate=0.05, max_depth=6, random_state=42),
    "LinearRegression": LinearRegression(),
    "ExtraTrees": ExtraTreesRegressor(n_estimators=200, max_depth=10,
random_state=42),
    "HistGradientBoosting": HistGradientBoostingRegressor(max_iter=200,
max_depth=10, random_state=42),
    "SVR": SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1),
    "KNeighbors": KNeighborsRegressor(n_neighbors=5),
    "ElasticNet": ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=42)
}
```

Tras esto se entrena y evalúa cada modelo guardando los valores del mae, maps y rmse:

```
resultados = []

for nombre, modelo in modelos.items():
    try:
```

```
print(f"\nEntrenando modelo: {nombre}")
if "XGB" in nombre:
    modelo.fit(X_train, y_train, eval_set=[(X_train, y_train),
(X_test, y_test)], verbose=False)
else:
    modelo.fit(X_train, y_train)

predicciones = modelo.predict(X_test)
mae = mean_absolute_error(y_test, predicciones)
rmse = np.sqrt(mean_squared_error(y_test, predicciones))
mape = np.mean(np.abs((y_test - predicciones) / y_test)) * 100

resultados.append({
    "modelo": nombre,
    "MAE": mae,
    "RMSE": rmse,
    "MAPE (%)": mape
})

except Exception as e:
    print(f"Error entrenando {nombre}: {e}")
    traceback.print_exc()
```

Entrenando modelo: XGBoost

Entrenando modelo: RandomForest

Entrenando modelo: GradientBoosting

Entrenando modelo: LinearRegression

Entrenando modelo: ExtraTrees

Entrenando modelo: HistGradientBoosting

Entrenando modelo: SVR

Entrenando modelo: KNeighbors

Entrenando modelo: ElasticNet

Tras esto se muestra el resultado:

```
# Mostrar resultados
resultados_df = pd.DataFrame(resultados).sort_values("RMSE")
print("\n📊 Comparación de Modelos:")
print(resultados_df)
```

```
📊 Comparación de Modelos:
```

| | modelo | MAE | RMSE | MAPE (%) |
|-----|------------|--------------|--------------|-----------|
| 4 | ExtraTrees | 5.305703e+05 | 7.944687e+05 | 3.253496 |
| ... | | | | |
| 7 | KNeighbors | 7.698103e+05 | 1.152389e+06 | 4.667593 |
| 6 | SVR | 1.791276e+06 | 2.285623e+06 | 10.951811 |

Y tras la comparación se muestra el mejor resultado:

```
# Mejor modelo
mejor_modelo = resultados_df.iloc[0]['modelo']
print(f"\n✅ Mejor modelo: {mejor_modelo}")
```

```
✅ Mejor modelo: XGBoost
```

12.1.5.6 Importancia de las variables en el entrenamiento

Finalmente, en esta sección, mostramos un gráfico que indica qué tan importante fue cada atributo temporal para que el modelo aprendiera y hiciera sus predicciones.

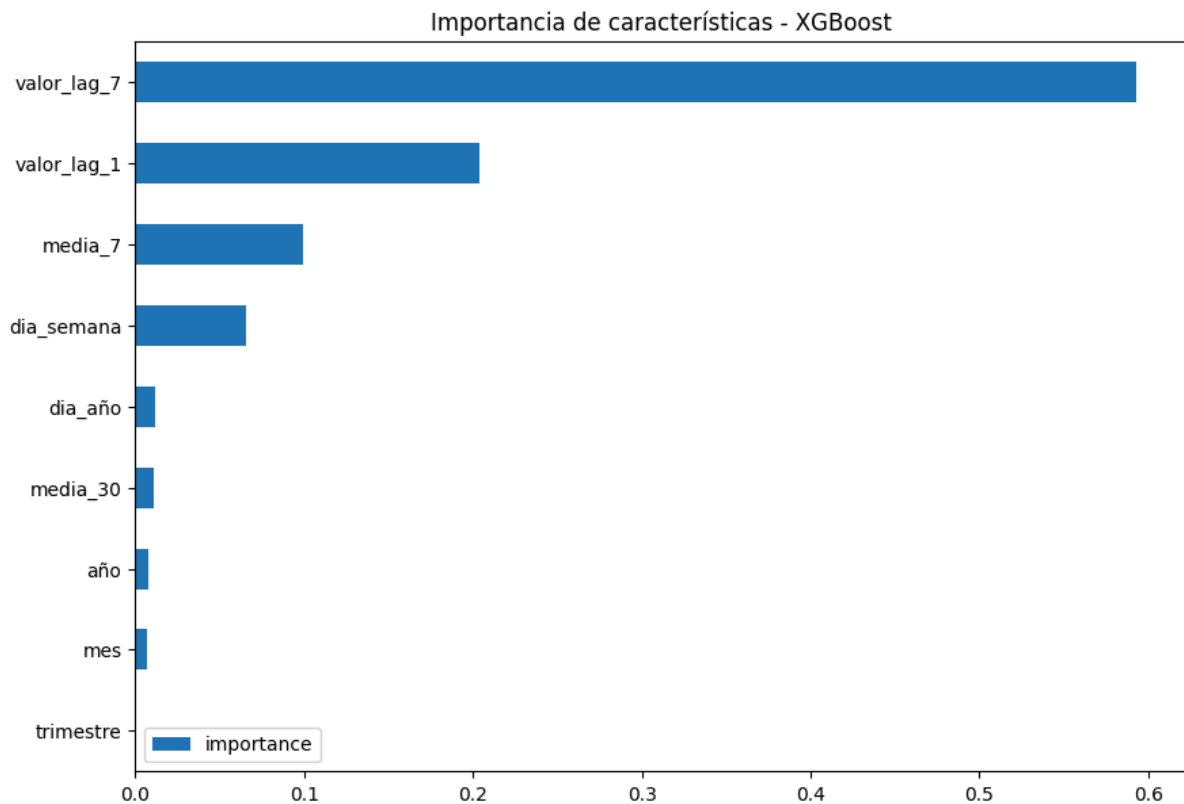
```
# Obtener el modelo XGBoost ya entrenado
reg = modelos["XGBoost"]

