UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Nuevas Formas de Movilidad y su Implantación en el Proyecto de Estacionamiento Inteligente de Valencia

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Gomariz Pérez, Álvaro

Tutor/a: Esteban González, Héctor

Cotutor/a externo: GOMEZ SACRISTAN, ANGEL AURELIANO

CURSO ACADÉMICO: 2022/2023

Resumen

En este proyecto se quiere desarrollar un programa de Python que tiene como finalidad realizar predicciones futuras sobre la cantidad de vehículos que circularán por cada tramo de calle de la ciudad de Valencia, a través del uso de este mismo dato en momentos anteriores. Estas predicciones podrán realizarse para diferentes momentos en el futuro. Para llevar a cabo este programa de predicción, se ha optado por utilizar varios métodos conocidos como Modelo Autorregresivo Integrado de Media Móvil (ARIMA) o las redes neuronales. Los resultados obtenidos a partir de estas predicciones tienen diversos propósitos. Por un lado, se busca proporcionar información visual y gráfica a los ciudadanos, por otro lado, estos resultados también serán utilizados por los gestores del tráfico, quienes podrán emplearlos para mejorar la planificación y la gestión del tráfico

Resum

En aquest projecte es vol desenvolupar un programa de Python que té com a finalitat realitzar prediccions futures sobre la quantitat de vehicles que circularan per cada tram de carrer de la ciutat de València, a través de l'ús d'aquesta mateixa dada en moments anteriors. Aquestes prediccions podran realitzar-se per a diferents moments en el futur. Per a dur a terme aquest programa de predicció, s'ha optat per utilitzar diversos mètodes coneguts com a Model Autorregresivo Integrat de Mitjana Mòbil (ARIMA) o les xarxes neuronals. Els resultats obtinguts a partir d'aquestes prediccions tenen diversos propòsits. D'una banda, es busca proporcionar informació visual i gràfica als ciutadans, d'altra banda, aquests resultats també seran utilitzats pels gestors del trànsit, els qui podran emprar-los per a millorar la planificació i la gestió del trànsit.

Abstract

In this project we want to develop a Python program that aims to make future predictions about the number of vehicles that will circulate on each stretch of street in the city of Valencia, through the use of this same data at previous times. These predictions can be made for different times in the future. To carry out this prediction programme, we have chosen to use various methods known as the Autoregressive Integrated Moving Average Model (ARIMA) or neural networks. The results obtained from these predictions serve several purposes. On the one hand, the aim is to provide visual and graphical information to citizens, on the other hand, these results will also be used by traffic managers, who will be able to use them to improve traffic planning and management.

Índice

Índice de figuras

Capítulo 1. Introducción

1.1 Resumen de proyecto

El Ayuntamiento de Valencia, a través del Servicio de Ciudad Inteligente, está llevando a cabo una serie de proyectos para mejorar la calidad de vida, la competitividad, la eficiencia del funcionamiento y los servicios urbanos.

El tráfico de vehículos en el contexto de las ciudades contemporáneas se ha convertido en un problema creciente que afecta la calidad de vida de los residentes. En Valencia, la congestión del tráfico se ha convertido en un problema persistente que necesita una solución viable como resultado del rápido crecimiento de la población y el aumento del número de vehículos en las calles. Para garantizar un flujo de vehículos efectivo, acortar los tiempos de viaje y mejorar la seguridad en las calles, la gestión del tráfico es crucial.

En las siguientes figuras se pueden observar la cantidad de vehículos en España y en Valencia. En ambas se puede comprobar cómo existe un aumento en la cantidad de los distintos tipos de vehículos a lo largo de los años. Además, se puede comprobar que la mayoría de estos vehículos son turismos, representan una gran parte del tráfico en València y su correcta gestión puede contribuir significativamente a la reducción de la congestión.

