Facultad
de
Ciencias

Análisis y predicción de series temporales para consumo eléctrico y presión en el supercomputador *Finis Terrae II* mediante el uso de Spark

Analysis and prediction of time series for electric power consumption and pressure on the *Finis Terrae II* supercomputer using Spark

Trabajo Fin de Máster

para acceder al

MÁSTER EN CIENCIA DE DATOS

Autor: Luis Moro Carrera

Tutor: Javier Cacheiro López

Septiembre 2023

Agradecimientos

En primer lugar quisiera agradecer a mi director, Javier, el esfuerzo empleado durante este tiempo y su cercanía. Me ha permitido compaginar la realización de este proyecto con mi vida laboral.

Quisiera dar las gracias también a los profesores del máster por ofrecerme la oportunidad de introducirme en un campo que me ha cautivado hasta el punto de ser mi sustento y pasión a día de hoy.

Finalmente, quisiera dar las gracias a mi familia y amigos. En especial, a todos los que he conocido durante mi vida en Barcelona y comparten conmigo el amor por esta disciplina.

Resumen

En el contexto de los Centros de Supercomputación, la mejora de la eficiencia energética resulta de enorme importancia debido a las elevadas necesidades energéticas que se requieren para su funcionamiento, pudiendo alcanzar varios MW. Es por tanto fundamental anticipar el consumo eléctrico y el seguimiento de distintas series temporales relacionadas con el correcto funcionamiento de los sistemas de refrigeración.

Se ha realizado, en colaboración con el Centro de Supercomputacion de Galicia (CESGA), un proyecto para predecir el consumo de sus enfriadoras y la presión a la que se encuentran sometidos los compresores de estas.

Debido al elevado volumen de datos a procesar, es necesario el uso de herramientas de *Big Data* tales como Apache Spark y el ecosistema Apache Hadoop.

En cuanto a los modelos y su tratamiento estadístico, se han probado modelos clásicos basados en autorregresión, suavizado exponencial (Holt-Winters), Prophet y una combinación de suavizado exponencial, media móvil y autorregresión donde se introducen términos adicionales para dar cuenta de varias componentes estacionales (TBATS). Este último ha sido el modelo más adecuado en el caso de la predicción de una serie temporal de consumo energético con múltiples estacionalidades. La variante autorregresiva (SARIMA) ha sido satisfactoria para predecir la presión máxima, ya que se ha observado una clara estacionalidad y una función de autocorrelación con patrón sinusoidal.

Palabras clave: Predicción de series temporales, Apache Spark, *machine learning*, *data science*.

Abstract

In the context of Supercomputing Centers, the improvement of energy efficiency is of enormous importance due to the high energy needs required for their operation, which can reach several MW. It is therefore essential to anticipate the electrical consumption and the monitoring of different time series related to the correct operation of the cooling systems.

A project has been carried out in collaboration with the Supercomputing Center of Galicia (CESGA) to predict the consumption of its chillers and the pressure to which their compressors are subjected.

Due to the high volume of data to be processed, it is necessary to use Big Data tools such as Apache Spark and the Apache Hadoop ecosystem.

Regarding the models and their statistical treatment, classical models based on autoregression, exponential smoothing (Holt-Winters), Prophet and an exponential smoothing state-space model with Box-Cox transformation, ARMA errors, Trend, and Seasonal components (TBATS) were tested.

The latter has been the most suitable in the case of predicting a time series of energy consumption with multiple seasonalities. The autoregressive variant (SARIMA) has been satisfactory for predicting peak pressure, since a clear seasonality and an autocorrelation function with a sinusoidal pattern have been observed.

Keywords: Time series forecasting, Apache Spark, machine learning, data science.

Índice

Índice de figuras

1. Introducción

Durante este capítulo se expondrá, en primer lugar, la motivación para requerir modelos predictivos de las variables características de un sistema de enfriamiento en un Centro de Procesamiento de Datos (CPD).
Seguidamente, se detallará cómo funciona tal sistema de enfriamiento, haciendo énfasis en las enfriadoras, el objeto principal del trabajo.
Por último, se explicará la estructura y alcance del proyecto realizado.

1.1. Motivación y contexto

En el contexto del procesamiento de grandes volúmenes de datos en la nube, una infraestructura segura, con capacidad de escalado y alta velocidad de comunicación desde donde ejecutar los cálculos y procesos es requerida. El Centro de Procesamiento de Datos (CPD) cubre estas necesidades.
Para la implementación y diseño de los CPD, existen una serie de recomendaciones y directrices especificadas en el estándar ANSI-TIA (*American National Standards Institute and Telecommunications Industry Association*) número 492. Entre otras caracterísiticas, se incide en las temperaturas a las que se deben encontrar los servidores y los métodos para refrigerar el aire a su alrededor [1].
Tanto los servidores, como los sistemas de almacenamiento, redes y enfriamiento, requieren un consumo energético del orden de varios cientos de kW. En el caso de este proyecto, el consumo medio del CPD del Centro de Supercomputación de Galicia (CESGA) es de 300 kW [2].
En este contexto, optimizar la eficiencia energética del CPD es prioritario. Para estimarlo, una métrica empleada es el *Power Usage Effectiveness* (PUE), que relaciona el consumo total del centro con el consumo debido a la ejecución de procesos de cómputo en los servidores [3]:

$$\text{PUE} = \frac{P_{total}}{P_{IT}} \tag{1.1}$$

Siendo P_{total} la potencia total consumida por las instalaciones y P_{IT} la potencia total consumida por el equipamiento de los servidores y redes.
Para obtener valores del PUE próximos a la unidad, es necesario reducir el consumo de procesos destinados al mantenimiento de las condiciones óptimas de funcionamiento del CPD.
Uno de estos componentes es la enfriadora, responsable del mantenimiento de la temperatura del aire que rodea a los *rack* de servidores. Anticipar la potencia requerida por este sistema, y otras magnitudes físicas como la presión que ejercen sus compresores, es requerido para garantizar la eficiencia energética del CPD.

1.2. Descripción del problema y entendimiento del sistema

Finis Terrae II es un supercomputador híbrido desplegado por el Centro de Supercomputación de Galicia (CESGA) en 2017. Está compuesto por 298 nodos delgados y 4 nodos de GPU, conectados por el bus de comunicaciones *InfiniBand FBR* y emplea una arquitectura de topología tipo *Fat Tree*, garantizando baja latencia y suficiente ancho de banda [4].
Para refrigerar el aire que rodea este sistema, se emplean dos enfriadoras de agua (*water-cooled chiller*), cada una de las cuales tiene dos compresores.
A continuación se describen los principales componentes de una enfriadora de agua y las principales magnitudes físicas medidas por los sensores incorporados en sus componentes principales [5]:

- El agua a baja temperatura T_{out} circula por la climatizadora, intercambiando energía térmica con el aire que rodea a los servidores. El aire del CPD se enfría y la temperatura del agua T_{in} aumenta hasta los 18-25 °C.

- El agua a temperatura T_{in} circula hasta el sistema de dos enfriadoras, cada una de las cuales tiene una bomba que acelera el flujo. Estas bombas garantizan un consumo base de 10.0 ± 0.1 kW. Si una de las bombas no está funcionando, se corta el flujo a través de la enfriadora correspondiente.

- El agua a temperatura T_{in} circula hasta el evaporador de la enfriadora correspondiente, donde se mide su temporatura $T_{evaporator}$. Durante este trayecto, si la temperatura ambiente $T_{ambient}$ es suficientemente baja, el agua sufre un proceso de enfriamiento de forma natural, haciendo innecesarios los siguientes procesos de las enfriadoras y ahorrando consumo de potencia eléctrica. Este fenómeno se denomina *free cooling* y contribuye al aumento de la eficiencia energética global del CPD.

  Debido a que el CESGA se encuentra situado en la región de Galicia, el *free cooling* ocurre con mayor frecuencia durante los meses de invierno.

- El flujo de agua a T_{in} alcanza el evaporador. Si la temperatura ambiente $T_{ambient}$ ha permitido que tenga lugar el *free cooling*, y la temperatura medida al llegar al evaporador $T_{evaporator}$ es inferior o igual al $T_{threshold}$ de 15 °C, el ciclo de enfriamiento termina. El agua atravesará los componentes restantes de la enfriadora desconectados, hasta alcanzar de nuevo la climatizadora. De este modo, el agua enfría nuevamente el aire del CPD y el proceso vuelve a comenzar. La temperatura T_{out} a la salida de las enfriadoras es medida por un sensor.

- Si $T_{evaporator}$ >15 °C, al menos uno de los compresores de una de las dos enfriadoras deberá estar encendido. La cantidad de compresores conectados variará en función de la presión y temperatura del agua.

  Los compresores aumentan la presión y temperatura del agua, pasando esta a estado gaseoso para que pueda circular hasta un sistema de condensador, que lo devuelve a estado líquido de alta presión, y válvula de expansión. En la válvula de expansión el agua líquida a alta presión sufre una regulación de su flujo, que disminuye su presión nuevamente. Al disminuir la presión, el agua disminuye también su temperatura.

  Finalmente, el agua sale de las enfriadora, donde se mide su temperatura T_{out}, y vuelve a refrigerar el aire del CPD a través de la climatizadora.

El objetivo principal de la memoria será por tanto estimar el consumo suma de las dos enfriadoras con granularidad horaria, disponiendo del histórico de consumos en kW. En particular, el Centro de Supercomputación de Galicia propone verificar la validez de los modelos para el día 31 de mayo de 2021 (24 puntos).
De modo similar, se propone también estimar, para el mismos rango de tiempo, la máxima presión alcanzada por cualquiera de los 4 compresores p_{max} durante el proceso de enfriamiento. También se dispone de un histórico de medidas en bar. Como es en verano y principios de otoño cuando se realiza un uso más intensivo de las enfriadoras, se propone la fecha del 22 de septiembre de 2020 para realizar predicciones horarias (24 puntos).

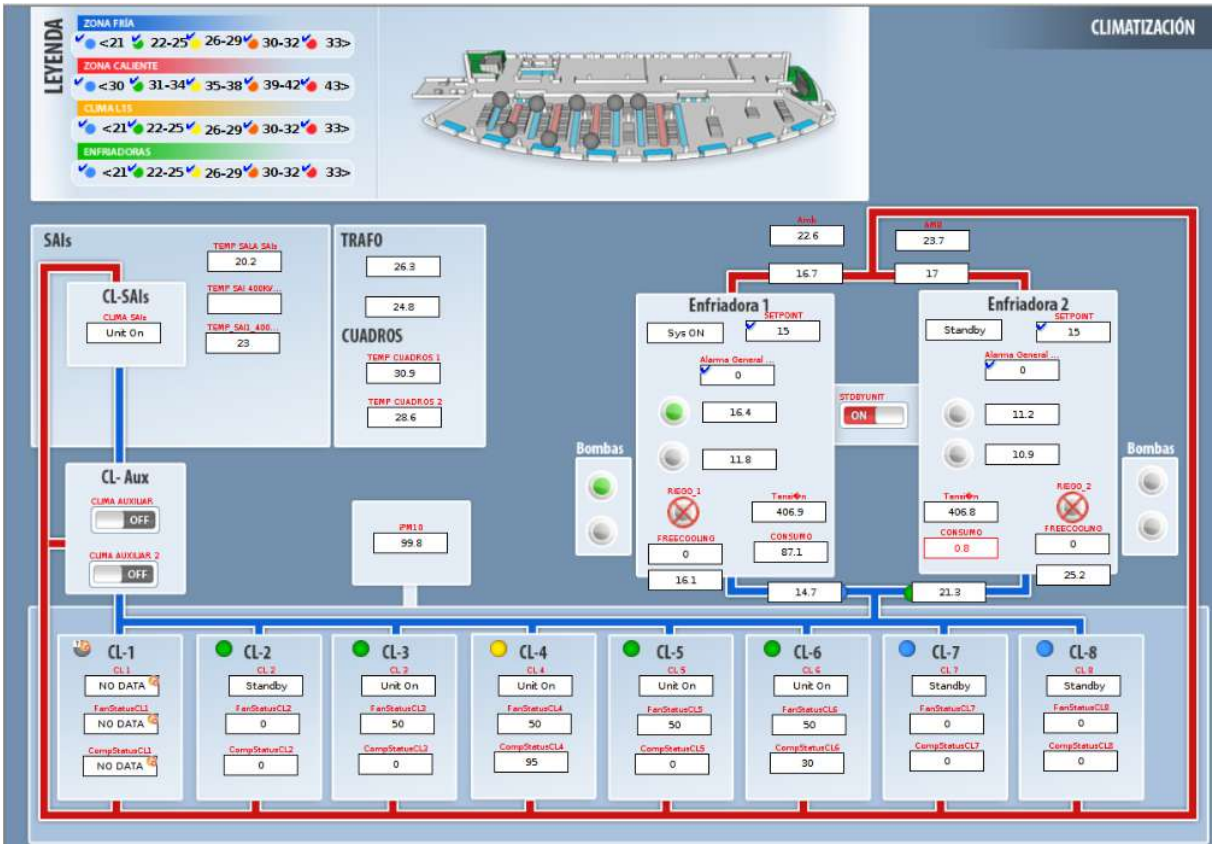

Figura 1: Esquema del sistema de enfriamiento del CESGA. Dos enfriadoras, con dos compresores cada una, reciben el agua a alta temperatura proveniente de refrigerar el clúster y reducen su temperatura.

1.3. Organización y alcance del trabajo

El principal objetivo de este trabajo es la implementación de modelos predictivos que permitan estimar las próximas medidas de consumo eléctrico de las dos enfriadoras, así como la presión máxima P_{max} registrada por cualquiera de sus 4 compresores, que el Centro de Supercomputación de Galicia utiliza para mantener la temperatura de su clúster principal estable.

El histórico de datos, tanto de variables exógenas como endógenas, consta de medidas cada varios segundos para los últimos 3 años. La frecuencia exacta es distinta en cada caso, dependiendo de la variable y el sensor responsable de registrarla. Debido a esto, el *dataset* original consta de 288 series temporales para el consumo de los nodos principales del clúster, 4 series temporales para la temperatura en cada paso por la enfriadora, 4 series temporales con la presión registrada en cada compresor y 2 series temporales con el consumo de cada enfriadora. En todos los casos, cada serie tiene más de un millón de registros, que deben ser preprocesados antes de entrenar los modelos. Por todo ello, se requiere del uso del *framework* de computación distribuida Apache Spark [6], instalado en el propio clúster del CESGA.

La naturaleza de dichas mediciones, con una granularidad del orden del minuto tras su preprocesado, implica una serie de dificultades particulares a la hora de obtener modelos predictivos satisfactorios. Esto se debe a que dichas series temporales presentarán múltiples estacionalidades, principalmente diarias y anuales [7]. En este escenario, los modelos clásicos basados en una aproximación econométrica, en específico aquellos que consideran solamente una estacionalidad, como es el caso del frecuentemente utilizado SARIMAX [8] no son adecuados.

Se definen por tanto tres tareas fundamentales:

- Realización de un proceso de *Extract, Transform and Load* (ETL) para limpiar, agrupar y preparar el gran volumen original de datos.
- Análisis Exploratorio de Datos (EDA por sus siglas en inglés), con especial énfasis en indicadores estadísticos y visualizaciones usuales en el ámbito del estudio de series temporales.
- Implementación y validación de modelos predictivos compatibles con múltiples estacionalidades para la estimación del consumo conjunto de ambas enfriadoras y la máxima presión registrada por cualquiera de los 4 compresores.

2. Fundamento teórico y estado del arte

En los siguientes apartados se introduce el *framework* de computación distribuida Apache Spark y la teoría detrás del análisis y el modelado predictivo de series temporales.

Como se ha explicado en el capítulo anterior, debido a la existencia de múltiples estacionalidades en los datos, los modelos clásicos basados en suavizado exponencial y los basados en una aproximación econométrica pueden no ser óptimos para esta tarea. Sin embargo, su introducción es necesaria debido a que algunos de los modelos que comprenden múltiples estacionalidades, como es el caso de la Aproximación Trigonométrica Box-Jenkins Estacional Con Estacionalidades que Varían en el Tiempo (TBATS por sus siglas en inglés)[9] son correcciones o combinaciones de estos.

2.1. Apache Hadoop

Apache Hadoop es un *framework* de computación distribuida. Permite escalar desde un único servidor hasta miles de máquinas (clúster), cada una de ellas abstrayéndo sus recursos en nodos, con su capacidad local de cómputo y almacenamiento [10]. Su ecosistema incluye módulos que permiten realizar las distintas tareas asociadas a manejar grandes volúmenes de datos.
En particular, para la realización de este trabajo es necesario entender el Sistema de Ficheros Distribuido de Hadoop (HDFS por sus siglas en inglés), Apache Parquet y Hadoop YARN. También se introducirá Apache Spark, que no pertenece estrictamente al proyecto Apache Hadoop, pero utiliza librerías clientes de este.

2.1.1. Apache Hadoop YARN

Apache Hadoop YARN, cuyas siglas en inglés significan "*Yet Another Resource Negotiator*" es el módulo del ecosistema Apache Hadoop encargado de gestionar los recursos del clúster en tiempo real. En particular, gestiona los recursos de CPU, memoria y almacenamiento. También se encarga de la monitorización y planificación de la ejecución de tareas. Estas dos funcionalidades principales son separadas en dos *daemons* independientes [11].

Las abstracciones principales detrás de Apache Hadoop YARN son el Gestor de Recursos global (RM por sus siglas en inglés) y el Maestro de Aplicación (*Application Master* o AM en inglés). Cada aplicación tiene su propio AM y puede estar compuesta de una única tarea (*job*) o un Grafo Acíclico Dirigido (DAG) de tareas.
Desde una perspectiva de *framework* para el cómputo de datos, los componentes principales serían el Gestor de Recursos global (RM) y el Gestor de Nodos (*Node Manager*). Cada máquina del clúster tiene su propio Gestor de Nodos.

El Gestor de Recursos global es la autoridad última que decide cómo se reparten los recursos disponibles en el clúster entre todas las aplicaciones que se están intentando ejecutar como trabajo (*job* o DAG). El Gestor de Nodos de cada máquina es un *framework agent* responsable de la administración de los contenedores, monitorización de sus recursos (CPU, memoria, disco, red) e informar sobre ello al Gestor de Recursos.
El Maestro de Aplicación es una librería cuya función es negociar los recursos con el Gestor de Recursos global y trabajar con el Gestor de Nodos para ejecutar y monitorizorear las tareas.