# Crear DataFrame de importancias
fi = pd.DataFrame(
    data=reg.feature_importances_,
    index=reg.feature_names_in_,
    columns=['importance']
)

# Graficar
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10, 7))
fi.sort_values('importance').plot(kind='barh', ax=ax)
```

```
plt.title("Importancia de características - XGBoost")
plt.show()
```



12.1.6 Validación del modelo

12.1.6.1 Gráfica

Una vez entrenado el modelo y entendido su funcionamiento, generamos una nueva columna con las predicciones del valor de demanda eléctrica y la añadimos a la tabla de datos inicial.

```
# Predicciones para el año 2022
test['prediccion'] = reg.predict(X_test)

# Añadir a la tabla de datos las predicciones obtenidas
data_total = data_total.merge(test[['prediccion']], how="left",
left_index=True, right_index=True)
```

Y mostramos la comparativa entre predicción y valor real:

```
# Predicciones para el año 2022
test['prediccion'] = reg.predict(X_test)

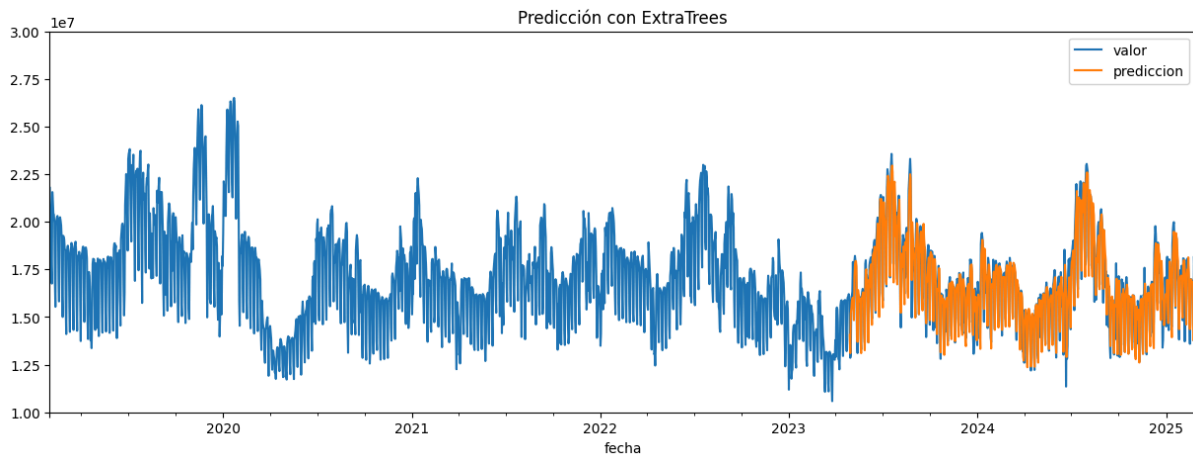
# Añadir a la tabla de datos las predicciones obtenidas
```

```
data_total = data_total.merge(test[['prediccion']], how="left",
left_index=True, right_index=True)
```

```
# Predicción y visualización
modelo_final = modelos[mejor_modelo]
test = test.copy()
test['prediccion'] = modelo_final.predict(X_test)
data_total = data_total.merge(test[['prediccion']], how="left",
left_index=True, right_index=True)

# Ploteamos el valor real de la demanda eléctrica junto a la predicción
para tener una referencia visual de lo acertado que es el modelo
ax = data_total[['valor']].plot(figsize=(15,5), label="Real")
ax.set_ylim(10000000, 30000000)
data_total[['prediccion']].plot(ax=ax, style='-', label="Predicción")
plt.legend()
plt.title(f"Predicción con {mejor_modelo}")
plt.show()
```

Como podemos observar, los valores se ajustan de manera bastante precisa sobre el histórico:



12.1.6.2 Métrica de precisión

A continuación, se calcula el Error Porcentual Absoluto Medio (MAPE – Mean Absolute Percentage Error), que se utiliza como métrica para evaluar la precisión de las predicciones en series temporales.

```
# Error porcentual absoluto medio (MAPE – Mean Absolute Percentage
Error) como métrica de evaluación de las predicciones de serie temporal
test["Porcentaje"] = (abs(test["prediccion"] - test["valor"]) /
test["valor"]) * 100
test["Porcentaje"].mean()
```


Como se puede observar este es el error absoluto medio:

```
np.float64(3.540729347275893)
```

Calculado de manera más clara sería:

```
# Calcula el porcentaje de error absoluto para cada fila
test["Porcentaje"] = (abs(test["prediccion"] - test["valor"]) /
test["valor"]) * 100

# Calcula el promedio del porcentaje (MAPE)
mape = test["Porcentaje"].mean()

# Imprime el resultado con texto personalizado
print(f"Error porcentual absoluto medio: {mape:.2f}")
```

```
Error porcentual absoluto medio : 3.54
```

y si prefieres una visualización del acierto y no del fallo, sería:

```
# Calculamos el error porcentual absoluto medio (MAPE)
test["Porcentaje"] = (abs(test["prediccion"] - test["valor"]) /
test["valor"]) * 100

# Calculamos el porcentaje de acierto como 100 - MAPE
porcentaje_acierto = 100 - test["Porcentaje"].mean()

print(f"Porcentaje de acierto: {porcentaje_acierto:.2f}%")
```

```
Porcentaje de acierto: 96.46%
```

12.1.7 Predicción de valores futuros

12.1.7.1 Gráfica comparativa

Sobre una gráfica mostramos la comparativa total antes de hacer la predicción con el fin de corroborar la precisión de manera visual, esta vez con el intervalo total del histórico y no solo con un breve período de tiempo:

```
future = pd.date_range('2019-01-01', '2025-02-28')
df_future = pd.DataFrame(index=future)

# Creamos los atributos en el dataset
```

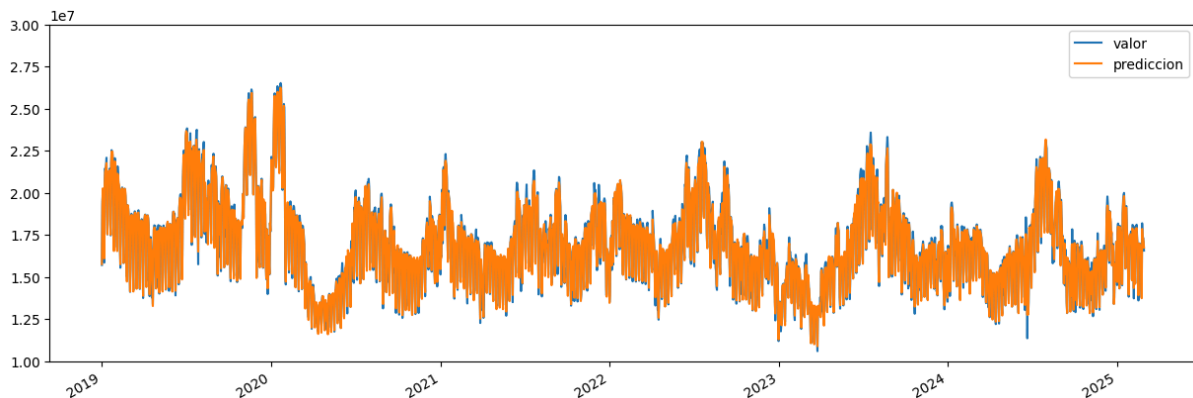
```
df_future = creacion_atributos(df_future)

# Juntamos la tabla de datos reales al dataset creado anteriormente
df_final = pd.concat([data_total, df_future])

# Generamos las predicciones para el año 2023 y las añadimos a la tabla
de datos iniciales.
df_future['prediccion']= reg.predict(df_future[atributos])
df_final = df_final.merge(df_future[['prediccion']], how="left",
left_index=True, right_index=True)

# Graficamos la serie temporal con los valores reales de consumo y la
predicción para el 2023
ax = df_final[['valor']].plot(figsize=(15, 5))
ax.set_ylim(10000000,30000000 )
df_future[['prediccion']].plot(ax=ax)
```

Como podemos observar en la gráfica se ajusta de manera casi perfecta, lo que confirma una alta precisión y que no hay sobreajuste.



12.1.7.2 Ejemplo de predicción futura

A continuación, crea un nuevo conjunto de datos con fechas futuras, genera sus atributos, une esos datos futuros con los reales, predice valores para esas fechas futuras usando un modelo entrenado, y finalmente grafica la serie temporal mostrando los valores reales y las predicciones para ese periodo futuro.

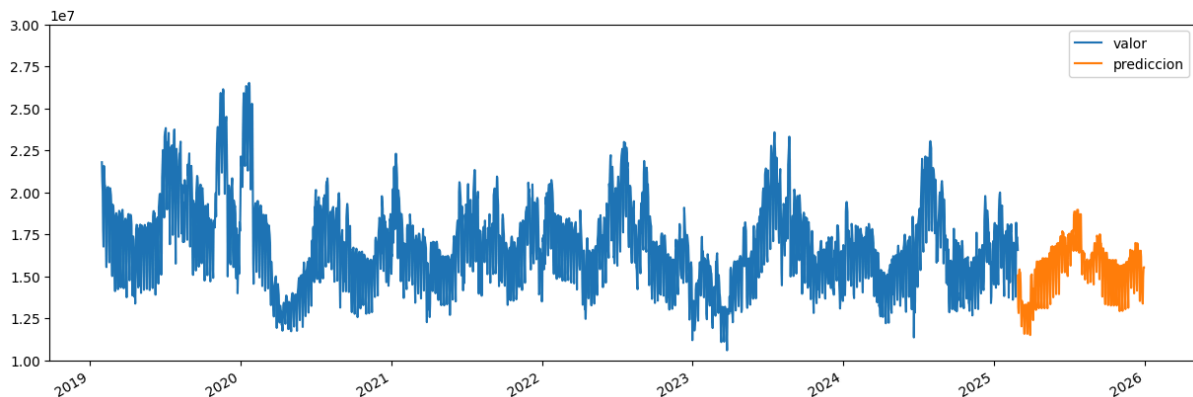
```
future = pd.date_range('2025-02-28', '2025-12-31')
df_future = pd.DataFrame(index=future)

# Creamos los atributos en el dataset
df_future = creacion_atributos(df_future)
```

```
# Juntamos la tabla de datos reales al dataset creado anteriormente
df_final = pd.concat([data_total, df_future])