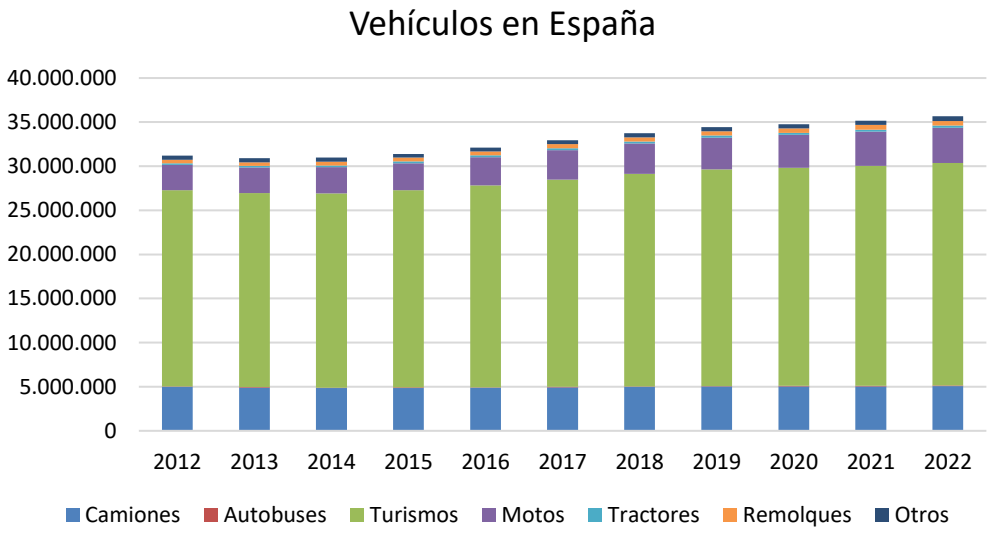

Fig 1. Parque de vehículos de España.

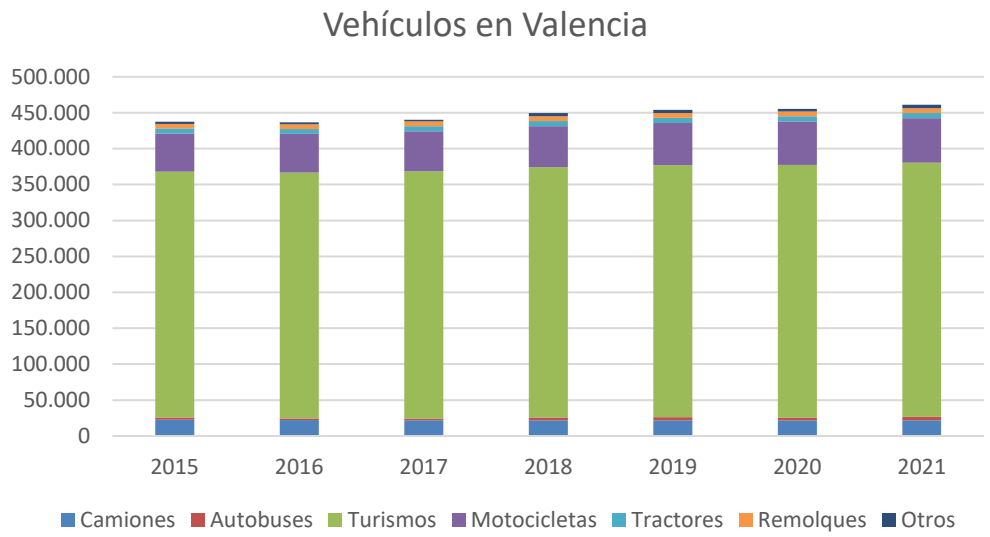

Fig 2. Parque de vehículos de Valencia.

1.2 Objetivos

El objetivo principal de este proyecto es el desarrollo de diferentes métodos predictivos para la gestión del tráfico en València. La recopilación y el análisis de datos en tiempo real, de los datos sobre la densidad del tráfico, serán la base de este enfoque. Se prevé que este enfoque sea capaz de pronosticar con precisión el flujo de tráfico utilizando algoritmos de aprendizaje automático y técnicas de modelado predictivo.

1.3 Ventajas

Hay muchos beneficios al usar un sistema de gestión de tráfico predictivo en Valencia, incluyendo:

- Los tiempos de viaje más cortos y una mayor eficiencia de movilidad resultarán de la capacidad de las autoridades para predecir el flujo de tráfico y tomar medidas proactivas para reducir la congestión y optimizar el flujo de vehículos.
- Mejora de la seguridad vial, se pueden tomar acciones preventivas para reducir los riesgos y aumentar la seguridad vial al prever posibles problemas en las carreteras, como accidentes o condiciones climáticas desfavorables.
- La planificación de rutas será más eficaz porque los conductores tendrán acceso a datos en tiempo real sobre las futuras condiciones del tráfico. Esto les ayudará a planificar sus viajes de manera más inteligente y evitar el tráfico innecesario.
- Reducción de emisiones contaminantes, al mejorar el flujo vehicular, se disminuirán los tiempos de espera y la congestión, lo que ayudará a reducir las emisiones de gases nocivos y mejorará la calidad del aire de la ciudad.

Capítulo 2. Componentes del proyecto

2.1 Datos abiertos

El Proyecto Datos Abiertos es una iniciativa llevada a cabo por el Ayuntamiento de València que tiene como principal objetivo la transparencia y el acceso libre a los datos que producen y recogen las distintas entidades públicas de la ciudad. El gobierno local está haciendo un esfuerzo para promover el uso de datos abiertos entre los ciudadanos para fomentar la colaboración, la participación y la innovación.