A su vez, el Gestor de Recursos global tiene dos componentes principales: el planificador (*Scheduler*) y el Gestor de Aplicaciones (*Applications Manager*).
El planificador es el responsable de asignar los recursos a cada una de las aplicaciones que dependen de restricciones tales como la capacidad de cómputo de los nodos, o de que un *job* previo haya finalizado (en caso de que la aplicación consista en un DAG de tareas). El planificador no monitorizorea el estado actual de cada aplicación ni garantiza que, en caso de fallo, la tarea se

vuelva a lanzar. Basa su organización únicamente en los recursos requeridos por cada aplicación y los recursos disponibles en cada contenedor.
El Gestor de Aplicaciones (*Applications Manager*) es el responsable de aceptar los *jobs*, negociar el primer contenedor en el que empezará a ejecutar el correspondiente Maestro de Aplicación (*Application Master*) y relanzar este último en caso de fallo en el contenedor.

El esquema de la arquitectura de Apache Hadoop YARN se muestra en la figura 2.

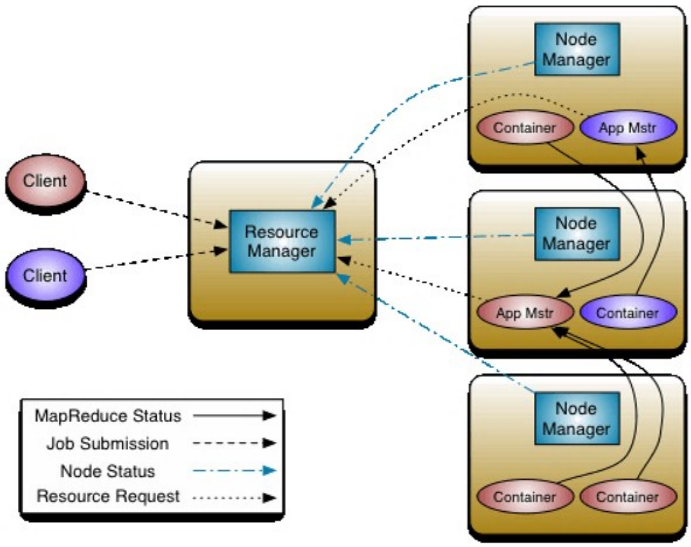

Figura 2: Arquitectura de Apache Hadoop YARN. Aportado por la referencia [11] de bibliografía.

2.1.2. Arquitectura HDFS y Apache Parquet

El *dataset* original empleado para el proyecto, que será procesado mediante Apache Spark, se encuentra almacenado en HDFS. HDFS es el Sistema de Ficheros Hadoop Distribuido. Permite distribuir la información a almacenar entre las distintas máquinas que conforman los nodos del clúster. Se trata de un sistema tolerante a fallos, pues la información se encuentra replicada en varios nodos [12].
HDFS tiene un mejor rendimiento en grandes ficheros (más de 100 MB). Los archivos son del tipo "*Write Once Read Many*" (WORM), de modo que están optimizados para escribirse una única vez y consultarse múltiples veces, teniendo que generar otro archivo desde el principio y no pudiendo modificar parcialmente el original.
Los archivos se separan en bloques, donde el tamaño por defecto del bloque es de 128 MB, y por defecto se replican en 3 nodos.

HDFS tiene una arquitectura de maestro y esclavo, con una estructura jerárquica tradicional de los archivos. Un clúster HDFS consiste en un único *NameNode*, que es un nodo maestro que se encarga de administrar el *namespace* del sistema y regula el acceso a los archivos por parte de los clientes conectados al clúster. Ejecuta operaciones tales como la apertura, cierre y renombrado de archivos y directorios.
Habitualmente, cada nodo del clúster posee un *DataNode*, encargado de gestionar el almacenamiento en memoria de dicho nodo. Se encargan de servir las operaciones de lectura y escritura solicitadas por los clientes conectados al clúster. También gestionan la ejecución de creación, borrado y replicación de bloques tras recibir la solicitud desde el *NameNode*.
Aunque HDFS exponga al usuario un *namespace* de un sistema de ficheros tradicional, internamente los archivos están separados en uno o más bloques, y cada bloque se encuentra replicado en varios nodos.

El esquema de la arquitectura de HDFS se muestra en la figura 3.

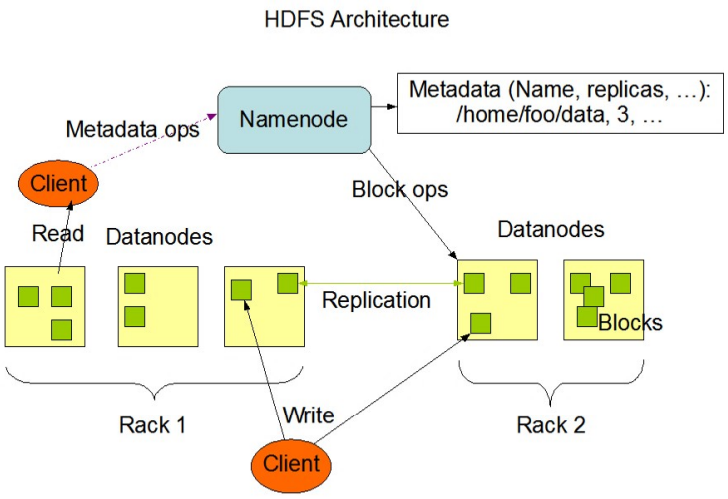

Figura 3: Arquitectura de Apache HDFS. Aportado por la referencia [12] de bibliografía.

Respecto al formato del propio fichero que se va a almacenar, se emplea Apache Parquet [13]. Es un formato de archivo basado en el indexado de columnas, a diferencia de los formatos de archivo tradicionales que almacenan sus datos en formato fila. Esto permite a Apache Parquet dividir los datos en conjuntos de filas y columnas relacionadas entre sí en *row groups*, lo que permite leer solamente las columnas necesarias para la consulta y no todas las incluídas en el *dataset*. Además, permite filtrar bloques innecesarios en la ejecución de la consulta basándose en los metadatos del fichero y estadísticas de lecturas previas mediante una serie de operaciones conocidas como "*predicate pushdown*".
De este modo, Apache Parquet es especialmente beneficioso cuando se trabaja con un sistema de ficheros distribuído como Apache HDFS, en especial cuando se va a trabajar con una gran cantidad de registros en formato tabular.

2.1.3. Apache Spark

Apache Spark es un *framework* de computación distribuida que permite paralelizar el procesamiento de grandes volúmenes de datos [6]. Su código fuente se encuentra escrito en el lenguaje de programación Scala, ejecutándose en la *Java Virtual Machine* (JVM). Spark dispone así mismo de APIs para otros lenguajes de programación. Este es el caso de este proyecto, realizado íntegramente en la API de Spark para Python; PySpark. Con independencia de la API o lenguaje de programación empleado, se utiliza este mismo entorno de ejecución.

Spark, pese a su integración con diversas herramientas del ecosistema Hadoop, no utiliza el paradigma *MapReduce*.
Una aplicación de Spark se compone de un programa *driver* que ejecuta el código principal, distribuyendo las operaciones entre los ejecutores asignados a la aplicación por Apache Hadoop YARN. Esto se ilustra en la figura 4 aportada por la documentación de la referencia [6] de bibliografía.

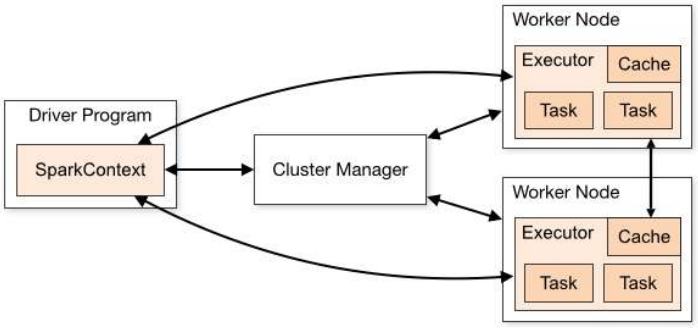

Figura 4: Arquitectura de Apache Spark. Aportado por la referencia [6] de bibliografía.

Para operar con las particiones entre los distintos nodos se introduce el concepto de *Resilient Distributed Dataset* (RDD). Es una abstracción que representa una colección de elementos distribuídos entre los nodos del clúster y proporciona una serie de métodos que permite realizar operaciones en paralelo de forma transparente, de modo que durante el desarrollo, el RDD es tratado como un único objeto. Como objeto principal para comunicarse con la API RDD, Spark dispone del *SparkContext*.
La colección de elementos de un RDD se separa y procesa entre los distintos nodos ejecutores conformando las particiones.
Cuando se emplean grandes archivos en formato HDFS (con una elevada cantidad de bloques HDFS), las particiones del RDD pueden considerarse equivalentes a los bloques HDFS del archivo dado.

Las distintas operaciones que debe realizar cada nodo ejecutor se ordenan en un Grafo Acíclico Dirigido (DAG por sus siglas en inglés). En general, cada tarea de la aplicación se ejecuta sobre cada partición del RDD en su correspondiente nodo ejecutor. Las operaciones en Spark se dividen, de forma similar a *MapReduce*, en dos grupos: Transformaciones y Acciones:

- Las Transformaciones no se ejecutan en el momento de ser evaluadas, sino cuando son requeridas para una Acción posterior del grafo. Este concepto se suele denominar en bibliografía con el término inglés "*lazy evaluation*". Ejemplos de ello son la propia operación *map* y el filtro (*filter*).
- Las acciones requieren ser evaluadas y fuerzan la ejecución del grafo hasta el punto donde son declaradas. Además, el resultado de las mismas no se almacena temporalmente en los nodos ejecutores, sino que se trata de algún tipo de cálculo que requiere agrupar de nuevo la partición del RDD que se encuentra en cada nodo y enviar el resultado al *driver*. Ejemplos de ello son la propia operación *reduce* y *collect*.

Además de la programación en Spark mediante RDD el uso de la API para RDD, existen APIs que proporcionan un mayor nivel de abstracción, permitiendo trabajar con datos estructurados y que disponen de un esquema (metadatos que describen sus relaciones y organziación). Este es el caso de Spark SQL, que además de permitir una sintaxis similar a ANSI SQL o HiveQL, provee una estructura de datos de tipo *DataFrame* definida sobre los RDD. El procesamiento de estos *DataFrame* presenta optimizaciones a bajo nivel mediante el *Catalyst Query Optimizer*. Los *DataFrame* se particionan en subconjuntos de filas. De modo similar al *SparkContext*, la API de Spark SQL dispone de un objeto de entrada llamado *SparkSession*.

Es posible definir operaciones de Python para aplicar a cada partición mediante las Funciones Definidas por el Usuario (UDF por sus siglas en inglés). En particular, existe una categoría especial denominada Pandas UDF, que permite definir funciones para aplicar a cada fila del *DataFrame* original del mismo modo que se podría usar una función *lambda* dentro de una función

apply en un *DataFrame* de la librería Pandas de Python. Esto permite escalar funcionalidades definidas sobre Pandas con Spark.

Apache Spark dispone de un conjunto de librerías para realizar distintos tipos de procesado y análisis. Ejemplos de ello son la ya mencionada Spark SQL, MLlib para entrenar y utilizar modelos de *Machine Learning* cuyo formalismo permita paralelizar sus operaciones o GraphX para el procesamiento de grafos.

En el Apéndice A se muestra un ejemplo del procesado de un fichero de Apache Parquet que contiene series temporales utilizando la API de Spark SQL.

2.2. Series temporales

Una serie temporal es un conjunto de N medidas ordenadas en el tiempo t e igualmente espaciadas [14].

En el caso de ser medidas de una única característica, las series temporales reciben el nombre de series temporales univariantes, pudiendo representarse vectorialmente como:

$$\vec{x}(t) \equiv [x_0, x_1, x_2, \ldots, x_N] \tag{2.1}$$

Cuando se realizan medidas de M características, con $M > 1$, se denomina serie temporal multivariante, cuya representación matricial viene dada por:

$$\mathbf{X} \equiv \begin{bmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & & \vdots \\ x_{N1} & \cdots & x_{NM} \end{bmatrix} \tag{2.2}$$

De este modo, x_{ij} será una observación de la característica j-ésima para el instante i-ésimo dentro de la secuencia temporal igualmente distribuída en el tiempo t.

2.3. Análisis de series temporales

Como paso previo a la implementación de modelos predictivos, es necesario realizar un estudio de las distintas características de la serie temporal. Esto permite comprender qué modelos son más adecuados para el problema a resolver, inferir algunos hiperparámetros de dichos modelos, y diseñar *features* que puedan ser más adecuadas para el entrenamiento [15].

2.3.1. Estacionariedad y test de Dickey-Fuller Aumentado

Los modelos para series temporales tratan de predecir propiedades tales como la media o la varianza de la serie. Es conveniente que las futuras propiedades estadísticas de la serie no difieran de aquellas que podemos observar en las medidas realizadas [15]. Esta idea motiva la definición de estacionariedad, que muchos modelos asumen (por ejemplo, los modelos de tipo autorregresivo).

Una serie temporal se considera estacionaria si sus propiedades estadísticas no cambian en el tiempo. Es decir, si posee una media, varianza y autocorrelación independientes del tiempo t. Esto se ejemplifica en la figura 5.

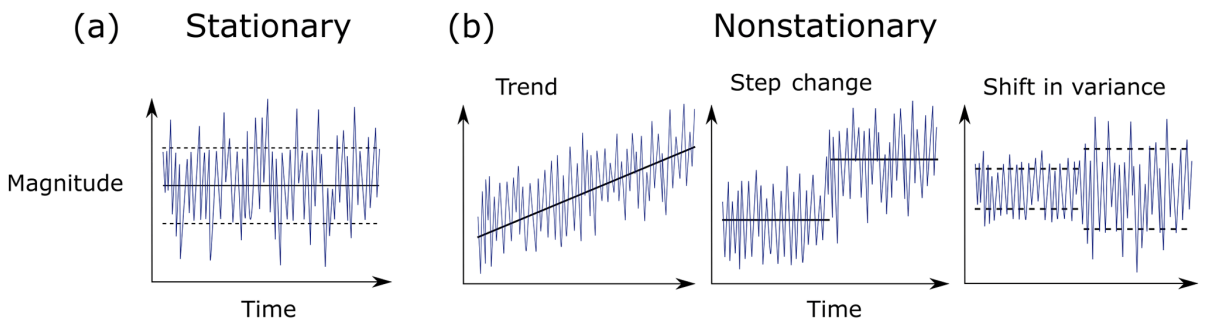

Figura 5: Representación gráfica de las diferencias entre una serie no estacionaria (a la izquierda) y una estacionaria (a la derecha). Aportado por la referencia [16] de bibliografía.

Cuando la varianza es constante en el tiempo se denomina homocedasticidad. Cuando no es constante, se denomina heterocedasticidad.

Para determinar si una serie temporal es estacionaria, se introduce el test de Dickey-Fuller Aumentado (ADF por sus siglas en inglés). El test ADF tiene como hipótesis nula H_0 la existencia de una raíz unitaria en la serie temporal. La hipótesis alternativa afirma que no exite ninguna raíz unitaria en la serie, siendo por tanto estacionaria [17]. Si el valor p (*p-value*) del test resulta ser inferior a 0.05, se puede rechazar la hipótesis nula H_0 y asumir la serie temporal como estacionaria.

Un ejemplo simple de aplicación del test de Dickey-Fuller Aumentado sería considerar la serie temporal dada por la siguiente expresión [14]:

$$x_t = C + \alpha_1 x_{t-1} + \epsilon_t \tag{2.3}$$

Donde C es una constante y α_1 la raíz de la serie temporal. Ambas deben ser calculadas. Además, ϵ_t es un término de error tal que ϵ_t y $\epsilon_{t'}$ para cualquier t' distinto de t no presentan correlación significativa (se considera aproximadamente ruido blanco).
La serie temporal solamente será estacionaria si $|\alpha_1|$ <1.

Como se verá posteriormente, al aplicar el test ADF a las series temporales de presión y consumo de las enfriadoras se ha obtenido en ambos casos un valor p sensiblemente inferior a 0.05. No ha sido necesario realizar transformaciones sobre las series temporales para hacerlas estacionarias. Sin embargo, es conveniente tener en cuenta que las dos transformaciones más habituales para hacer estacionaria una serie temporal son tomar logaritmos de las medidas o diferenciar. La diferenciación x'_t de primer orden de cada medida x_t viene dada por:

$$x'_t = x_t - x_{t-1} \tag{2.4}$$

De este modo, en la serie temporal obtenida tras realizar la diferenciación de primer orden se pierde la primera medida original correspondiente a x_0.

2.3.2. Descomposición ETS

Una serie temporal $\vec{x}(t)$ puede ser expresada como la combinación de las siguientes componentes o patrones:

- La tendencia $\vec{T}(t)$ es un término que se da cuando la media de las medidas incrementa o disminuye su valor significativamente a largo plazo. No es necesario que sea lineal. Puede ser creciente o decreciente y alternar entre ambos comportamientos en el tiempo.

- La estacionalidad $\vec{S}(t)$ es un término que describe patrones que se repiten en el tiempo con una frecuencia fija y conocida. Un día o un año son frecuencias usuales de estacionalidad. Un ejemplo de estacionalidad puede ser la compra de un producto que principalmente se utiliza en una cierta época del año.
- La componente cíclica $\vec{C}(t)$ está asociada a fluctuaciones que no se dan con una frecuencia fija pero son observables en un período de tiempo usualmente superior a los dos años [8].
- La componente de ruido o error $\vec{E}(t)$ es la componente que no se puede ajustar ni atribuir a ninguna de las otras tres componentes y está causada por procesos estocásticos.