# Generamos las predicciones para el año 2023 y las añadimos a la tabla
de datos iniciales.
df_future['prediccion']= reg.predict(df_future[atributos])
df_final = df_final.merge(df_future[['prediccion']], how="left",
left_index=True, right_index=True)

# Graficamos la serie temporal con los valores reales de consumo y la
predicción para el 2023
ax = df_final[['valor']].plot(figsize=(15, 5))
ax.set_ylim(10000000,300000000 )
df_future[['prediccion']].plot(ax=ax)
```



La serie predicha mantiene la continuidad con los datos históricos, respeta los patrones estacionales observados en años anteriores y se mantiene dentro de un rango coherente de valores. No presenta saltos ni distorsiones inesperadas, lo que sugiere que el modelo ha captado bien tanto la estacionalidad como la magnitud del consumo energético.

12.1.8 Guardado de los pesos y valor predicho

Por último, guardamos en una nueva tabla de datos los resultados de las predicciones, de cara a poder mostrarlas en una visualización interactiva. Además de los pesos del modelo y la comparativa de los modelos:

```
# Generamos y guardamos en un archivo .csv la tabla de datos con las
predicciones
df_future.to_csv("predictive_data.csv")

# Guardar modelo
joblib.dump(modelo_final, "../app/model.pkl")

# Guardar resultados
```

```
resultados_df.to_csv("comparacion_modelos.csv", index=False)
```

La comparativa de los modelos se guarda en comparacion_modelos.csv:

```
modelo,MAE,RMSE,MAPE (%)
XGBoost,559049.9830729166,790374.9996819036,3.429509012139291
GradientBoosting,551130.0813221367,795303.4647908674,3.385948158024691
HistGradientBoosting,562917.9035774459,804712.7699271175,3.438726734431763
RandomForest,563455.666630712,825492.0579636518,3.4561487455899895
ElasticNet,722464.9764515965,952608.286400266,4.357259676598643
LinearRegression,724494.409448088,954024.0257567847,4.370915340244461
KNeighbors,759089.9061662947,1134054.237938926,4.59862054207213
SVR,1786146.6765252976,2278986.197234547,10.907396702643554
```

El valor predicho en predictive_data.csv:

```
,dia_semana,dia_año,trimestre,mes,año
2025-02-28,4,59,1,2,2025
2025-03-01,5,60,1,3,2025
2025-03-02,6,61,1,3,2025
2025-03-03,0,62,1,3,2025
2025-03-04,1,63,1,3,2025
2025-03-05,2,64,1,3,2025
2025-03-06,3,65,1,3,2025
2025-03-07,4,66,1,3,2025
2025-03-08,5,67,1,3,2025
```

Los pesos los guarda en app/model.pkl:

```
▼ app
  > __pycache__
  > csv
  + api.py
  {} model.json
  ≡ model.pkl
```

12.1.9 Carga de los pesos

Este código carga los pesos guardados previamente en app/model.pkl:

```
# Cargar el modelo desde el archivo
modelo_cargado = joblib.load("../app/model.pkl")
```

12.2 Frontend

12.2.1 Resumen

Esta aplicación web, desarrollada con Streamlit, permite al usuario seleccionar un rango de fechas para visualizar predicciones de consumo energético diario. Tras enviar las fechas, la app realiza una petición al backend, recibe los datos predichos, los visualiza gráficamente y muestra un resumen estadístico con un diseño personalizado.

12.2.2 Estilos y CSS

Mediante ``st.markdown``, se inyecta código HTML y CSS para personalizar visualmente los componentes de Streamlit, logrando una interfaz más atractiva y diferenciada del estilo predeterminado. Se ajustan los títulos (``h1``) con un tamaño reducido y márgenes optimizados; los botones (``st.button``) se estilizan con un fondo azul, texto blanco, efectos al pasar el cursor y alineación centrada; las entradas de fecha (``st.date_input``) presentan un fondo oscuro con texto blanco para mayor contraste; y los cuadros de resultado (``result-box``) adoptan un diseño uniforme con bordes redondeados, sombra, fondo coloreado y contenido centrado.

12.2.3 Funcionalidad

El usuario comienza seleccionando una fecha de inicio y una fecha de fin, tras lo cual la aplicación valida que el rango sea correcto (es decir, que la fecha inicial no sea posterior a la final). Una vez validadas las fechas, al hacer clic en el botón "Mostrar Predicciones", se realiza una solicitud a la API (``/predict_range``) con los parámetros seleccionados.

Si la respuesta del servidor es válida, se visualiza un gráfico de líneas que muestra el consumo energético diario estimado para el período elegido. Además, la aplicación calcula y presenta varios indicadores clave: el consumo total, el consumo medio diario, así como los días con la máxima y mínima demanda estimada.

Estos resultados se presentan de forma clara y atractiva en cuatro columnas, cada una con un cuadro de color personalizado que incluye un título, los valores en kWh y GWh, el día asociado (en los casos que aplica), y un emoji representativo. Esta visualización se genera mediante la función ``cuadro_resultado()``, que devuelve HTML con estilos CSS personalizados para mantener una presentación uniforme y amigable.

12.2.4 Visualización

Aquí esta el ejemplo de la visualización del frontend:



12.2.5 Conclusión

Este frontend ofrece una interfaz clara, interactiva y visualmente cuidada para consultar predicciones energéticas. Usa Streamlit para la estructura, HTML/CSS embebido para personalización, y se comunica con un backend mediante peticiones HTTP para obtener datos reales o simulados. Es un excelente ejemplo de cómo combinar ciencia de datos con una experiencia de usuario amigable.

12.2.6 Código completo

```
import streamlit as st
import requests
import pandas as pd
from streamlit_extras.stylable_container import stylable_container
```

```
# Estilos personalizados
st.markdown("""
<style>
h1 {
    font-size: 2rem !important;
    margin-top: -2rem !important;
    margin-bottom: 1rem !important;
}

div.stButton > button:first-child,
div.stButton > button:first-child:focus,
div.stButton > button:first-child:active {
    background-color: #007BFF;
    color: white !important;
    font-weight: bold;
    border-radius: 5px;
    height: 3em;
    width: 50%;
    display: block;
    margin: auto;
    font-size: 18px;
    border: none;
}

div.stButton > button:first-child:hover {
    background-color: #0056b3;
    color: white !important;
}

div[class="stDateInput"] input {
    color: white !important;
    background-color: #333333 !important;
    border-radius: 5px !important;
    padding: 0.5em;
}

.st-f0::after { background-color: #007BFF; }
.st-eu::after { border-left-color: #007BFF; }
.st-es::after { border-bottom-color: #007BFF; }
.st-er::after { border-top-color: #007BFF; }
.st-et::after { border-right-color: #007BFF; }
```

```
.st-b3, .st-b4, .st-b5, .st-b6, .st-by, .st-bx, .st-bz, .st-c0 {
    border-color: white;
}

/* Estilo común para los cuadros de resultados */
.result-box {
    padding: 1.2em;
    border-radius: 10px;
    color: black;
    font-size: 1.1rem;
    font-weight: 600;
    margin-top: 1em;
    box-shadow: 2px 2px 8px rgba(0,0,0,0.1);
    text-align: center;

    /* Dimensiones iguales */
    width: 100%;
    min-height: 180px;

    /* Centramos el contenido vertical y horizontalmente */
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
}
</style>
""" , unsafe_allow_html=True)

def cuadro_resultado(titulo, contenido, color_hex, emoji=""):
    return f"""
    <div class="result-box" style="background-color:{color_hex};">
        <strong>{emoji} {titulo}</strong><br>
        {contenido}
    </div>
    """

st.title("Predicción de Consumo por Intervalo de Fechas")
```



```
# Inputs de fechas
coll1, coll2 = st.columns(2)
with coll1:
    start_date = st.date_input("Fecha inicio")
with coll2:
    end_date = st.date_input("Fecha fin")

# Validación de fechas
if start_date > end_date:
    st.error("La fecha de inicio debe ser menor o igual a la fecha de fin.")
else:
    if st.button("Mostrar Predicciones"):
        url =
f"http://127.0.0.1:5000/predict_range?start_date={start_date}&end_date={end_date}"
        response = requests.get(url)

        if response.status_code == 200:
            data = response.json()
            if "error" in data:
                st.error(f"Error en la predicción: {data['error']}")
            else:
                df = pd.DataFrame(data)
                df['fecha'] = pd.to_datetime(df['fecha'])
                df.set_index('fecha', inplace=True)

                st.line_chart(df['prediccion'])

                # Cálculos
                total_consumo_kwh = df['prediccion'].sum()
                total_consumo_gwh = total_consumo_kwh / 1_000_000

                media_consumo_kwh = df['prediccion'].mean()
                media_consumo_gwh = media_consumo_kwh / 1_000_000

                max_dia = df['prediccion'].idxmax().date()
                max_valor_kwh = df['prediccion'].max()
                max_valor_gwh = max_valor_kwh / 1_000_000

                min_dia = df['prediccion'].idxmin().date()
```

```

min_valor_kwh = df['prediccion'].min()
min_valor_gwh = min_valor_kwh / 1_000_000

# Mostrar resultados en columnas
col_tot, col_med, col_max, col_min = st.columns(4)

with col_tot:
    st.markdown(
        cuadro_resultado(
            "Consumo total estimado",
            f"{total_consumo_kwh:,.2f} kWh<br>({total_consumo_gwh:,.2f} GWh)",
            "#90ee90",
            "📊"
        ),
        unsafe_allow_html=True
    )



with col_med:
    st.markdown(
        cuadro_resultado(
            "Consumo medio diario",
            f"{media_consumo_kwh:,.2f} kWh<br>({media_consumo_gwh:,.4f} GWh)",
            "#87cefa",
            "📊"
        ),
        unsafe_allow_html=True
    )

with col_max:
    st.markdown(
        cuadro_resultado(
            "Máxima demanda estimada",
            f"{max_valor_kwh:,.2f} kWh<br>({max_valor_gwh:,.4f} GWh)<br>📅 Día:
<strong>{max_dia}</strong>",
            "#FFA500",
            "📈"
        ),
        unsafe_allow_html=True
    )

with col_min:

```

```

st.markdown(
    cuadro_resultado(
        "Mínima demanda estimada",
        f"{min_valor_kwh:,.2f} kWh<br>({min_valor_gwh:,.4f} GWh)<br> Día:
<strong>{min_dia}</strong>",
        "#FFFF99",
        " "
    ),
    unsafe_allow_html=True
)