De entre estas componentes, una forma usual de descomponer la serie temporal para el estudio de sus características es la descomposición en error, tendencia y estacionalidad (ETS por sus siglas en inglés).
Si la magnitud de las oscilaciones asociadas a la componente estacional $\vec{S}(t)$, o a la combinación de ciclo $\vec{C}(t)$ y tendencia $\vec{T}(t)$ no dependen significativamente del tiempo y se mantienen estables, la descomposición ETS aditiva obtiene mejores resultados [8]:

$$\vec{x}(t) = \vec{S}(t) + \vec{T}(t) + \vec{E}(t) \tag{2.5}$$

Cuando la magnitud de las oscilaciones de la componente estacional o las de la combinación ciclo-tendencia varían significativamente en el tiempo, entonces es necesario usar la descomposición ETS multiplicativa, que no asume independencia entre los términos de tendencia, estacionalidad y error:

$$\vec{x}(t) = \vec{S}(t) \cdot \vec{T}(t) \cdot \vec{E}(t) \tag{2.6}$$

Una vez identificado el tipo de descomposición ETS, se calcula la tendencia $\vec{T}(t)$ mediante un filtro de media móvil [18].
Después, se obtiene la estacionalidad $\vec{S}(t)$ restando (ecuación 2.5) o dividiendo (ecuación 2.6) $\vec{x}(t)$ y la $\vec{T}(t)$ obtenida según corresponda y agrupando los datos (tomando la media) para la frecuencia estacional estimada. Esta frecuencia suele aproximarse mediante la autocorrelación, que se introducirá en la próxima sección.
Finalmente, se calcula la componente de ruido $\vec{E}(t)$ despejando de 2.5 o 2.6 según corresponda.

2.3.3. Autocorrelación

La función de autocorrelación r_k (ACF por sus siglas en inglés) de una serie temporal estacionaria $\vec{x}(t)$ para un retardo (*lag*) k se define como la correlación de la serie con una versión de sí misma con un retardo de k medidas, $\vec{x}_{t-1}$ [8]. Puede escribirse como:

$$r_k = \frac{\sum_{i=k+1}^{N} (x_i - \bar{x}) \cdot (x_{i-k} - \bar{x})}{\sum_{i=1}^{N} (x_i - \bar{x})^2} \tag{2.7}$$

Donde las medidas de la serie temporal se encuentran igualmente espaciadas en el tiempo t y N es su última y más reciente medida.
De este modo, la autocorrelación r_k mide la relación lineal que existe etre una medida x_i de una serie temporal y la obtenida k mediciones antes x_{i-k}.

Es importante notar que la medida x_{i-2} afecta a la medida x_i de modo directo. Sin embargo, también tendrá un impacto en x_{i-1}, y esta a su vez en x_i, de modo que x_{i-2} tendrá también un impacto indirecto sobre x_i.
Del mismo modo, se puede decir que x_{i-1} impacta de modo directo en x_i pero también actúa como variable de confusión (*confounding variable*) en la relación entre x_i e x_{i-2}.
Para mitigiar este efecto y solamente apreciar los impactos directos entre mediciones se introduce el concepto de autocorrelación parcial (PACF).

Para obtener la autocorrelación parcial k-ésima p_k, se construye un modelo de regresión basado en la suma de las k mediciones anteriores. Los pesos de dicho ajuste serán las autocorrelaciones parciales:

$$x_i = p_1 x_{i-1} + p_2 x_{i-2} + \ldots + p_k x_{i-k} + \epsilon_i \tag{2.8}$$

2.3.4. Métricas de validación

Para evaluar y comparar la validez de los distintos modelos empleados es necesario definir una métrica que permita cuantificar el error en las predicciones para la partición de validación del conjunto de medidas original.
Tanto en problemas de regresión, como en predicción de series temporales en particular, una de las métricas más extendidas consiste en tomar la raíz cuadrada del error cuadrático medio (RMSE por sus siglas en inglés). Se trata de una métrica de error no relativa, y por tanto dependiente de la escala de las medidas [8]:

$$RMSE = \sqrt{\frac{\sum_{j=1}^{\mu} \left(x_j^p - x_j^t\right)^2}{M}} \tag{2.9}$$

Donde x_j^p es el valor predicho j-ésimo de entre las próximas M medidas que se tratan de estimar y x_j^t es el valor real medido (*ground truth*) con el que se corresponde en el conjunto de medidas de validación.

En adición al RMSE, se empleará otra métrica de error; el error porcentual absoluto medio simétrico (sMAPE por sus siglas en inglés) [19]. Se trata de un error relativo, lo cual resulta más intuitivo al tratarse de un error independiente de la escala de las medidas. Además, a diferencia de otros errores independientes de la escala, el sMAPE penaliza por igual que la predicción x_j^p exceda o sea inferior al valor observado x_j^t. Se trata de una correción al error de porcentaje medio absoluto (MAPE), cuya formulación acota inferiormente el peso del error de una predicción inferior al valor real mientras que el error de una predicción mayor que el valor real no tiene cota superior. Esto favorece a modelos que realicen predicciones inferiores al valor real o conservadores, lo cual no es conveniente cuando uno de nuestro problemas es tratar de predecir máximos de presión en los compresores de las enfriadoras.
La fórmula del sMAPE viene dada por:

$$sMAPE = \frac{1}{M} \sum_{j=1}^{M} \frac{\left|x_j^p - x_j^t\right|}{\left(x_j^p + x_j^t\right)/2} \tag{2.10}$$

2.4. Modelos clásicos para la predicción de series temporales

Es posible separar a los modelos clásicos en dos grandes familias: modelos basados en suavizado exponencial y modelos autorregresivos o econométricos (llamados así a causa de su orígen histórico) [15].

2.4.1. Modelos de suavizado exponencial

Comenzando con una aproximación *naivë*, podemos asumir que el valor futuro x_t de la medida de una serie temporal depende de la media de sus k valores previos. De este modo, tenemos una media móvil:

$$x_t^p = \frac{1}{k} \sum_{i=1}^{k} x_{t-i} \tag{2.11}$$

Donde x_t^p es el valor futuro t-ésimo estimado con los k valores previos.

Como siguiente paso, en lugar de utilizar de forma equitativa los k valores previos, asignamos a cada una de las observaciones existentes un peso, que decrece exponencialmente cuando nos alejamos hacia atrás en el tiempo. Se define de este modo el suavizado exponencial:

$$x_t^p = \alpha x_t^t + (1-\alpha) \cdot x_{t-1}^p \tag{2.12}$$

Donde el valor aproximado t-ésimo x_t^p se considera una media con pesos de el valor real medido t-ésimo x_t^t y el valor previo aproximado x_{t-1}^p. El peso α se denomina constante de suavización. Define cómo de rápido "se olvidan" las últimas observaciones reales disponibles [15].
El decaimiento exponencial se debe a la recursividad de la función 2.12, ya que se multiplica por $(1-\alpha)$ repetidamente a x_{t-i}^p, quien contiene los $(1-\alpha)$ de los pasos anteriores.

Hasta ahora, en esta sección se ha discutido cómo estimar una única medida futura. Es necesario extenderlo para dos o más puntos futuros.
Para ello, se descompone la serie temporal en dos componentes, cada una con su constante de suavización: el nivel l_t y la pendiente (tendencia) b_t. Esto se conoce como suavizado exponencial doble o método de Holt [15]:

$$\begin{cases} l_t = \alpha x_t + (1-\alpha)\,(l_{t-1} + b_{t-1}) \\ b_t = \beta\,(l_t - l_{t-1}) + (1-\beta) b_{t-1} \\ x_{t+1}^p = l_t + b_t \end{cases} \tag{2.13}$$

Donde la primera ecuación describe el nivel, que como en el caso del suavizado simple depende del valor actual (inmediatamente anterior al que se quiere predecir) de la serie. La segunda euación describe la tendencia, que depende del valor previo de la tendencia y del cambio de nivel en el paso actual. Podemos pensar en el nivel intuitivamente como la media local; "cómo de alto" nos encontramos en el eje de ordenadas.

Por último, en el caso de que nuestro modelo presente una estacionalidad, podemos introducir una tercera componente s_t al modelo que da cuenta de esa estacionalidad y, por tanto, que explique las variaciones de forma repetitiva de nivel y tendencia. Para conocer la duración de un período estacional, es decir, el número m de observaciones que se toman hasta que la variación a causa de la estacionalidad se repite, tendremos m componentes estacionales separadas. Por ejemplo, para una estacionalidad de 7 días, si se tienen medidas cada hora, se tendrán 168 componentes estacionales.
Este método se conoce como triple suavizado exponencial o método de Holt-Winters [20]. El efecto estacional puede ser aditivo o multiplicativo, de modo similar a como se ha explicado en la descomposición ETS. En el caso particular de este proyecto se debe considerar el método aditivo:

$$\begin{cases} l_t = \alpha\,(x_t - s_{t-m}) + (1-\alpha)\,(l_{t-1} + b_{t-1}) \\ b_t = \beta\,(l_t - l_{t-1}) + (1-\beta) b_{t-1} \\ s_t = \gamma\,(x_t - l_{t-1} - b_{t-1}) + (1-\gamma) s_{t-m} \\ x_{t+h}^p = l_t + h b_t + s_{t+h-m} \end{cases} \tag{2.14}$$

Donde m es la estacionalidad de la serie, γ es la constante de suavización de la componente estacional y h es el número de medidas por delante de la última t disponible, para la que se quiere obtener una estimación x_{t+h}^p.
Se tiene además que $0 \leq \gamma \leq 1-\alpha$.
El nivel l_t depende ahora del valor actual de la serie x_t menos la componente estacional de un período antes s_{t-m}. La tendencia se define del mismo modo que en el doble suavizado exponencial y la componente estacional s_t depende del valor actual de la serie, el nivel l_{t-1} y la tendencia b_{t-1} en el paso anterior y el valor de la propia componente estacional un período antes s_{t-m}.

2.4.2. Modelos autorregresivos

Un proceso autorregresivo AR(p) de orden p establece que el valor de la medida a estimar x_t^p en una serie temporal depende linealmente de los valores que tomó en las p mediciones previas [14]. Viene dado por la siguiente expresión:

$$x_t^p = C + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + \epsilon_t \tag{2.15}$$

Donde C es una constante, x_{t-i} el valor i mediciones antes y ϵ_t el término de error actual, aproximado por ruido blanco.

Para trabajar con modelos autorregresivos es fundamental que la serie temporal sea estacionaria. Este concepto se ha introducido en la sección 2.3.1.

La aproximación autorregresiva de una serie temporal se puede complementar con un proceso de media móvil MA(p) de orden p (*moving average* en inglés). En un proceso de media móvil, el valor x_t^p depende linealmente de la media, del término de error actual y de los p términos pasados [21]. Viene dado por la siguiente expresión:

$$x_t^p = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_p \epsilon_{t-p} \tag{2.16}$$

Donde μ es la media y ϵ_i los términos de error.

El orden p de un proceso de media móvil puede ser determinado mediante la autocorrelación explicada en el apartado 2.3.3. Los coeficientes de autocorrelación serán significativos hasta un *lag* p. En el caso de que los coeficientes de autocorrelación decaigan lentamente o muestren un patrón sinusoidal, el proceso proablemente sea autorregresivo en lugar de media móvil.

Generalmente, los modelos predictivos más habituales basados en autorregresión consideran también componentes de media móvil. Un ejemplo recurrente en bibliografía es ARIMA (*Autoregressive Integrated Moving Average* por sus siglas en inglés). Este modelo combina una autorregresión de orden AR(p), una media móvil de órden MA(q) y una integración de orden I(d).
La integración indica cuántas veces es necesario diferenciar la serie temporal original hasta obtener una serie estacionaria sobre la que aplicar AR(p) y MA(q).
El modelo ARIMA se define con la siguiente expresión:

$$x_t^{p'} = C_1 + \phi_1 x'_{t-1} + \ldots + \phi_p x'_{t-p} + \theta_1 \epsilon'_{t-1} + \ldots + \theta_p \epsilon'_{t-p} + \epsilon_t \tag{2.17}$$

De este modo, primero se aplica 2.12 a la serie $\vec{x}(t)$ y se obtiene la serie diferenciada $\vec{x'}(t)$. Este proceso se repite tantas veces como orden d de integración se tenga.
Seguidamente, se calcula $x_t^{p'}$ utilizando 2.17.
Finalmente, se deshace la diferenciación (tantas d veces como se haya realizado previamente). Este paso es el que recibe el nombre de integración I(d). De este modo, se obtiene x_t^p.

El modelo ARIMA no considera de forma directa la estacionalidad. Para resolver esta limitación, se define el modelo SARIMA$(p, d, q)(P, D, Q, m)$ o ARIMA estacional [22]. Los coeficientes de orden (p, d, q) tienen el mismo significado que en el caso de un ARIMA no estacional. En cuanto a (P, D, Q), se trata del orden del proceso autorregresivo estacional AR(P), el orden de integración estacional I(Q), y el orden de la media móvil estacional MA(Q). El número de observaciones por ciclo estacional viene dado por m.
La definición formal del modelo SARIMA viene dada por:

$$\phi_p(B)\Phi_P\left(B^m\right) W_t = \theta_q(B)\Theta_Q\left(B^m\right)\omega_t \tag{2.18}$$

Donde ω_t representa el ruido blanco y W_t el diferenciador de la serie en el paso t-ésimo. B es un operador relacionado con el retardo (*lag*). Tanto ϕ_p como Φ_P, θ_q y Θ_Q son los términos polinómicos que representan los procesos de autorregresión y de media móvil, general y estacional respectivamente.

2.5. Modelos de estado del arte

En las secciones previas, se han presentado algunos de los modelos clásicos más extendidos para la predicción de series temporales.
En las siguientes páginas, se presentan algunos de los modelos de estado del arte más citados. Debido a que el desarrollo formal de estos modelos es más extenso y abarcar cada uno en detalle supondría una memoria por sí solo, se describen sus características principales y se facilitan las referencias bibliográficas convenientes para profundizar más en los mismos.

2.5.1. Prophet

Prophet es un modelo de predicción de series temporales desarrollado por el equipo de Ciencia de Datos principal de META y está basado en la descomposición ETS vista en la sección 2.3.2, a la que añade una componente adicional para dar cuenta de las fechas de vacaciones, ya que pueden exhibir un comportamiento distinto al del resto de la serie. Concretamente, considera una descomposición ETS aditiva y cada componente se trata de la siguiente manera:

- Prophet considera múltiples estacionalidades. En particular, descompone la estacionalidad anual y semanal en series de Fourier [23]. Esto le permite capturar patrones complejos que se repiten anualmente, tales como los períodos vacacionales y las tendencias anuales.
- Considera una tendencia que puede ser lineal o logística. Identifica una serie de puntos en los datos de entrenamiento a partir de los cuales la tendencia parece cambiar. Se conoce como detección de *changepoints*.
- Prophet admite el uso de series temporales adicionales, que tengan una correlación de Pearson significativa con la serie a predecir, como datos adicionales de entrada. Será necesario por tanto hacer una predicción de estas variables exógenas previamente a realizar inferencia para las serie a predecir. Los modelos que presentan esta características se denominan modelos multivariantes de series temporales.
- Respecto a los períodos vacacionales, la propia API de Prophet dispone preimplementada de una serie de vacaciones por país, pudiendo especificarse manualmente fechas vacacionales locales [24].
- Prophet retorna intervalos de confianza para las predicciones, permitiendo de este modo estimar el nivel de incertidumbre.

2.5.2. Múltiple estacionalidad con BATS y TBATS

Cuando se tienen series temporales para varios años con granularidad del órden del minuto, es frecuente que la serie presente múltiples estacionalidades. Por ejemplo, puede que a ciertas horas, o días de la semana, se suelan obtener medidas elevadas y, del mismo modo, observar que algunos meses o estaciones las observaciones tiedan a tomar valores más elevados.
Este es el caso el consumo energético de las enfriadoras. Durante la noche se ejecutan muchos menos procesos en el clúster, y las temperaturas descienden. Por tanto, durante estas horas se requiere una menor refrigeración y por tanto el consumo desciende. Del mismo modo, en las épocas de año donde las temperaturas son mayores, como el verano, no es posible beneficiarse tan a menudo del fenómeno de *free cooling*.

En el caso de los métodos clásicos que consideran la estacionalidad de forma explícita, como Holt-Winters y SARIMA, sólo se considera una única estacionalidad, esto se ha demostrado formalmente al obtener sus ecuaciones.
Por tanto, si bien es posible entrenar el modelo considerando únicamente la estacionalidad diaria o la semanal y tomar los últimos meses, ya que se quiere predecir con 24 horas de antelación, el único modelo formalmente correcto para esta tarea sería Prophet de entre los descritos.
Sin embargo, existen algunos modelos basados en los clásicos, que añaden términos adicionales y otras correcciones para suplir esta carencia. De este modo, se introducen BATS y TBATS, cuya siglas en inglés significan "*exponential smoothing state-space model with Box-Cox transformation, ARMA errors, Trend, and Seasonal components*", es decir, una combinación de suavizado exponencial, media móvil y autorregresión donde se introducen términos adicionales para dar cuenta de varias componentes estacionales. Además, se realiza la Transformación Box-Cox.

Para profundizar en el desarrollo formal de los métodos BATS y TBATS es conveniente consultar la referencia [25] de bibliografía. La diferencia principal entre ambos modelos es que TBATS representa cada componente estacional como una serie de Fourier [7]. Esto permite que, cuando se trabaja con datos con altas frecuencias estacionales, o donde se dispone de un número elevado de medidas TBATS requiera de menor tiempo de entrenamiento, ya que su orden de convergencia es mayor.
A continuación, se enumeran las características principales del modelo BATS:

- BATS comienza realizando una transformación Box-Cox [26] a los datos de entrenamiento y ajustando estos a un modelo Holt-Winters (explicado en sección 2.4.1). Las transformaciones Box-Cox son una familia de transformaciones potenciales empleadas para estabilizar la varianza de la serie. En particular, es frecuente utilizar una modificación de la transformación Box-Cox propuesta por [27] y dada por la siguiente expresión:

$$\omega_i = \begin{cases} \log(x_i) \text{ si } \lambda = 0 \\ \left(\text{signo}(x_i)\,|x_i|^{\lambda} - 1\right)/\lambda \quad \text{en otro caso} \end{cases} \tag{2.19}$$

  Donde w_i es la medida x_i transformada y λ no negativo.

- Una vez ajustado el modelo Holt-Winters, se calculan los residuos y se ajusta a estos un modelo autorregresivo de media móvil (ARMA) con el fin de explicar dependencias temporales que no hayan sido contempladas con el modelo de suavizado exponencial.

- BATS emplea como *framework* el modelado en espacio de estados (*state-space modelling*). Consiste en suponer que la serie temporal observada es un conjunto de datos que depende de otro conjunto de factores no observados. El modelado en espacio de estados formaliza la relación entre las medidas de la serie y estos factores. Se emplea como *framework* para ajustar el modelo ARMA a los residuos.