else:
    st.error(f"Error al obtener datos del servidor:
{response.status_code}")

```

12.3 Backend

12.3.1 Resumen

El backend se encuentra en `app/api.py` y fue desarrollado con Flask, expone una API que permite predecir el consumo energético diario en un rango de fechas determinado. Utiliza un modelo de regresión entrenado con XGBoost, cargado desde un archivo JSON (`model.json`), y una función auxiliar para la creación de atributos predictivos.

12.3.2 Funcionamiento

Al iniciarse la aplicación, se carga un modelo de XGBoost previamente entrenado desde un archivo en disco. Este modelo se utiliza para realizar predicciones de consumo energético a partir de atributos derivados de fechas, como el día de la semana, mes, año, día del año y trimestre. Estos atributos se generan mediante una función externa llamada `creacion_atributos`, importada desde el módulo `utils`.

12.3.3 Ruta Principal (`/predict_range`)

El backend expone un único endpoint activo, `/predict_range`, que acepta peticiones GET con dos parámetros: `start_date` y `end_date`, ambos en formato YYYY-MM-DD. Al recibir una solicitud, la API genera un rango de fechas comprendido entre las fechas indicadas.

Para cada día dentro de ese rango:

- Se crea un DataFrame con la fecha como índice.
- Se generan atributos predictivos mediante la función `creacion_atributos()`.

- Se realiza una predicción de consumo utilizando el modelo XGBoost cargado previamente.
- Se almacena el resultado como un diccionario que incluye la fecha y el valor estimado.

Una vez procesadas todas las fechas, la API responde con una lista de predicciones en formato JSON, lista para ser utilizada por el frontend. En caso de que ocurra algún error (por ejemplo, si las fechas tienen un formato inválido), se devuelve un mensaje detallando el problema.

12.3.4 Consideraciones implementadas

- El modelo no necesita reentrenarse: se carga directamente desde un archivo .json, lo que mejora el rendimiento y facilita su despliegue.
- Se desactiva el logging por defecto de Flask (werkzeug) para evitar saturar la consola con mensajes en cada solicitud.

12.3.5 Conclusión

Este backend actúa como el motor de predicción de la aplicación, respondiendo a las solicitudes del frontend con estimaciones de consumo energético para cada día en el rango seleccionado. Su diseño modular y eficiente permite escalarlo o integrarlo fácilmente con otras herramientas de visualización o análisis.

12.3.6 Código completo

```
from flask import Flask, request, jsonify
import pandas as pd
#import pickle
from utils import creacion_atributos
import json
import xgboost as xgb
import os
import logging
log = logging.getLogger('werkzeug')
log.setLevel(logging.ERROR)

app = Flask(__name__)

# Carga del modelo XGBoost desde JSON
modelo = xgb.XGBRegressor()

#modelo.load_model('model.json')
#ruta_modelo = os.path.join(os.path.dirname(__file__),
#                             'notebooks/model.json')
ruta_modelo = os.path.join(os.path.dirname(__file__), 'model.json')
```

```
modelo.load_model(ruta_modelo)

atributos = ['dia_semana', 'mes', 'año', 'dia_año', 'trimestre']

# @app.route('/predict', methods=['GET'])
# def predict():
#     fecha = request.args.get('fecha') # formato YYYY-MM-DD
#     try:
#         fecha_dt = pd.to_datetime(fecha)
#         df = pd.DataFrame(index=[fecha_dt])
#         df = creacion_atributos(df)
#         pred = modelo.predict(df[atributos])[0]
#         return jsonify({"fecha": fecha, "prediccion": round(pred,
2)})
#     except Exception as e:
#         return jsonify({"fecha": fecha, "prediccion":
float(round(pred, 2))})

@app.route('/predict_range', methods=['GET'])
def predict_range():
    start_date = request.args.get('start_date') # YYYY-MM-DD
    end_date = request.args.get('end_date') # YYYY-MM-DD
    try:
        fechas = pd.date_range(start=start_date, end=end_date)
        resultados = []
        for fecha_dt in fechas:
            df = pd.DataFrame(index=[fecha_dt])
            df = creacion_atributos(df)
            pred = modelo.predict(df[atributos])[0]
            resultados.append({
                "fecha": fecha_dt.strftime('%Y-%m-%d'),
                "prediccion": float(round(pred, 2))
            })
        return jsonify(resultados)
    except Exception as e:
        return jsonify({"error": str(e)})

if __name__ == '__main__':
    app.run(debug=True)
```

12.4 Instalador

12.4.1 Resumen

Este script batch automatiza la verificación, instalación y configuración de Python, así como la creación de un entorno virtual para el proyecto.

12.4.1.1 Verificación de Python

Primero, el script comprueba si Python ya está instalado en el sistema. Si es así, salta directamente a la instalación de dependencias. Si no lo encuentra:

Descarga automáticamente el instalador oficial de Python 3.11.4 (64-bit) desde el sitio web de Python, si aún no está descargado.

Ejecuta el instalador en modo silencioso, con los parámetros necesarios para instalarlo para todos los usuarios, añadirlo al PATH, y omitir paquetes de prueba.

12.4.1.2 Validación posterior a la instalación

Una vez instalado, vuelve a comprobar si Python está disponible en el sistema. Si no lo está, muestra un mensaje de error y termina el proceso.

12.4.1.3 Gestión de pip

El script verifica si pip está instalado. En caso contrario, intenta instalarlo mediante ensurepip. Luego, actualiza pip a la última versión.

12.4.1.4 Creación del entorno virtual

Utiliza `python -m venv venv` para crear un entorno virtual llamado venv. Si falla, muestra un error y detiene la ejecución.

12.4.1.5 Activación e instalación de dependencias

Activa el entorno virtual y ejecuta la instalación de paquetes definidos en el archivo `requirements.txt`. Si hay errores durante esta instalación, se notifica al usuario.

12.4.1.6 Finalización

Al finalizar correctamente, informa que el entorno está listo para usarse y muestra cómo activarlo manualmente en futuras sesiones (`venv\Scripts\activate.bat`).

Este script es especialmente útil para entornos Windows, facilitando una instalación limpia y automática sin necesidad de intervención manual.

12.4.2 Código completo