2.6. Criterio de información de Akaike

Cada uno de los modelos descritos previamente, requiere de un algoritmo para estimar sus parámetros. Ejemplos de ello son las ecuaciones de Yule-Walker para calcular los parámetros de una media móvil o de una autorregresión simple y el método de Newton truncado de gradiente conjugado para obtener los coeficientes de un modelo Holts-Winters [29].
En todos los casos, es necesario además escoger la combinación de hiperparámetros (eg. *lags*, tendencia lineal o logística o los valores $(p, d, q)(P, D, Q, m)$ en un modelo SARIMA) que minimicen el error de las predicciones en los datos de test.
Una forma de realizar esto, sugerida por [15], es realizar validación cruzada de manera continua (*cross-validation on a rolling basis*) con los datos disponibles. Consiste en entrenar el modelo con una pequeña cantidad t de medidas consecutivas (empezando en la primera disponible en el

tiempo) como conjunto de entrenamiento de la serie temporal y predecir con este las n medidas siguientes. Posteriormente, se entrena con las primeras $t+n$ medidas y se realizan predicciones hasta $t+2n$ y así sucesivamente. De este modo, se obtienen tantos *folds* como veces se pueda tomar n nuevas medidas en el *dataset*. Además, se evita realizar muestreo aleatorio, el cual no conserva las dependencias temporales entre las medidas próximas. Un ejemplo de esto se observa en la figura 6.
Para cada una de estas divisiones entre datos de entrenamiento y test, se computa el error cuadrático medio o la raíz cuadrada de este, dada por 2.9.

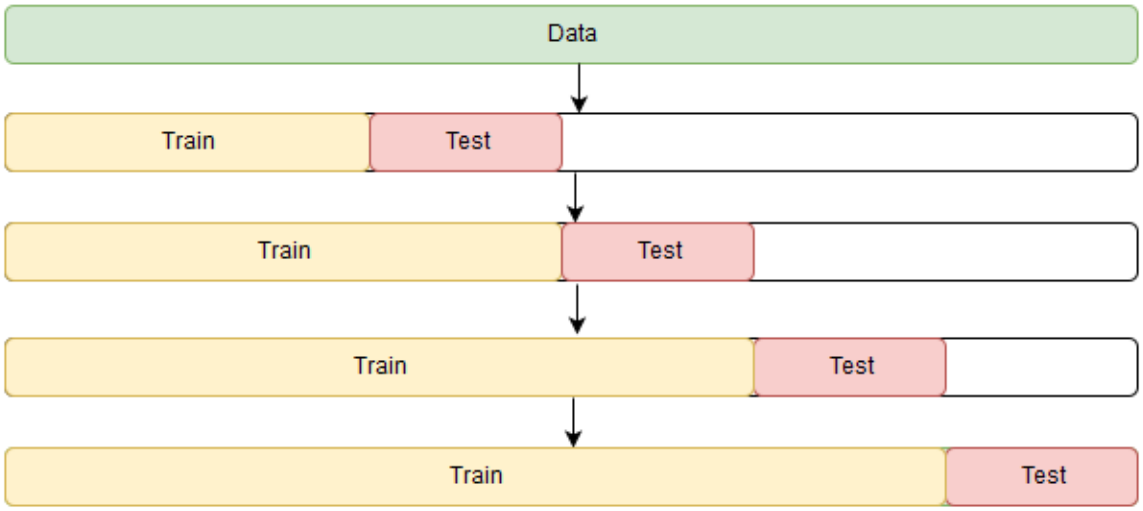

Figura 6: Representación gráfica de la realización de una validación cruzada de manera continua. Aportado por la referencia [15] de bibliografía.

En el caso de modelos autorregresivos, otra forma de seleccionar tanto la combinación de hiperparámetros, como el propio modelo, es el criterio de información de Akaike [30] o AIC. Se define mediante la siguiente expresión:

$$\mathrm{AIC} = T \log\left(\frac{\mathrm{SSE}}{T}\right) + 2(k+2) \tag{2.20}$$

Donde T es el número de medidas empleadas para entrenar el modelo, SSE es la suma del cuadrado de los errores para cada una de las predicciones de las medidas de test y k es el número de parámetros del modelo. Bajo este criterio, el modelo a seleccionar será aquel que minimice el AIC. Además de premiar aquellos modelos que presenten un menor error en las predicciones para el conjunto de test, penaliza aquellos modelos con más parámetros, y por tanto más complejos a priori.

3. Desarrollo

Todos los datos empleados en esta memoria pertenecen a mediciones realizadas en el clúster y las enfriadoras del Centro de Computación de Galicia (CESGA). Estos datos se han recibido en un fichero de Apache Parquet.

3.1. Descripción de los Datos

El fichero Apache Parquet con los datos originales presenta el esquema Spark mostrado en la figura siguiente:

```
root
 |-- node: string (nullable = true)
 |-- start_time: timestamp (nullable = true)
 |-- end_time: timestamp (nullable = true)
 |-- power: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- Power13: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- Power14: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- in: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- out: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- evaporator: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- ambient: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- Compressor1: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
 |-- Compressor2: map (nullable = true)
 |    |-- key: timestamp
 |    |-- value: float (valueContainsNull = true)
```

Figura 7: Esquema Spark del fichero con los datos originales empleados para realizar el proyecto.

Este fichero contiene mediciones cada 15 segundos, entre las fechas 01/01/2018 y 01/06/2021 de las variables a predecir (consumo de las enfriadoras y presión de sus compresores). Las columnas del esquema tienen los siguientes significados:

- La columna "*node*" contiene el nombre del nodo del clúster en el que se toman los datos de consumo en vatios (eg. c7102). En total, se tienen 288 nodos distintos. En el caso de datos de temperatura en grados Celsius y presión en bar de las dos enfriadoras, esta columna muestra los valores "1" y "2" respectivamente. En el caso de los datos de consumo en kilovatios de las enfriadoras 1 y 2, esta columna muestra el valor "*basement*".
- Las columnas "*start time*" y "*end time*" indican la fecha de inicio y final de medida para cada serie temporal, siendo en todos los casos 01/01/2018 y 01/06/2021 respectivamente.

- La columna "*power*" incluye los consumos en vatios de los nodos del clúster. Como se puede observar en el esquema Spark, cada nodo (fila del *DataFrame* de Spark) presenta un diccionario en esta columna. Dicho diccionario tiene como claves la fecha, hora, minuto y segundo de la medida. Los valores son las medidas registradas. Esta misma estructura se repite para cada una de las series temporales contenidas en este *DataFrame*. El método para deshacer esta estructura de datos y poder operar con las series temporales es una funcionalidad de la API de Spark SQL que se discutirá en las próximas secciones.

- Las columnas "*Power 13*" y "*Power 14*" se corresponden con el consumo en kW de las dos enfriadoras. Como se ha indicado, sólo debe presentar datos en la fila donde la columna "*node*" toma el valor "*basement*". En específico, presentará un diccionario con la serie temporal de consumo de la enfriadora correspondiente.

- Las columnas "*in*", "*evaporator*", "*out*" y "*ambient*" representan las temperaturas T_{in}, $T_{evaporator}$, T_{out} y $T_{ambient}$ descritas en la Subsección 1.2. Sólo presentan datos en las filas donde la columna "*node*" toma el valor "1" o "2", siendo este un diccionario que contiene la serie temporal de medidas de temperatura en grados Celsius.

- Las columnas "*Compressor 1*" y "*Compressor 2*" representan las presiones en bar registradas por los compresores 1 y 2 de ambas enfriadoras. Sólo presentan datos en las filas donde la columna "*node*" toma el valor "1" o "2", siendo este un diccionario que contiene la serie temporal de medidas de presión en bar.

3.2. Preprocesado con Apache Spark

Para realizar el procesado ETL (por sus siglas en inglés *Extract, Transform, Load*) se ha utilizado un nodo *driver* con 16GB de memoria RAM y 4 nodos ejecutores con 8 GB de memoria cada uno. Se ha empleado la API de Spark SQL para Python por ser la más adecuada para manipular datos en formato tabular.

En primer lugar, se han preprocesado las series temporales de consumo en vatios de cada uno de los 288 nodos del clúster. Como se desea sumar todas para obtener una única serie temporal con el consumo total $P_{clúster}$, es necesario obtener cada una de las 288 series en un formato tabular, donde se tenga una columna con el tiempo (fecha, hora, minuto y segundo, eg. "2018-01-01 00:00:00"), y otra con las medidas. La columna con el tiempo debe tomar los mismos valores para cada serie, de lo contrario no sería posible sumarlas sin tomar algún criterio adicional.

Al cargar el fichero Apache Parquet con los datos originales, se genera un *DataFrame*. Mediante la función de Spark SQL `createOrReplaceTempView()` se genera una vista temporal local del *DataFrame*, pudiendo interaccionar con esta mediante una sintaxis similar a la de ANSI SQL. Como se ha explicado en la sección Subsección 3.1, cada una de las series se encuentra en un diccionario. Para transformar este diccionario en una serie temporal en formato tabular, existe en SPARK SQL la sentencia `EXPLODE`.
De este modo, se obtiene la serie temporal de consumo de un nodo con varios registros de consumo por minuto. La frecuencia de muestreo, varía dependiendo del nodo o enfriadora. Sin embargo, en todos los casos, las series temporales tienen aproximadamente 1100000 filas originalmente.

Una vez se dispone de la serie temporal, es necesario agrupar las medidas con una frecuencia fija. Se ha decidido agrupar las medidas con Spark cada 30 minutos (en el minuto 0 y 30 de cada hora), para pasar a un *DataFrame* de Spark con 25824 medidas en cada serie.
Para ello se utiliza una función ventana, la cual requiere un método de agrupación. En todos los casos la agrupación sugerida ha sido la media de las medidas durante los 30 minutos, con la excepción de la presión, donde se usa el máximo de cualquiera de los 4 compresores para evitar que un máximo registrado en cualquiera de estos quede suavizado al tomar el valor medio.

Sería también posible utilizar la función `resample()`, de PySpark Pandas [31], pero esta funcionalidad ha sido implementada recientemente y no se encuentra aún instalada en los módulos de las instancias que se ejecutan en los *pods* del clúster disponible.

Una vez agrupadas las medidas cada 30 minutos, existe la posibilidad de que por fallos en los sensores haya ventanas temporales donde no se haya registrado ninguna medida. Para asegurar que estos escenarios no pasen inadvertidos, y que todas las series temporales finales tienen el mismo número de registros, se define manualmente una columna de tiempos entre los rangos de fechas deseados, con registros cada 30 minutos, y se realiza una sentencia `LEFT JOIN` entre dicha columna de tiempos y el *DataFrame* de Spark con la serie temporal agrupada cada 30 minutos, empleando su columna temporal.

En el caso del consumo de los nodos $P_{clúster}$, conforme se van obteniendo las series temporales procesadas para cada nodo, se van sumando en un *DataFrame* que almacena el acumulado de consumo en vatios. Debido a que el conjunto de metadatos y registros de los 288 nodos puede requerir mayor memoria a la disponible entre todos los recursos del clúster, este proceso se ha realizado en *batches* de 24 series temporales.
De modo similar, se procede para obtener la serie procesada de cada una de las temperaturas T_i en grados celsius, el consumo suma de las 2 enfriadoras $P_{chiller}$ en kW y las presiones máximas registradas en cualquiera de las 4 enfriadoras (se toma el máximo para cada tiempo de entre las series correspondientes a cada enfriadora).
De este modo, se tienen 7 series temporales con 25824 registros entre 01/01/2018 y 01/06/2021. A diferencia del tamaño original, el tamaño *DataFrame* final es compatible con la librería Pandas de Python.

Gracias al uso de Apache Spark, se he logrado ejecutar toda la ETL en un tiempo total de 1187 minutos, 1179 de los cuales fueron necesarios para el preprocesado del consumo de los nodos.

En el Apéndice A se muestra un ejemplo del script empleado para procesar las series temporales de consumo eléctrico en vatios de los nodos del clúster.

3.3. Tratamiento de *missing data* y *outlayer*

Tras realizar el preprocesado con Apache Spark y obtener un *DataFrame* de Pandas con 7 series temporales y 25824 filas, se ha procedido a la eliminación de *outlayers*. Para ello, se ha consultado con los administradores del clúster, quienes, empleando la documentación técnica de componentes y refrigeradoras, han indicado los siguientes criterios:

- Respecto al consumo total de los nodos $P_{clúster}$, valores inferiores a 42 kW no deben ser considerados. En particular, valores de 0 o ausencia de medidas suelen corresponderse con caídas de tensión en el Centro de Supercomputación, los cuales no son representativos de la serie.

- Respecto al consumo total $P_{chiller}$ de las enfriadoras, cualquier registro superior a 200 kW es un error de medida del sensor correspondiente. Del mismo modo, cualquier consumo inferior a 10 kW debe considerarse incorrecto, ya que ese es el consumo base únicamente de la bomba. No se eliminan los registros con un percentil superior al 95 debido a que lo que resulta particularmente útil de predecir el máximo de presión horariamente es anticipar períodos de presión particularmente elevada.

- Ninguna temperatura T_i registrada correctamente debería ser inferior a -10 ºC. Para cada unas de las 4 temperaturas distintas T_{in}, $T_{evaporator}$, T_{out} y $T_{ambient}$ se tienen realmente 2 series temporales, una por enfriadora. Para determinar la medida de qué serie es la correcta en cada tiempo, se ha empleado el criterio mostrado en el código Python del Apéndice B.

Respecto al tratamiento de datos faltantes (*missing data*), se ha empleado una interpolación con las medidas próximas empleando la función `interpolate(method="time")` de la librería Pandas.
Es importante notar que, pese a que el preprocesado mediante Apache Spark se ha realizado para obener medidas cada 30 minutos (para disponer de un *DataFrame* de tamaño compatible con Pandas), una vez en Pandas se ha hecho un `resample()` a 1 hora debido a que esta granularidad es el requisito para el proyecto.

4. Resultados y análisis

En esta sección se muestran los Análisis de Datos Expolatorios (EDA por sus siglas en inglés) y los resultados de los modelos aplicados para predecir el consumo suma en kW de las dos enfriadoras y la presión máxima que alcanzan los compresores para las próximas 24 horas, tomando como fechas de referencia el 31 de mayo de 2021 (24 puntos) y el 22 de septiembre de 2020 respectivamente.
Durante los EDA, se muestran las series temporales endógenas (consumo de las enfriadoras y presión) y su correlación de Pearson con el resto de series temporales cuando es necesario. Respecto a las variables exógenas (temperaturas, consumo de los nodos), se muestran en el Apéndice C.

4.1. *Exploratory Data Analysis* y predicción del consumo de las enfriadoras

En primer lugar, se ha realizado el test de Dickey-Fuller para la serie temporal del consumo suma de ambas enfriadoras en kW, obteniéndose un valor p de $1{,}35 \times 10^{-11}$. Al ser inferior a 0.05, es posible rechazar la hipótesis nula de que la serie tiene una raíz unitaria. Por tanto la serie es estacionaria y no es necesario diferenciar. Esto también quedará patente cuando se observen los hiperparámetros del SARIMA que han minimizado el AIC (obteniéndose un AIC de 15298) tras hacer un *grid search*, donde el coeficiente de integración vale 0.
La serie temporal de consumo $P_{chiller}$ de las enfriadoras se muestra en la 8.

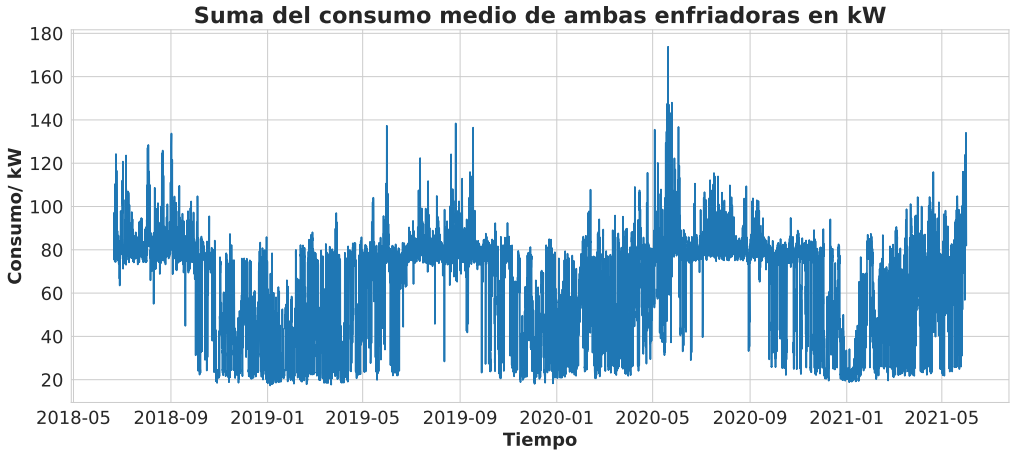

Figura 8: Consumo suma $P_{chiller}$ de las enfriadoras en kW entre mayo de 2018 y mayo de 2021 (todo el conjunto de datos disponible).

Debido a la granularidad horaria, habiendo 25824 puntos en la figura, se muestran las medidas del último mes disponible en la figura 9.

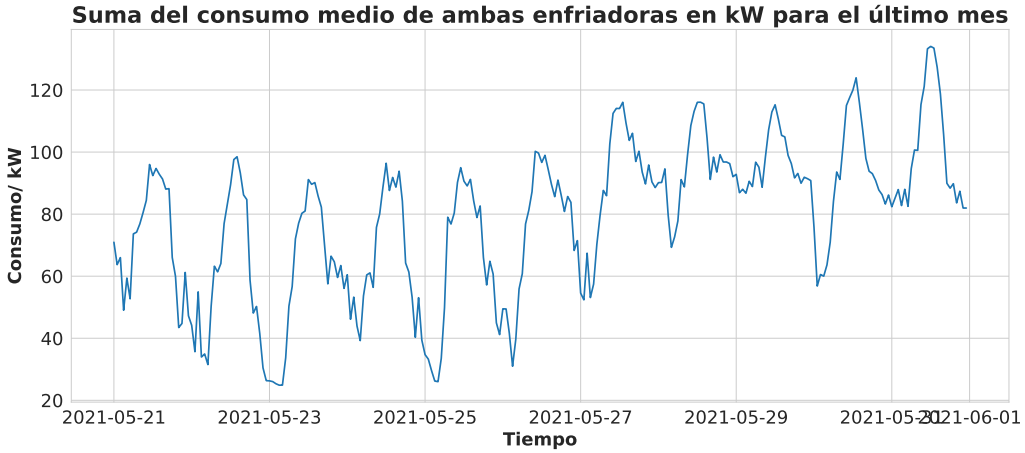

Figura 9: Consumo suma $P_{chiller}$ de las enfriadoras en kW para el último mes disponible.