```
@echo off
SETLOCAL EnableDelayedExpansion

REM URL del instalador oficial Python 3.11.4 (64-bit)
set "PYTHON_INSTALLER=python-3.11.4-amd64.exe"
set
"PYTHON_URL=https://www.python.org/ftp/python/3.11.4/%PYTHON_INSTALLER%"
"

echo Verificando instalación de Python...

where python >nul 2>nul
IF %ERRORLEVEL% EQU 0 (
    echo Python ya está instalado.
    goto InstallDeps
) ELSE (
    echo Python no está instalado. Se procederá a descargar e instalar.
)

REM Descargar instalador si no existe
if not exist "%PYTHON_INSTALLER%" (
    echo Descargando instalador de Python...
    powershell -Command "Invoke-WebRequest -Uri '%PYTHON_URL%' -OutFile '%PYTHON_INSTALLER%'"
    IF %ERRORLEVEL% NEQ 0 (
        echo [ERROR] No se pudo descargar el instalador de Python.
        pause
        exit /b 1
    )
) ELSE (
    echo Instalador ya descargado.
)

REM Ejecutar instalador en modo silencioso
echo Ejecutando instalador de Python en modo silencioso...
"%CD%\%PYTHON_INSTALLER%" /quiet InstallAllUsers=1 PrependPath=1
Include_test=0
IF %ERRORLEVEL% NEQ 0 (
    echo [ERROR] La instalación de Python falló. Asegúrate de ejecutar
    este script con permisos de administrador.
)
```

```
    pause
    exit /b 1
)

echo Instalación completada.

REM Verificar de nuevo python
where python >nul 2>nul
IF %ERRORLEVEL% NEQ 0 (
    echo [ERROR] Python sigue sin estar disponible en el PATH tras la
instalación.
    echo Intenta reiniciar la consola o añadir Python manualmente al
PATH.
    pause
    exit /b 1
)

:InstallDeps

echo.
echo Verificando instalación de pip...

python -m pip --version >nul 2>nul
IF %ERRORLEVEL% NEQ 0 (
    echo pip no encontrado. Intentando instalar pip...
    python -m ensurepip --default-pip
    IF %ERRORLEVEL% NEQ 0 (
        echo [ERROR] No se pudo instalar pip automáticamente.
        echo Por favor, instala pip manualmente e intenta de nuevo.
        pause
        exit /b 1
    )
) ELSE (
    echo pip está instalado.
)

echo.
echo Actualizando pip a la última versión...
python -m pip install --upgrade pip
IF %ERRORLEVEL% NEQ 0 (
    echo [ERROR] No se pudo actualizar pip.
```



```
    pause
    exit /b 1
)

echo.
echo === Creando entorno virtual ===
python -m venv venv

IF NOT EXIST venv (
    echo [ERROR] No se pudo crear el entorno virtual.
    pause
    exit /b 1
)

echo.
echo === Activando entorno virtual ===
CALL venv\Scripts\activate.bat

echo.
echo Instalando dependencias desde requirements.txt...
python -m pip install -r requirements.txt

IF %ERRORLEVEL% NEQ 0 (
    echo [ERROR] Falló la instalación de paquetes.
    pause
    exit /b 1
)

echo.
echo ✅ Todo listo. El entorno está preparado.
echo Para activarlo manualmente en el futuro, ejecuta:
echo     venv\Scripts\activate.bat
echo.

pause
ENDLOCAL
```

12.5 Ejecutable

12.4.1 Resumen

Este script batch automatiza el arranque simultáneo de la API Flask y la aplicación Streamlit en Windows, asegurando un entorno adecuado para su ejecución.

Primero, cambia la consola a UTF-8 para mejorar el soporte de caracteres especiales. Luego, intenta activar un entorno virtual llamado venv si está disponible; si no, avisa que se usará la instalación global de Python.

A continuación, ejecuta en segundo plano (start /b) el archivo app/api.py que contiene la API Flask, redirigiendo su salida para no mostrarla en pantalla. Después, inicia la aplicación Streamlit (app.py) también en segundo plano, pero esta vez dejando visibles sus logs para facilitar el seguimiento.

Finalmente, muestra mensajes informativos en colores azul y rojo mediante comandos de PowerShell, indicando las URLs locales donde se pueden acceder a la app Streamlit (<http://localhost:8501>) y a la API Flask (<http://127.0.0.1:5000>). También advierte que para detener ambos servicios basta con cerrar la ventana de la consola.

Este proceso automatiza el entorno para facilitar el desarrollo y pruebas sin necesidad de ejecutar comandos manualmente.

12.4.2 Código completo

```
@echo off
SETLOCAL

REM Cambiar consola a UTF-8 para mejor soporte de caracteres
chcp 65001 >nul

REM Activar entorno virtual si existe
if exist venv\Scripts\activate.bat (
    call venv\Scripts\activate.bat
) else (
    echo [WARNING] No se encontró entorno virtual 'venv'. Ejecutando
    con Python global.
)

REM Ejecutar app/api.py sin mostrar salida
start /b python app\api.py >nul 2>&1

REM Ejecutar Streamlit sin redirigir para que muestre sus logs
```

```
start /b streamlit run app.py

REM Mensajes informativos en azul y rojo con PowerShell
echo.

powershell -Command "Write-Host
'===== ' -ForegroundColor
Blue"
powershell -Command "Write-Host 'Ambos scripts se están ejecutando en
esta ventana.' -ForegroundColor Blue"
powershell -Command "Write-Host 'Puedes acceder a la app Streamlit en:'
-ForegroundColor Blue"
powershell -Command "Write-Host ' http://localhost:8501'
-ForegroundColor Blue"
powershell -Command "Write-Host 'Y la API Flask en:' -ForegroundColor
Blue"
powershell -Command "Write-Host ' http://127.0.0.1:5000'
-ForegroundColor Blue"
powershell -Command "Write-Host
'===== ' -ForegroundColor
Blue"

echo.

powershell -Command "Write-Host 'Para detenerlos, simplemente cierra
esta ventana.' -ForegroundColor Red"

REM No pause para no mostrar mensaje de "presione cualquier tecla..."