Es posible notar que existen varias estacionalidades. Una es diaria, siendo las horas de día donde se registra un mayor consumo, mientras que en la noche este cae. La otra es anual, siendo los veranos y principios de otoño donde se registran los mayores consumos. Si agrupamos mensualmente y comparamos año a año, notamos que esta segunda estacionalidad parece cumplirse correctamente todos los años. También notamos que no hay ningún año que, en principio, presente un comportamiento diferente al de los demás. Esto se muestra en la figura 10. Se puede notar un ligero aumento del consumo respecto a los años anteriores en la misma época para los meses abril, mayo y junio de 2020. Esto puede deberse al uso intensivo del clúster durante el período de máxima incidencia de Covid-19.

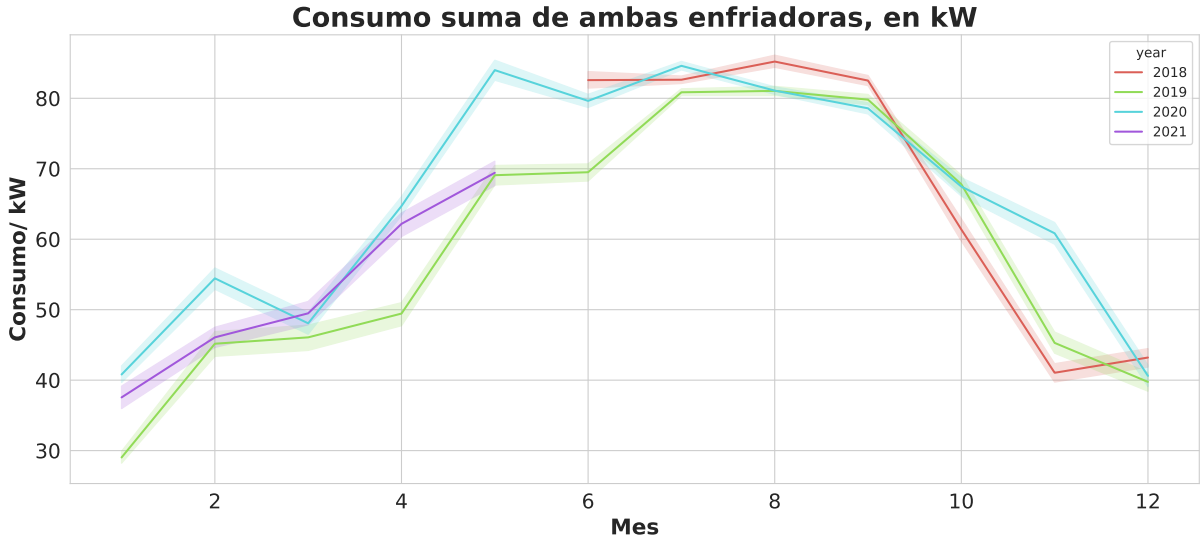

Figura 10: Agrupación del consumo suma $P_{chiller}$ de las enfriadoras en kW mensualmente y comparación año a año.

La figura 11 muestra el diagrama de cajas correspondientes a la tendencia semanal y horaria. Nuevamente, se observa que las semanas pertenecientes a los meses más cálidos del año son aquellas donde el consumo es significativamente más alto para las refrigeradoras. Además, se observa que, en cuanto a tendencia diaria, las enfriadoras trabajan más entre las 10 de la mañana y las 6 de la tarde, estando sus máximos entre las 2 y las 5 de la tarde, donde se suelen alcanzar las temperaturas máximas.

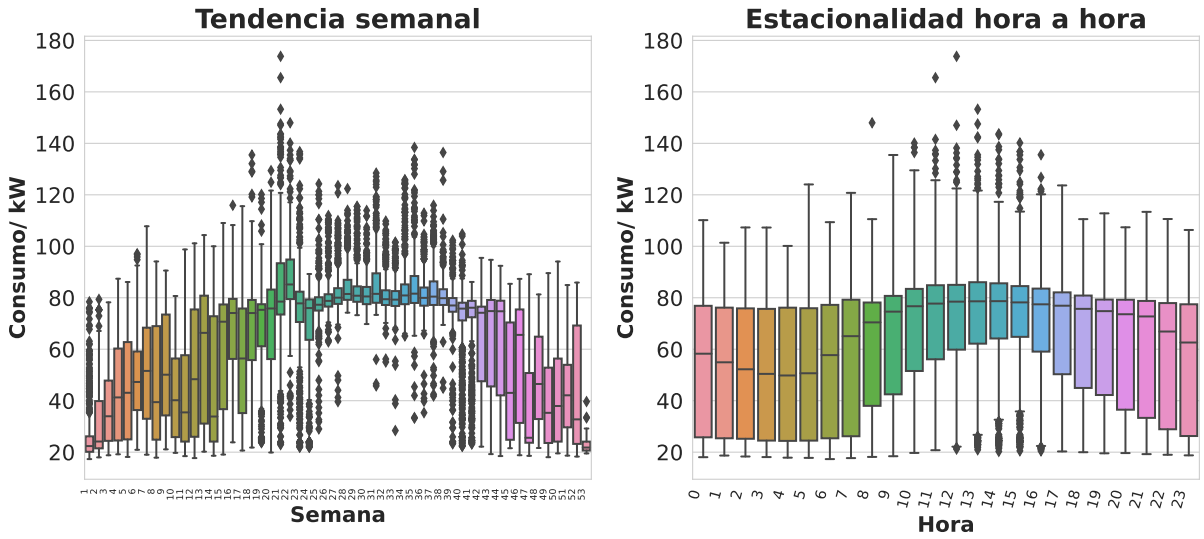

Figura 11: Diagramas de caja de la tendencia semanal y horaria de consumo $P_{chiller}$ de las enfriadoras en kW en el período 2018-2021.

Para sintetizar las ideas ya explicadas, se muestra en la figura 12 la descomposición ETS aditiva de consumo $P_{chiller}$ donde puede notarse la significativa contribución de la estacionalidad.

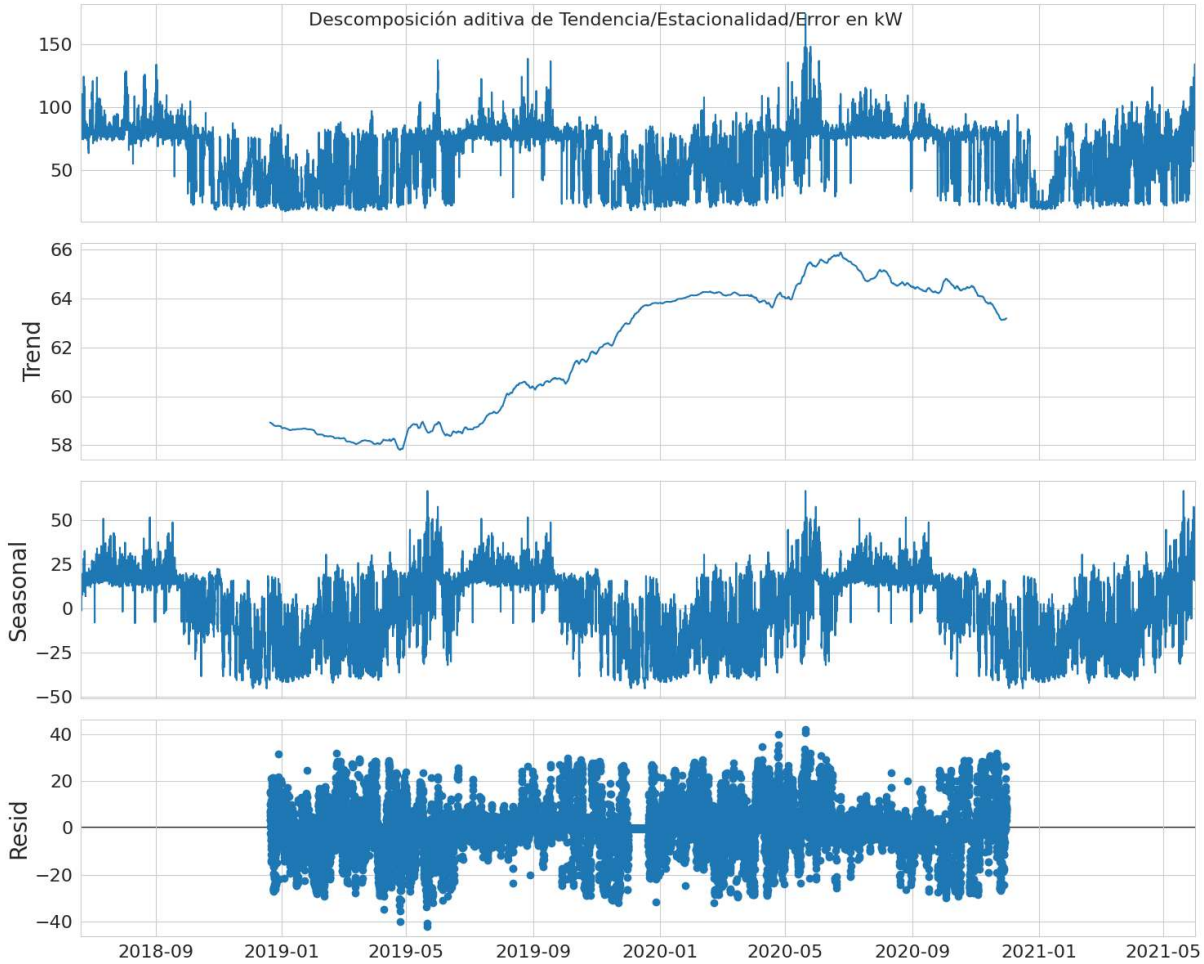

Figura 12: Descomposición ETS aditiva del consumo $P_{chiller}$ en el período 2018-2021.

Respecto a la autocorrelación, la figura 13 muestra la ACF para los primeros 150 *lags*, obser-

vándose un marcado caracter sinusoidal relacionado con un proceso autorregresivo, como se ha explicado en secciones anteriores. Por simplicidad de la figura, no se muestran más *lags*, pero estos han continuado oscilando y tomando valores significativos (superiores al ruido blanco) hasta 8760, que coincide con 365 días atrás y confirma la estacionalidad anual.

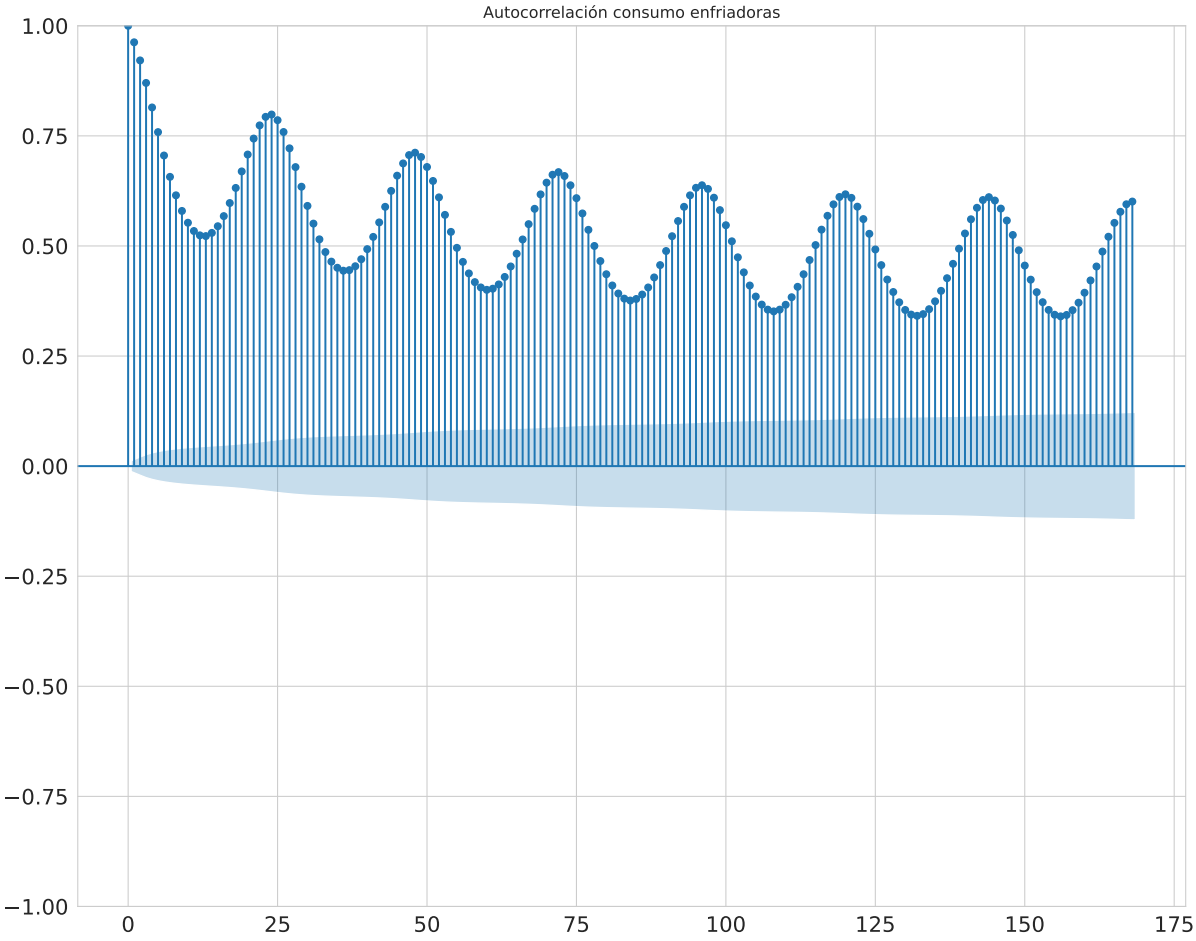

Figura 13: Diagrama ACF de consumo $P_{chiller}$ para los primeros 150 *lags*.

Finalmente, y tras este análisis, se muestran en la figura 14 los valores predichos por los modelos aplicados frente al valor real (*ground truth*) para el día 31 de mayo de 2021. Es importante notar que, tras consultar con los técnicos del CESGA, no existe una relación causal entre las demás variables estudiadas y el consumo de las enfriadoras, por lo que, a diferencia del estudio de la presión, para la predicción del consumo $P_{chiller}$ es conveniente utilizar series temporales de variable endógena únicamente. En el caso del modelo Sarima, tras realizar un *grid search*, se ha obtenido que, el SARIMA(1,1,0)(2,0,0) ha sido el que ha obtenido el menor AIC, de 15298.

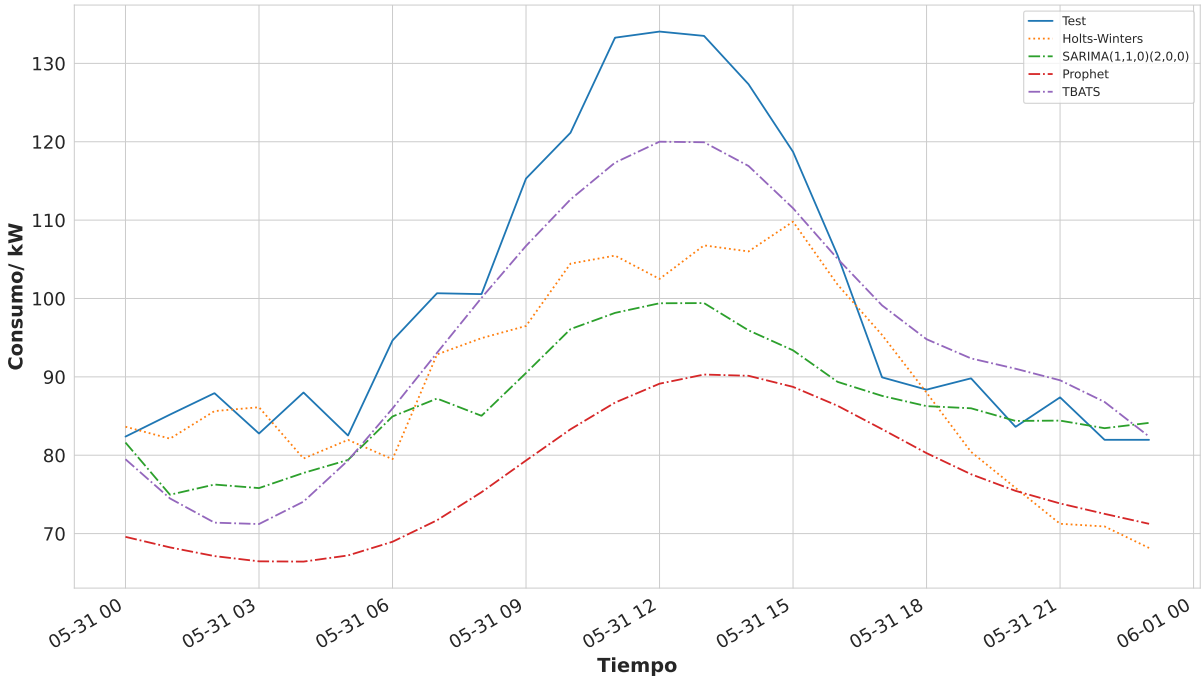

Figura 14: Comparación de las predicciones de Prophet, SARIMA(1,1,0)(2,0,0), Holt-Winters y TBATS con el valor real de consumo $P_{chiller}$ para las 24 horas del 31 de mayo de 2021.

En la figura siguiente se recogen los valores de RMSE y sMAPE porcentual obtenidos por cada modelo.

Modelo	**RMSE**	**sMAPE porcentual**
Holt-Winters	14.18	11.16
SARIMA	17.90	13.25
Prophet	25.92	24.72
TBATS	9.18	8.05

Figura 15: RMSE y sMAPE porcentual obtenido con los distintos modelos aplicados para predecir $P_{chiller}$.

De este modo, el modelo más adecuado para poner en producción es TBATS, teniendo un sMAPE próximo al 9 %. Esto es de esperar ya que en el caso del consumo, se está trabajando con una única serie temporal con múltiples estacionalidades y, como se explicó previamente, TBATS es el modelo adecuado cuando se presentan múltiples estacionalidades.

4.2. *Exploratory Data Analysis* y predicción de la presión de los compresores

Nuevamente, en el caso de la presión máxima p_{max} alcanzada por los compresores de forma horaria en el 22 de septiembre de 2020, el valor p ha sido inferior a 0.05, específicamente $1{,}88 \times 10^{-16}$, de modo que la serie es estacionaria.

El proceso de EDA es similar al visto para $P_{chiller}$, pero en este caso se ha sugerido considerar variables exógenas adicionales para el modelo Prophet. Por tanto, se mostrará una matriz de correlación en formato *heatmap* para seleccionar como variables exógenas aquellas con una correlación de Pearson superior a 0.5 con la presión p_{max}.
Debido a que el *free cooling* se da principalmente en invierno, cuando las temperaturas son más bajas en Galicia, es de esperar que los compresores trabajen más en verano. De modo que deben priorizarse dichos meses en el análisis. Estos se muestran en color naranja en la figura 16, que representa el conjunto de los datos disponibles de presión.

Sin embargo, de todos estos datos, se usará el subconjunto de entrenamiento que termina el 21 de septiembre de 2020, ya que se ha considerado que predecir fechas de 2021, donde aún no se disponían de medidas de verano en el momento de realización del proyecto, no es provechoso por los motivos expuestos relacionados con el *free cooling*. Es decir, la verdadera utilidad de predecir la presión p_{max} es obtenerla en verano, cuando los compresores son usados intensivamente.

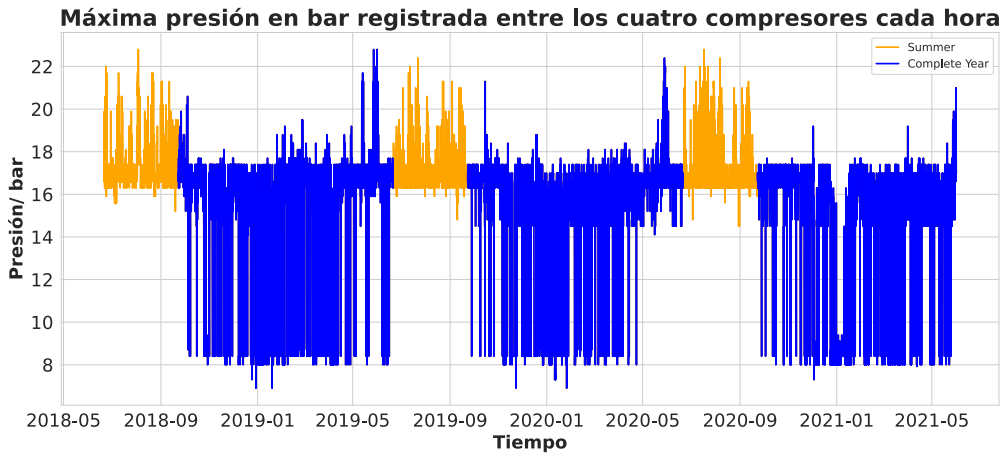

Figura 16: Histórico de p_{max} en bar para los 4 compresores.

Nuevamente, resulta difícil apreciar patrones estacionales en una figura con una cantidad tan elevada de medidas, a excepción de la clara estacionalidad anual, donde las presiones son más altas en verano (naranja). Por tanto, se muestra un subconjunto de medidas en la figura 17.

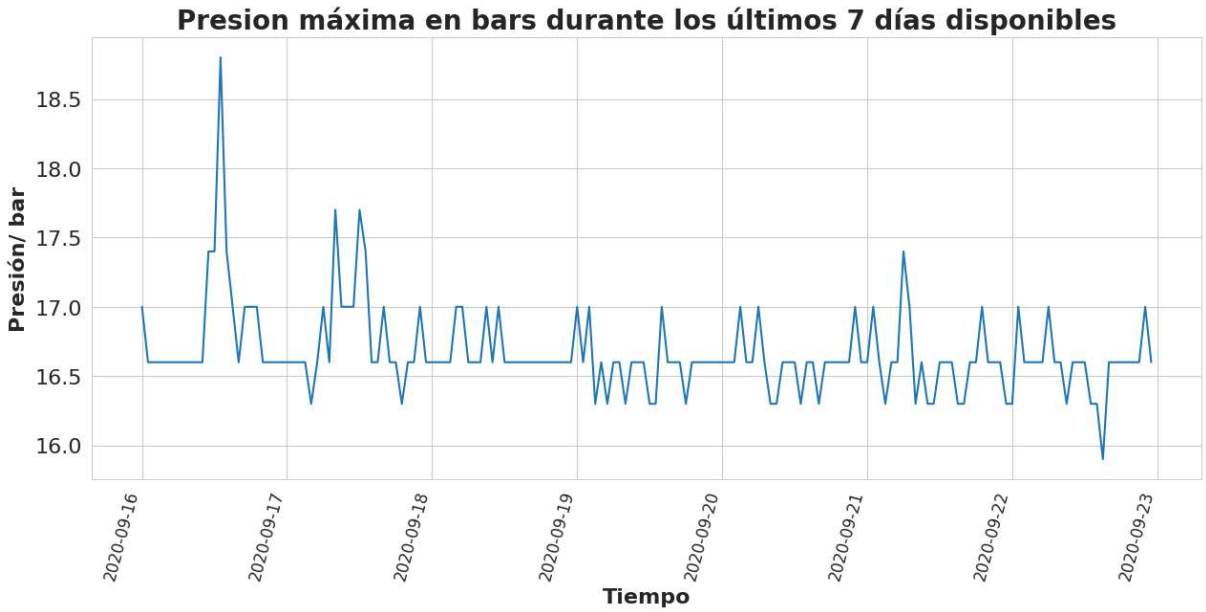

Figura 17: Subconjunto de medidas de p_{max}.

De modo similar a la $P_{chiller}$, en la figura 18 se muestra la agrupación mensual de medidas de presión p_{max} año a año (podemos ignorar el 2021 para nuestros fines) y en la figura 19 el diagrama de cajas semanal y horario calculado entre 2018 y septiembre de 2020.

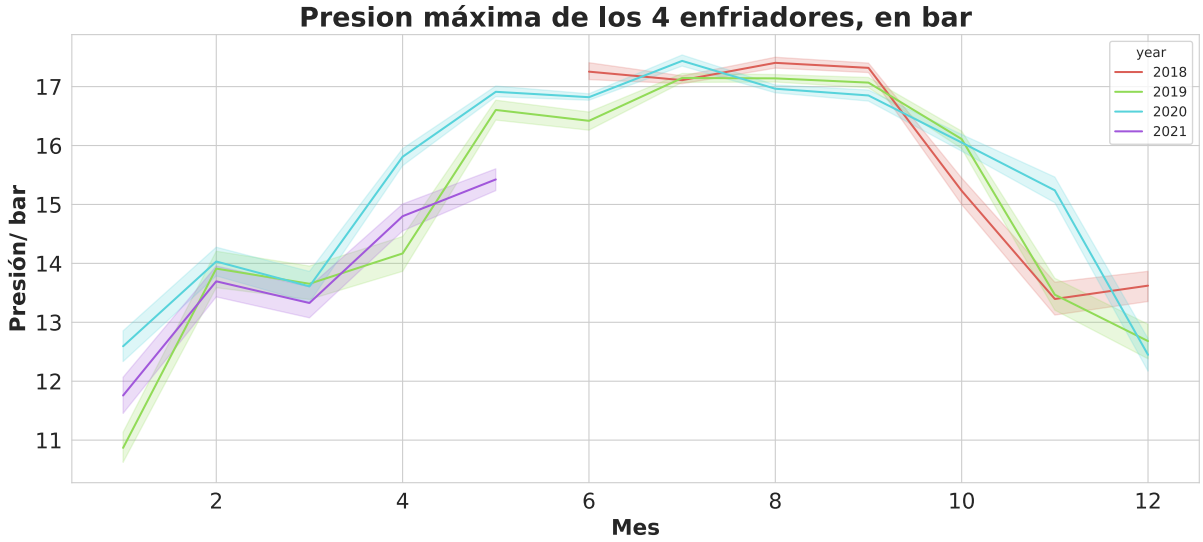

Figura 18: Agrupación del consumo suma p_{max} de los compresores en bar mensualmente y comparación año a año.

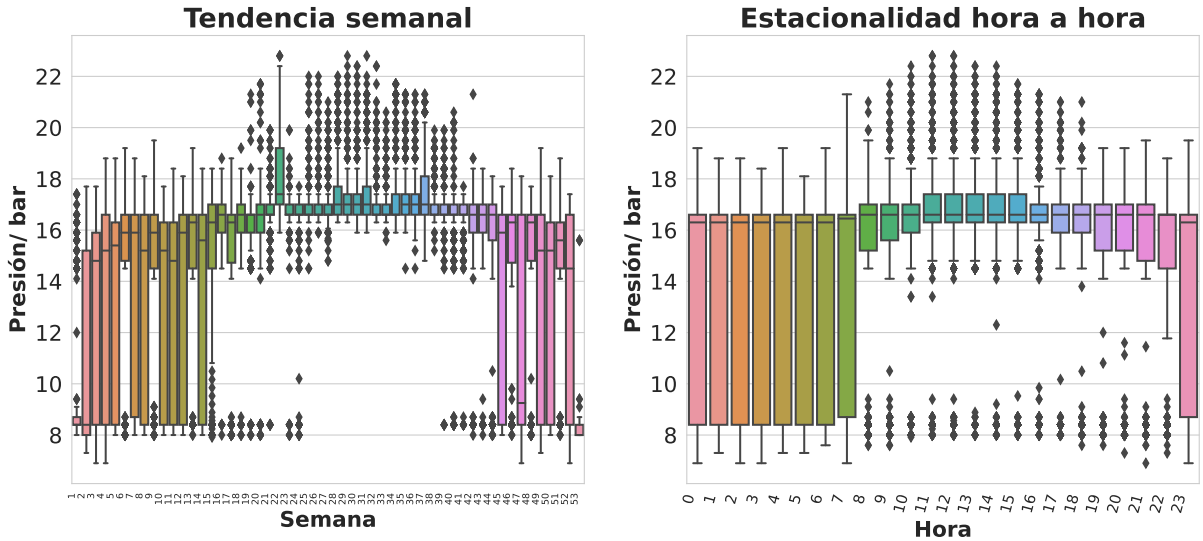

Figura 19: Diagramas de caja de la tendencia semanal y horaria de consumo p_{max} de las enfriadoras en kW en el período junio 2018-septiembre 2020.

El diagrama de cajas evidencia de forma clara que las presiones máximas se alcanzan durante el día y en particular en las semanas de final de primavera, verano y principios de otoño. Esto tiene sentido al tener que trabajar más los compresores en ausencia de *free cooling* por haber temperaturas $T_{ambient}$ más elevadas.
En este caso, los *outlayers* a nivel semanal se encuentran en su gran mayoría en los percentiles más elevados, habiendo muy pocos en presiones bajas (que se pueden asociar con el consumo en *standby* de los compresores). En el diagrama de cajas, aunque los *outlayers* se dan más a menudo en valores de presión por encima que por debajo del tercer cuartil, no hay la misma desproporción. Esto quiere decir que la estacionalidad que tiene más impacto en la presión es la anual, registrándose en los compresores una mayor disparidad entre períodos de alta presión y períodos de *standby* que si solamente nos fijmamos a nivel diario con independencia del mes.

Para reforzar estas ideas, la figura 20 muestra la descomposición ETS aditiva, donde la componente estacional anual tiene un peso significativo en la serie temporal. Además, la ACF mostrada en la figura 21 evidencia, con en el caso del consumo, un proceso marcadamente autorregresivo.

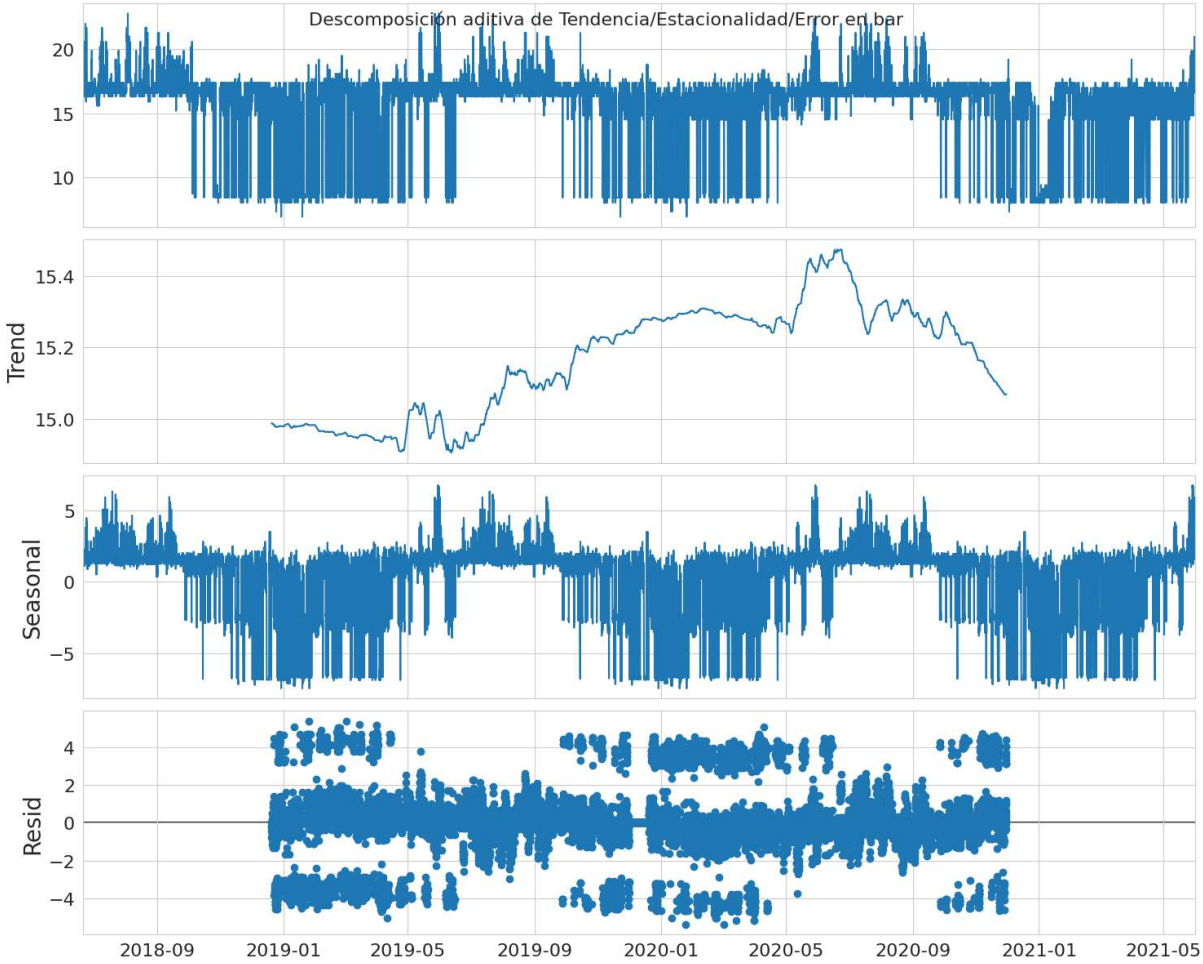

Figura 20: Descomposición ETS aditiva de p_{max}.

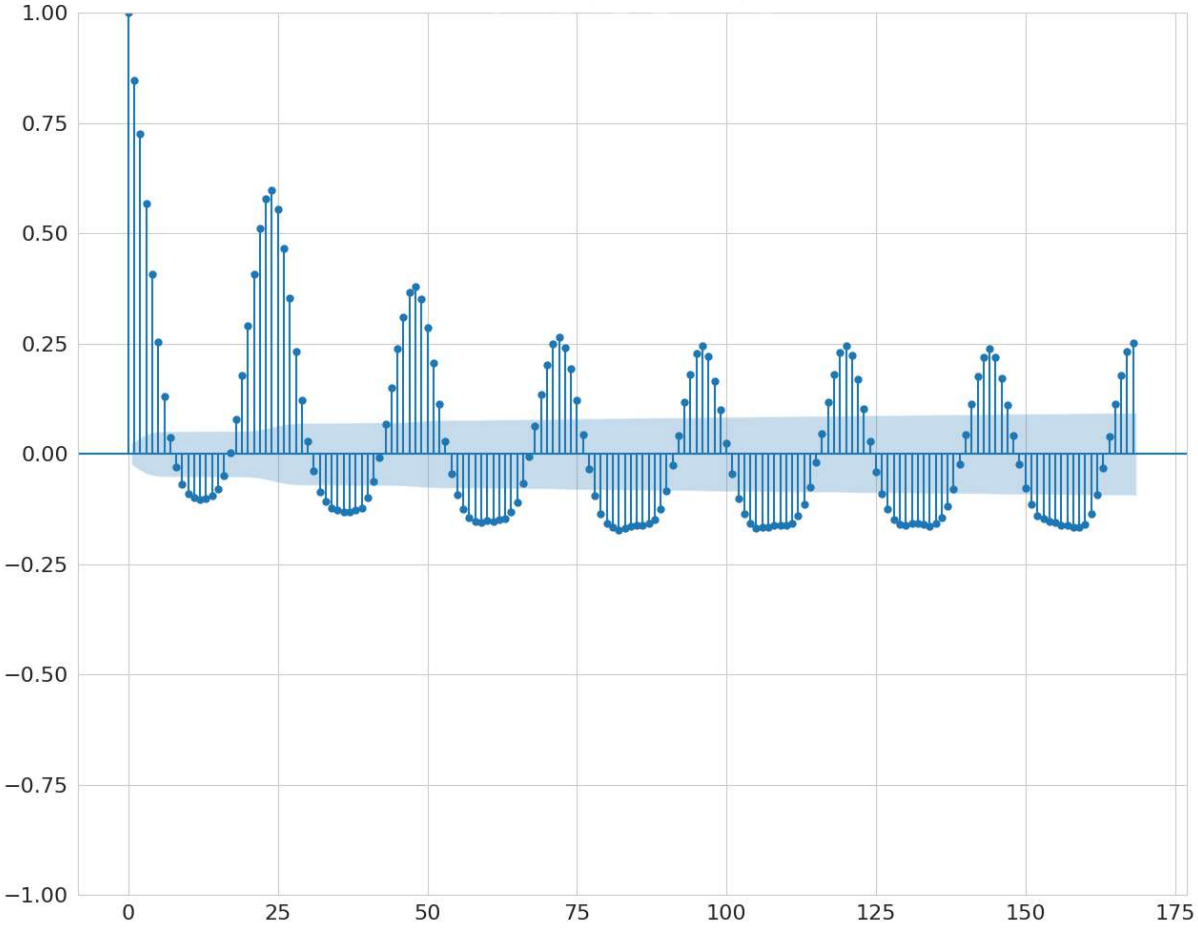

Figura 21: Diagrama ACF de p_{max}.