ENDLOCAL
```

13. Conclusión

El modelo de predicción eléctrica desarrollado ha alcanzado una precisión del 96%, lo que representa una mejora significativa respecto al modelo publicado por Red Eléctrica, el cual está entrenado únicamente con datos hasta el año 2022 y utiliza una arquitectura más antigua. Cabe señalar que esta comparación se realiza sobre la base del modelo público disponible; se desconoce si Red Eléctrica emplea internamente modelos más actualizados.

La mejora obtenida se debe, en gran medida, al uso de datos más recientes y a la implementación de técnicas de modelado más avanzadas, lo que permite capturar con mayor precisión las dinámicas actuales del sistema eléctrico. Esta actualización no solo refuerza la fiabilidad de las predicciones, sino que también demuestra el valor de contar con herramientas modernas y continuamente ajustadas a la evolución del entorno energético. Un modelo actualizado resulta especialmente útil en contextos de alta variabilidad, como los generados por la creciente penetración de fuentes renovables y los cambios en los patrones de consumo.

14. Bibliografía / Webgrafía.

- [1] DataDis. (s.f.). *Inicio*. [En línea]. Disponible en: <https://datadis.es/home>
- [2] Ajuntament de Barcelona. (s.f.). *Consum d'electricitat a Barcelona*. [En línea]. Disponible en: <https://opendata-ajuntament.barcelona.cat/data/es/dataset/consum-electricitat-bcn>
- [3] Documentación XGBoost. (s.f.). [En línea]. Disponible en: <https://xgboost.readthedocs.io/>
- [4] Pandas. (s.f.). *Python Data Analysis Library*. [En línea]. Disponible en: <https://pandas.pydata.org/>
- [5] Scikit-learn. (s.f.). *Machine Learning in Python*. [En línea]. Disponible en: <https://scikit-learn.org/>
- [6] J. Snoek y G. Kalweit, "Data-driven ML models for accurate prediction of energy consumption in a low-energy house: A comparative study of XGBoost, Random Forest, Decision Tree, and Support Vector Machine," *ResearchGate*, 2023. [En línea]. Disponible en: https://www.researchgate.net/publication/377489057_Data-driven_MLmodels_for_accurate_prediction_of_energy_consumption_in_a_low-energy_house_A_comparative_study_of_XGB_oost_Random_Forest_Decision_Tree_and_Support_Vector_Machine
- [7] W. Sun y Q. Wang, "Subway Energy Consumption Prediction based on XGBoost Model," *HSET*, 2020. [En línea]. Disponible en: https://drpress.org/ojs/index.php/HSET/article/view/13958?utm_source=chatgpt.com
- [8] W. Wu, Y. Pan y X. Huang, "Handling missing data of using the XGBoost-based multiple imputation by chained equations regression method," *PMC*, 2020. [En línea]. Disponible en: https://pmc.ncbi.nlm.nih.gov/articles/PMC12003350/?utm_source=chatgpt.com
- [9] J. Brownlee, "Data Preparation for Gradient Boosting with XGBoost in Python," 2020. [En línea]. Disponible en: https://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/?utm_source=chatgpt.com

- [10] T. Chen y C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *ACM Digital Library*, 2016. [En línea]. Disponible en: <https://dl.acm.org/doi/10.1145/2939672.2939785>
- [11] XGBoost GitHub Repository. (s.f.). [En línea]. Disponible en: <https://github.com/dmlc/xgboost>
- [12] J. Snoek y G. Kalweit, “Data-driven ML models for accurate prediction of energy consumption in a low-energy house,” *Energy and Buildings*, vol. 183, pp. 103-112, 2018. [En línea]. Disponible en: <https://www.sciencedirect.com/science/article/abs/pii/S0378778818317298?via%3Dihub>
- [13] W. Sun y Q. Wang, “Subway energy consumption prediction based on XGBoost model,” *Sustainability*, vol. 12, no. 14, p. 5698, 2020. [En línea]. Disponible en: <https://www.mdpi.com/2071-1050/12/14/5698>
- [14] W. Wu, Y. Pan y X. Huang, “Handling missing data using the XGBoost-based multiple imputation by chained equations regression method,” *Computers in Biology and Medicine*, vol. 123, p. 103879, 2020. [En línea]. Disponible en: <https://www.sciencedirect.com/science/article/abs/pii/S0010482520302328?via%3Dihub>
- [15] Codecogs. (s.f.). *Generador de fórmulas LaTeX para el informe*. [En línea]. Disponible en: <https://editor.codecogs.com/>
- [16] Energética.21. (s.f.). El consumo eléctrico de los hogares ha aumentado un 52% desde el inicio de la pandemia. [En línea]. Disponible en: <https://energetica21.com/noticia/el-consumo-electrico-de-los-hogares-ha-aumentado-un-52porciento-desde-el-inicio-de-la-pandemia>

15. Github