Antes de mostrar los resultados obtenidos con los distintos modelos, se muestra en la figura 22 la correlación Pearson entre las distintas variables, obteniéndose 0.78 entre p_{max} y $P_{chiller}$ (como era de esperar ya que parte del consumo de la enfriadora se debe al trabajo de los compresores) y 0.74 entre p_{max} y $T_{ambient}$ (también esperable debido a los períodos de *free cooling*).

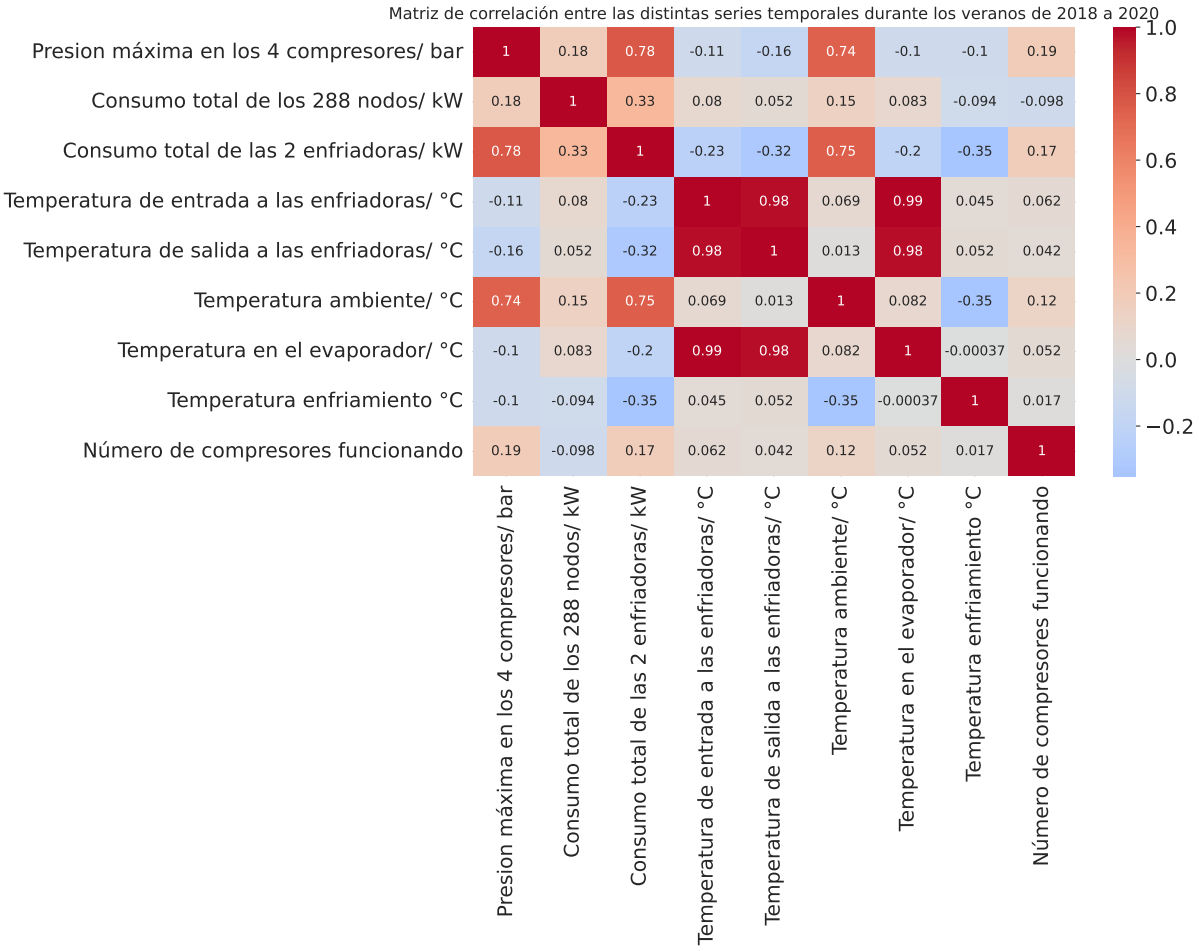

Figura 22: Mapa de calor de correlaciones de Pearson entre p_{max} y las demás series temporales exógenas.

Finalmente, y tras este análisis, se muestran en la figura 23 los valores predichos por los modelos aplicados frente al valor real (*ground truth*) para el día 22 de septiembre de 2021. Es importante notar que, Prophet ha sido empleado utilizando dos regresores (variables exógenas que a su vez han sido predichas para el 22 de septiembre con un Prophet monovariable) adicionales: $T_{ambient}$ y $P_{chiller}$. En cuanto al SARIMA, en este caso SARIMA(3,1,1)(2,0,0) ha sido el que ha obtenido el menor AIC, de 3780.

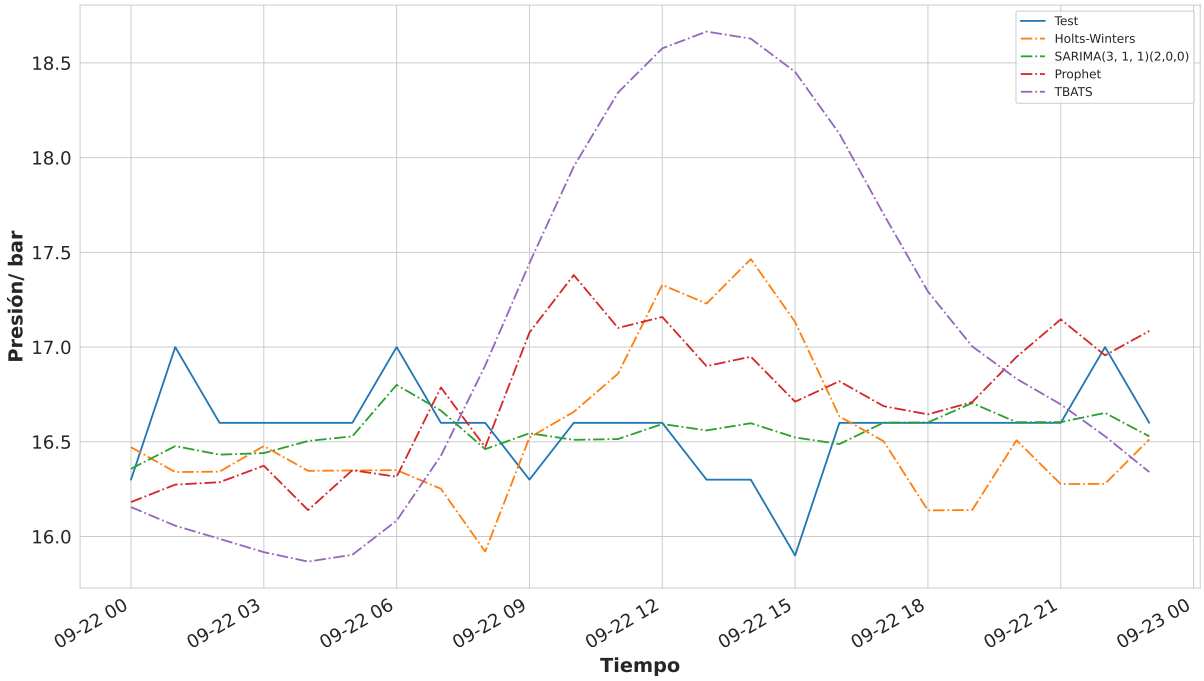

Figura 23: Comparación de las predicciones de Prophet (con regresores exógenos), SARIMA(3,1,0)(2,0,0), Holt-Winters y TBATS con el valor real de presión p_{max} para las 24 horas del 22 de septiembre de 2020.

Modelo	**RMSE**	**sMAPE porcentual**
Holt-Winters	0.55	2.58
SARIMA	0.22	0.94
Prophet	0.48	2.41
TBATS	1.22	5.72

Figura 24: RMSE y sMAPE porcentual obtenido con los distintos modelos aplicados para predecir $P_{chiller}$.

En este caso, es el modelo SARIMA(3,1,0)(2,0,0) el que ha obtenido un menor RMSE y un menor sMAPE porcentual. Esto se debe en parte al marcado carácter autorregresivo de la serie, como se desprende del ACF y que, pese a presentar múltiples estacionalidades, como se ha explicado previamente, en este caso domina claramente el patrón anual sobre el semanal y el diario. De este modo, los modelos que contemplan múltiples estacionalidades, como TBATS y Prophet, no han sido tan beneficiados.
Prophet, al considerar regresores exógenos con una elevada correlación, se ha aproximado más a Holt-Winters y SARIMA que TBATS.

5. Conclusiones y Discusión

En primer lugar, es importante reseñar que la componente principal de este trabajo es la familiarización con el ecosistema de Apache Hadoop y con Apache Spark para manipular las 298 series temporales con cerca de 1100000 registros en cada caso. Esto es tanto para desarrollar ETLs en el ámbito de la Ingeniería de Datos como para productivizar modelos en el ámbito la Ingeniería de *Machine Learning*. Una parte significativa del esfuerzo se ha empleado en optimizar el código para el preprocesamiento de datos, y en entender cómo monitorizorear los *jobs* y prevenir problemas de memoria durante su ejecución en el clúster.

De entre los métodos mostrados, TBATS ha evidenciado ser un método adecuado cuando se presentan múltiples estacionalidades, mientras que los métodos basados en autorregresión como SARIMA han funcionado particularmente bien cuando la ACF ha presentado un patrón sinusoidal.
Como segundo modelo más adecuado en el caso del consumo energético se ha obtenido Holt-Winters. Esto es de esperar, ya que TBATS es en esencia un modelo Holt-Winters con correciones autorregresivas adicionales (y optimización en el cálculo de las estacionalidades mediante el uso de transformadas de Fourier). Debido a que TBATS utiliza un modelo ARMA para ajustar sus residuos, también es entendible que el tercer modelo con mejor sMAPE porcentual haya sido SARIMA, siendo este solamente de 13.25 % frente al 11.16 % de Holt-Winters.
En definitiva, parece que la marcada estacionalidad diaria de esta serie, con un notable pico de consumo entre las 9 y las 15 horas, ha favorecido el desempeño de modelos que priorizan esta estacionalidad.
En el caso de la predicción de la presión, Prophet con regresores exógenos y Holt-Winters han sido el segundo y tercer método con mejores resultados, aunque el sMAPE porcentual mostrado por SARIMA ha sido más de un punto superior en ambos casos. TBATS ha evidenciado un desempeño notablemente inferior.
Es importante notar que la serie de la presión máxima p_{max} es particularmente compleja de predecir, ya que al necesitar reducir la granularidad de la serie y agrupar las mediciones de los 4 compresores tomando máximos cada hora para tener un tamaño de *DataFrame* manejable, se pueden estar enmascarando comportamientos tales como que varios compresores presenten máximos de presión a la vez, que solamente uno lo presente, o la duración en el tiempo de los mismos. Las enfriadoras no se encuentran sometidas a las mismas cargas de trabajo en cada una de estas situaciones, así que la correlación con alguna de las variables exógenas puede verse perjudicada al reducir la granularidad, lo que supone un detrimento para el desempeño de Prophet con variables exógenas.
Además, es importante notar que, frente a la *grid search* que se realiza en casos como SARIMA, donde se selecciona el modelo que minimiza el AIC, en el caso de TBATS es necesario imponer algunas restricciones tales como suponer el modelo libre de tendencia o indicar los períodos estacionales. Esto se debe a que TBATS, como se ha explicado, es una combinación de varios modelos, por lo que su tiempo de entrenamiento es sensiblemente superior, requiriéndose de varias horas de ejecución para obtener los resultados mostrados.

En resumen, se ha realizado un proyecto completo de predicción de consumo energético y presión de los compresores para el Centro de Supercomputación de Galicia, aportándose una solución a un requerimiento real de la institución. Para ello, se han combinado conocimientos teóricos introducidos en el máster junto con aprendizaje autónomo sobre el modelado de series temporales y varias herramientas para operar con grandes volúmenes de datos. En la predicción de ambas series temporales se han logrado resultados satisfactorios, obteniéndose modelos con un error sMAPE porcentual próximo al 1 % y 8 % respectivamente, siendo el segundo caso un problema particularmente complejo de predecir.
Todo el código desarrollado para el proyecto está disponible para su consulta en el siguiente repositorio https://github.com/lmc00/TS-Forecasting y las partes principales del mismo se tratan en los anexos.

Trabajo Futuro

Respecto a la parte puramente del entrenamiento de los modelos, pese a no ser incluídos en la memoria, ya que supondría alargar su extensión y complejidad, sería conveniente probar a aplicar otros modelos de *Machine Learning* como las Máquina de Vector Soporte (SVM). Otro modelo que conviene probar sería pasar a usar procesos gaussianos y aplicar una regresión en este caso.
Además, se han probado algunos modelos de *Deep Learning*, en particular la *Gated Recurring Unit* (GRU) [28] y una *Long Short-Term Memory* bidireccional.
Sin embargo, los resultados han estado lejos de mejorar los mostrados en la memoria. Para lograr un progreso sería necesario realizar *feature engineering*, que, junto con el reentreno con granularidad horaria, probando distintos optimizadores e hiperparámetros, puede suponer un proyecto en sí mismo. Por tanto, se ha omitido y propuesto como futura mejora.

Finalmente, respecto al apartado de puesta en producción de los modelos, un trabajo futuro sería la integración del código implementado con Kedro [32]. Kedro es un *framework* de *Machine Learning* para Python. Es especialmente beneficioso debido a que, una vez desarrollado el código en PySpark, se encarga automáticamente de gestionar todos los parámetros de configuración que el clúster usualmente requiere para ejecutar el código de forma óptima.
Además, facilita la implementación de *pipelines* para el preprocesado y entrenamiento de los modelos siguiendo la Convención de Ingeniería de Datos dada por [33] y el CI/CD de cada una de ellas. Posee preimplementados una gran cantidad de conectores con distintas fuentes de datos (eg. *Amazon S3*, *Google Cloud Storage*, *Azure Blob Storage*) y APIs. De este modo, las peticiones, respuestas, *logs*, almacenamiento de resultados y modelos o las conexiones con otros servicios que usualmente son necesarios en la puesta en producción de un modelo son realizados sin tener que implementar nuevas funcionalidades.

Referencias

[1] ANSI/TIA-942-2005, "*Telecommunications Infrastructure Standard for Data Centers*", pp 110-130, 2005.

[2] "Infraestructuras: Equipos de apoyo del Centro de Supercomputacion de Galicia". https://www.cesga.es/infraestructuras/equipos-de-apoyo/. Fecha de consulta: 20/07/2022.

[3] R. C. Zoie, R. Delia Mihaela and S. Alexandru, "An analysis of the power usage effectiveness metric in data centers". *2017 5th International Symposium on Electrical and Electronics Engineering (ISEEE), Galati, Romania*, 2017, pp. 1-3, doi: 10.1109

[4] A. G. Tato, "Evaluation of Machine Learning Frameworks on Finis Terrae II". *Galicia Supercomputing Center*, pp 6-9, 2017.

[5] Chaerun Nisa, Elsa, Kuan, Yean-Der. (2021). "Comparative Assessment to Predict and Forecast Water-Cooled Chiller Power Consumption Using Machine Learning and Deep Learning Algorithms". *Sustainability.*, pp 8-9, 2021.

[6] "Apache Spark". https://spark.apache.org/. Fecha de consulta: 15/08/2021.

[7] M. Peixeiro. (2022) "How to forecast time series with Multiple Seasonalities". https://towardsdatascience.com/how-to-forecast-time-series-with-multiple-seasonalities-23c77152347e. Fecha de consulta: 12/09/2022.

[8] R.J.Hyndmanand, G.Athanasopoulos,F. "Forecasting: principles and practice". https://otexts.com/fpp2/complexseasonality.html, ch 2, 11. Fecha de consulta: 12/09/2022.

[9] Phumchusri, Naragain, Ungtrakul, Phoom. "Hotel daily demand forecasting for high-frequency and complex seasonality data: a case study in Thailand". *Journal of Revenue and Pricing Management*, vol. 19, pp. 13-15 , 2020.

[10] "Apache Hadoop". https://hadoop.apache.org/. Fecha de consulta: 15/08/2021.

[11] "Apache Hadoop YARN". https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html. Fecha de consulta: 15/08/2021.

[12] "Apache HDFS". https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Fecha de consulta: 15/08/2021.

[13] "Apache Parquet". https://parquet.apache.org/docs/file-format/. Fecha de consulta: 15/08/2021.

[14] M. Peixeiro. "Time Series Forecasting in Python". *O'Reilly Media, Inc.*, 1st edition, pp 4-47, 2022.

[15] D. Sergeyev. "Mlcourse.ai". https://mlcourse.ai/book/, ch. 9. Fecha de consulta: 14/08/2022.

[16] Slater, L. J. and Anderson, B. and Buechel, M. and Dadson, S. and Han, S. and Harrigan, S. and Kelder, T. and Kowal, K. and Lees, T. and Matthews, T. and Murphy, C. and Wilby, R. L. "Nonstationary weather and water extremes: a review of methods for their detection, attribution, and management". *Hydrology and Earth System Sciencesn*, vol. 25, pp. 3987-3935, 2021.

[17] Dickey, D., Fuller, Wayne. (1979). "Distribution of the Estimators for Autoregressive Time Series With a Unit Root". *JASA. Journal of the American Statistical Association*, vol. 74, pp. 428-430, 1979.

[18] Akaike.T. "Seasonal adjustment by a bayesian modelling". *Journal of Time Series Analysis*, vol. 1, pp. 1-13, 1980.

[19] Armstrong, J. S. "Long-range forecasting: From crystal ball to computer". *John Wiley Sons*, vol. 1, pp. 378-379, 1978.

[20] Holt, C. C. "Forecasting seasonals and trends by exponentially weighted averages". *O.N.R. Memorandum No. 52, Carnegie Institute of Technology*, vol. 1, 1957.

[21] PennState Eberly College of Science. "STAT 510. Applied Time Series Analysis". https://online.stat.psu.edu/stat510/lesson/2/2.1, ch. 2. Fecha de consulta: 27/08/2023.

[22] Noor, Talal H., Abdulqader M. Almars, Majed Alwateer, Malik Almaliki, Ibrahim Gad, and El-Sayed Atlam. "SARIMA: A Seasonal Autoregressive Integrated Moving Average Model for Crime Analysis in Saudi Arabia" *Electronics 11*, vol 11, ch. 23. 2022.

[23] Taylor SJ, Letham B. Forecasting at scale. *PeerJ Preprints 5*, vol. 1, pp. 7-15. 2017.

[24] "Prophet". https://facebook.github.io/prophet/. Fecha de consulta: 20/08/2022.

[25] Phumchusri, Naragain Ungtrakul, Phoom. "Hotel daily demand forecasting for high-frequency and complex seasonality data: a case study in Thailand". *Journal of Revenue and Pricing Management*, vol. 1, pp. 13-15. 2020.

[26] Box, G. E. P., Cox, D. R. "An analysis of transformations. Journal of the Royal Statistical Society". *Series B, Statistical Methodology*, vol. 26, pp. 211–252. 1964.

[27] Bickel, P. J., Doksum, K. A. "An analysis of transformations revisited". *Journal of the American Statistical Association*, vol. 76, pp. 296–311. 1981.

[28] Aston Zhang and Zachary C. Lipton and Mu Li and Alexander J. Smola. "Dive into Deep Learning". *Cambridge University Press*, vol. 1, pp. 325-357. 2023.

[29] Stephen G. Nash. "A survey of truncated-Newton methods". *Journal of Computational and Applied Mathematics*, vol. 124, pp. 45-49. 2000.

[30] Akaike, Hirotugu. "A new look at the statistical model identification." *IEEE Transactions on Automatic Control*, vol. 19, pp. 716-723. 1974.

[31] "Apache PySpark Pandas". https://spark.apache.org/docs/latest/api/python//reference/pyspark.pandas/ Fecha de consulta: 24/08/2023.

[32] "Kedro". https://github.com/kedro-org/kedro. Fecha de consulta: 25/08/2023.

[33] J. Schwarmann. (2022) "The importance of layered thinking in data engineering". https://towardsdatascience.com/the-importance-of-layered-thinking-in-data-engineering-a09f685edc71. Fecha de consulta: 25/08/2023.

A. Anexo I: Código de ejemplo para la obtención del consumo del clúster

Código de ejemplo para el procesamiento de un *batch* del consumo de nodos. Se ha ejecutado en un Jupyter Notebook integrado con el clúster de Spark, por lo que no es necesario declarar manualmente la *SparkSession*. El código fuente del proyecto puede ser consultado en el siguiente repositorio: https://github.com/lmc00/TS-Forecasting/tree/master.

```
## Imports
from pyspark.sql import SparkSession
from pyspark import SparkContext
import os
import copy
import time
import statsmodels
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Spark dependencies
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark import StorageLevel, SparkConf
from pyspark.ml.feature import StandardScaler, VectorAssembler, PCA
from pyspark.mllib.linalg import SparseVector, DenseVector, VectorUDT
from pyspark.ml.classification import LogisticRegression

from pyspark.sql import functions as F
from pyspark.sql.window import Window
from datetime import datetime, timedelta
from pyspark.sql.types import TimestampType
from pyspark.sql.functions import col, expr, udf, sequence

# Other configs
pd.options.display.float_format = "{:.2f}".format

# Useful directory variables
src_path = os.getcwd()
root_path = os.path.dirname(src_path)
data_path = root_path + "/datasets"
visualization_path = root_path + "/data_visualization"

# Start counting time
start_t = time.time()
# Reading the original file
df = spark.read.parquet(
    "output_final.parquet"
)  # Functional programming. Reading the raw data file with the Structured API
# df.printSchema()
df.createOrReplaceTempView("df")

```

```
# Generating a dataFrame with the reference timestamp
dates = pd.date_range(
    start=datetime(2018, 1, 1, 0, 0, 0),
    end=datetime(2021, 6, 30, 0, 0, 0),
    freq="30min",
)
datetimes = [date.to_pydatetime() for date in dates]
time_df = (
    spark.createDataFrame(datetimes, TimestampType())
    .withColumnRenamed("value", "time")
    .sort(F.asc("time"))
)

# Obtaning each consumption node in a list
node_list = (
    spark.sql("SELECT node from df").rdd.flatMap(lambda x: x).collect()
)


#Obtaining each consumption node:
consumption_df = spark.createDataFrame(time_df.rdd, time_df.schema)
consumption_df = consumption_df.withColumn("total_average_power_consumption_W", lit(0))
for node in node_list[270:-3]:  # All the consumption related cluster nodes
    sql_query_node_consumption = """
                    SELECT
                        EXPLODE(power) as (time, node_{}_power_consumption)
                    FROM df
                    WHERE
                        node LIKE "{}"
                """.format(
        node, node
    )
    node_consumption = spark.sql(sql_query_node_consumption)
    node_consumption = node_consumption.withColumn(
        "time", F.to_timestamp(node_consumption.time, "yyyy-MM-dd HH:MM:SS")
    )
    node_consumption = node_consumption.groupBy(
        "time", F.window("time", "30 minutes")
    ).agg(
        avg("node_{}_power_consumption".format(node)).alias(
            "node_{}_power_consumption".format(node)
        ),
    )
    node_consumption = node_consumption.select(
        "time", "window.*", "node_{}_power_consumption".format(node)
    ).sort(F.asc("time"))
    node_consumption = node_consumption.select(
        col("end").alias("time"), col("node_{}_power_consumption".format(node))
    )
    node_consumption = node_consumption.groupBy("time").agg(
        avg("node_{}_power_consumption".format(node)).alias(
            "node_{}_average_power_consumption".format(node)
        )
```

```
    )
    node_consumption = node_consumption.select(
        "time", "node_{}_average_power_consumption".format(node)
    ).sort(F.asc("time"))
    consumption_df = consumption_df.join(node_consumption, ["time"], how="left").sort(
        F.asc("time")
    )
    consumption_df = consumption_df.fillna(0, subset=["node_{}_average_power_consumption".fo
    consumption_df = consumption_df.withColumn("total_average_power_consumption_W", col("tot
    consumption_df = consumption_df.drop("node_{}_average_power_consumption".format(node))
consumption_df.cache()

consumption_df.write.parquet("consumption_total_average_30min_W_271_to_288")
end_t = time.time()
print("Time in seconds " + str(end_t - start_t))
```

B. Anexo II: Código de ejemplo para la selección de la temperatura correcta

```
def get_active_chiller(row, threshold: int = 10) -> int:
    """
    Designed for being used as a part of an apply over a Pandas DataFrame

    inputs:
        row (pd.Series): row in the current iteration step of the applya
        threshold (int): electric consumption thrshold in kW
    outputs:
        (int): 0 if any, 1 if chiller 1, 2 if chiller 2, 3 if both at the same time
    """
    if (
        row.chiller_1_average_power_consumption > threshold
        and row.chiller_2_average_power_consumption > threshold
    ):
        return 3
    elif row.chiller_1_average_power_consumption > threshold:
        return 1
    elif row.chiller_2_average_power_consumption > threshold:
        return 2
    else:
        return 0


chiller_1_pandas = pd.read_parquet(
    data_path + "/02_intermediate/" + "chiller_1_consumption_total_average_30min_kW"
)
chiller_2_pandas = pd.read_parquet(
    data_path + "/02_intermediate/" + "chiller_2_consumption_total_average_30min_kW"
)
chiller_1and2_consumption_pandas = pd.merge(
    chiller_1_pandas,
```

```
    chiller_2_pandas,
    on="time",
)
chiller_1and2_consumption_pandas[
    "active_chiller"
] = chiller_1and2_consumption_pandas.apply(get_active_chiller, axis=1)
chiller_1and2_consumption_pandas[["time", "active_chiller"]].to_parquet(
    data_path + "/03_primary/" + "active_chiller"
)

def get_temperatures(row):
    """
    Designed for being used as a part of an apply over a Pandas DataFrame

    inputs:
        row (pd.Series): row in the current iteration step of the apply
    outputs:
        (list): list with the assigned values for the 4 temperature series depending on whic
    """

    if row.active_chiller == 1:
        return [
            row.chiller_1_Temperature_Ambient_Degrees,
            row.chiller_1_Temperature_Evaporator_Degrees,
            row.chiller_1_Temperature_In_Degrees,
            row.chiller_1_Temperature_Out_Degrees,
        ]
    if row.active_chiller == 2:
        return [
            row.chiller_2_Temperature_Ambient_Degrees,
            row.chiller_2_Temperature_Evaporator_Degrees,
            row.chiller_2_Temperature_In_Degrees,
            row.chiller_2_Temperature_Out_Degrees,
        ]

    if row.active_chiller == 0 or row.active_chiller == 3:
        return [
            row.chiller_1_Temperature_Ambient_Degrees,
            min(
                row.chiller_1_Temperature_Evaporator_Degrees,
                row.chiller_2_Temperature_Evaporator_Degrees,
            ),
            row.chiller_1_Temperature_In_Degrees,
            min(
                row.chiller_1_Temperature_Out_Degrees,
                row.chiller_2_Temperature_Out_Degrees,
            ),
        ]
    else:
        raise ValueError(
            "There are only 4 allowed active chiller values = {0,1,2,3}. Obtained {}".format
                str(row.active_chiller)
            )
```

```
        )


temperature_all = pd.read_parquet(
    data_path + "/03_primary/" + "chiller_1and2_temperatureAll_average_30min_P"
)
temperature_all = pd.merge(active_chiller, temperature_all, on="time")

temperature_all[
    [
        "Temperature_Ambient_Degrees",
        "Temperature_Evaporator_Degrees",
        "Temperature_In_Degrees",
        "Temperature_Out_Degrees",
    ]
] = temperature_all.apply(get_temperatures, axis=1, result_type="expand")
temperature_all = temperature_all[
    [
        "time",
        "Temperature_Ambient_Degrees",
        "Temperature_Evaporator_Degrees",
        "Temperature_In_Degrees",
        "Temperature_Out_Degrees",
    ]
].copy()
temperature_all.drop_duplicates(inplace=True)
setpoint = temperature_all.Temperature_Out_Degrees.mean()
temperature_all["Temperature_Ambient_Minus_Setpoint_Degrees"] = (
    temperature_all.Temperature_Ambient_Degrees - setpoint
)
temperature_all.to_parquet(data_path + "/04_feature/" + "average_temperature_Degrees")
```

C. Anexo III: Representación de series temporales exógenas

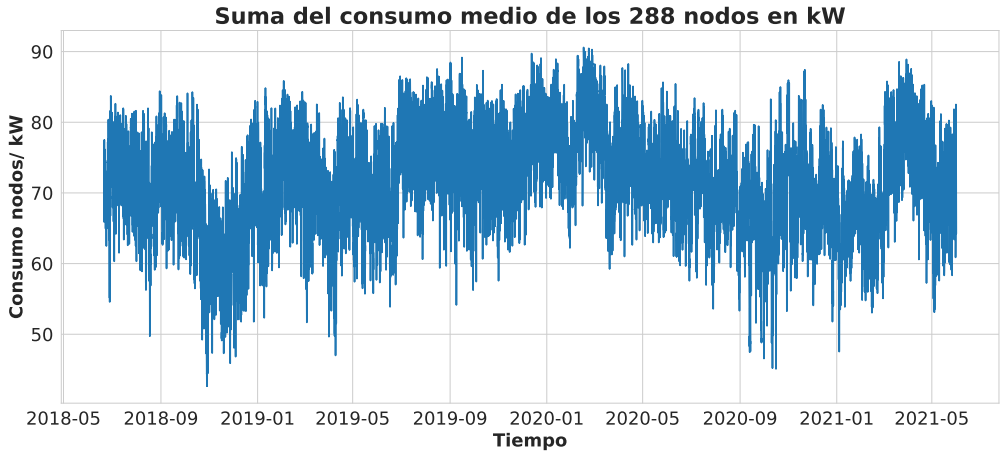

Figura 25: Serie temporal del consumo de total de los nodos P_{nodes} en kW.

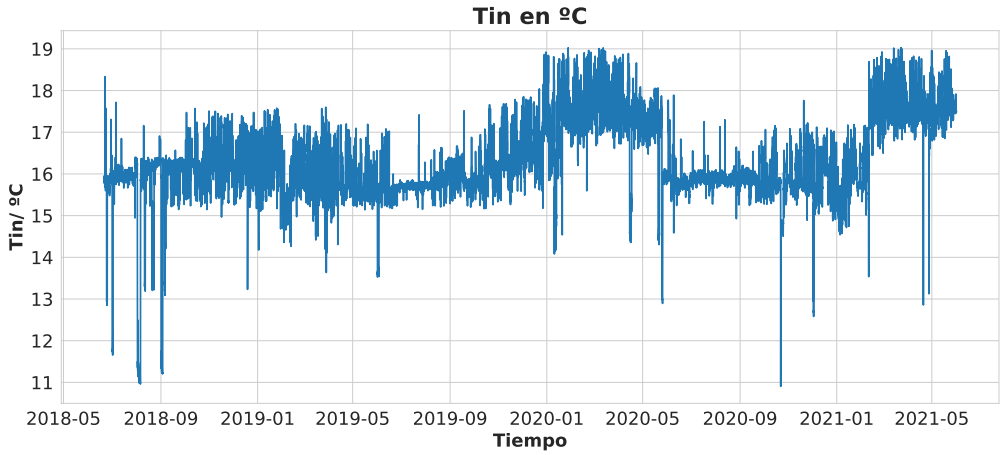

Figura 26: Serie temporal de la temperatura T_{in} en ºC.

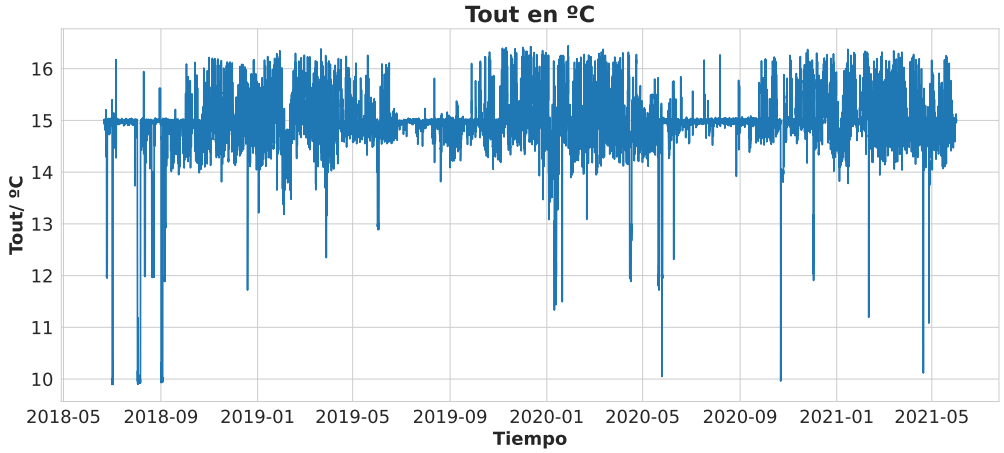

Figura 27: Serie temporal de la temperatura T_{out} en ºC.

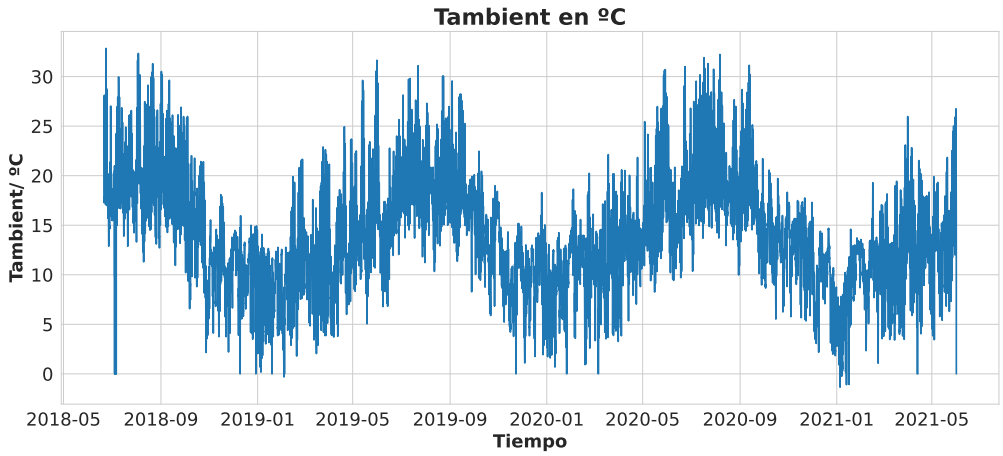

Figura 28: Serie temporal de la temperatura $T_{ambient}$ en ºC.

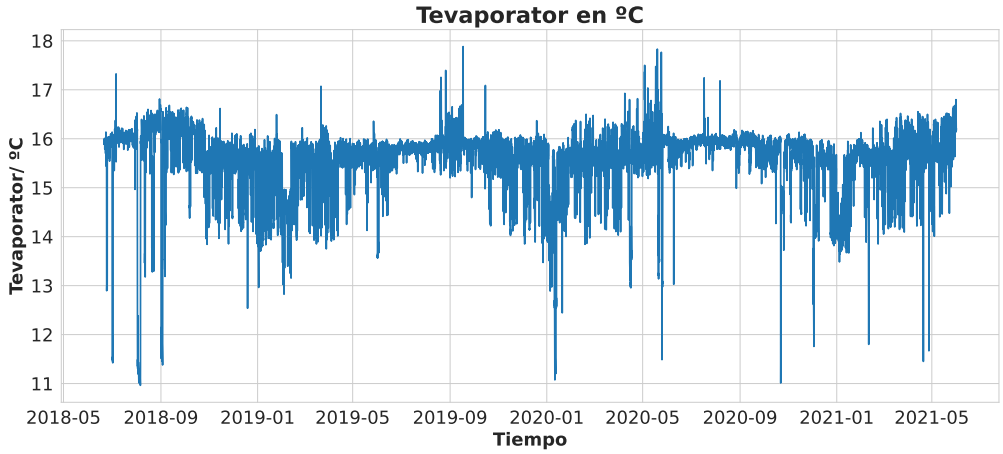

Figura 29: Serie temporal de la temperatura $T_{evaporator}$ en ºC.