Datos abiertos es un término utilizado para describir conjuntos de información que son fácilmente accesibles y reutilizables. En estos datos se puede incluir información sobre una variedad de temas, incluidos el transporte, el medio ambiente, la educación, la cultura y la economía, entre otros. Al hacer que estos datos sean accesibles al público en general, se crea la oportunidad para que los ciudadanos, investigadores, desarrolladores y empresarios los usen de manera innovadora y agreguen valor en varios campos.

Los objetivos que se pretenden impulsar a través del proyecto de Datos Abiertos son los siguientes:

- Transparencia: Dado que los datos son confiables y provienen de una fuente legítima, es una excelente manera de explicar cómo se lleva a cabo la gestión pública, cómo se rinden cuentas y cómo la gestión está sujeta a supervisión externa. El objetivo de todo ello es aumentar la confianza ciudadana en la gestión del sector público.
- Reutilización de los datos públicos: Cualquier organización genera una gran cantidad de información que puede ser útil para otras divisiones u organizaciones. El concepto de "Datos Abiertos" se define como una herramienta para la publicación y reutilización de información, mediante la cual las administraciones ponen su trabajo a disposición del público para su reutilización.
- Promover la innovación: La disponibilidad de nuevas fuentes de datos abre la puerta a la innovación, lo que permite fortalecer las líneas de negocio, desarrollar nuevos servicios, ver los problemas desde diferentes ángulos y trabajar juntos para superar los obstáculos, etc.

Se ha utilizado la plataforma de datos abiertos para la recolección de los datos requeridos durante el desarrollo de este proyecto. El conjunto de datos "Intensidad del punto de medición del tráfico" es el que fue elegido después de una revisión exhaustiva de la enorme variedad de *"datasets"* accesibles en esta plataforma.

Este conjunto de datos recopila la información de sensores colocados en toda la ciudad, los cuáles se ocupan de obtener y proporcionar la cantidad de vehículos que circulan por hora en cada sección. Este conjunto de datos ofrece las coordenadas geográficas de cada tramo, la fecha y hora en que se registró la información, junto con información sobre la intensidad del tráfico. En la plataforma, además de poder conseguir los datos mencionados, también se ofrece un mapa en el podemos observar dónde están colocado cada sensor con sus respectivos datos, tal y como se muestran en la Figura 3.

Fig 3. Mapa de unos sensores de "Intensidad del punto de medición del tráfico"

Dado que en la plataforma de acceso público solo se puede encontrar la medida más reciente, conseguir los datos proporcionados por el Ayuntamiento a través de su base de datos ha sido fundamental para este proyecto. Gracias a esta colaboración con el Ayuntamiento y al acceso a su base de datos, se ha podido recopilar un historial de datos de intensidad de tráfico que comprende desde el mes de abril hasta julio, crucial para la implementación de un método predictivo eficiente.

Se han obtenido un total de 15.000.000 registros durante el proceso de recopilación de datos de todos los sensores utilizados. Esto significa que el promedio aproximado es de 39.000 registros por sensor.

Esta cantidad considerable de datos recopilados proporciona una base sólida para el análisis y la aplicación del método predictivo en el proyecto de gestión del tráfico. Cuantos más datos estén disponibles, más fácil será detectar tendencias, patrones y anomalías en el flujo de vehículos a lo largo del tiempo. La amplitud y diversidad de estos datos hacen posible crear una imagen más completa de la cantidad de tráfico presente en varias partes de la ciudad, y lo largo del día.

2.2 Método de predicción temporal de Suavizado Exponencial Simple

Para pronosticar valores futuros en una serie de tiempo, se puede usar el método de suavizado exponencial simple (SES). Este enfoque es apropiado para series de tiempo sin tendencia o estacionalidad, es decir, para series que exhiben un comportamiento estable en el tiempo sin patrones evidentes de crecimiento o declive.

Para dar más peso a las observaciones más recientes y menos peso a las observaciones más antiguas, el suavizado exponencial simple funciona otorgando a las observaciones pasadas pesos exponenciales decrecientes. La observación actual (y(t)) y el pronóstico en el tiempo t (ŷ(t)) se combinan juntos para determinar el pronóstico en el tiempo t+1.

Cuando una serie se obtiene como un promedio ponderado de varios valores, se denomina suavizada porque las oscilaciones en la serie son más suavizadas o alisadas. El calificador exponencial, por otro lado, se refiere al hecho de que el peso de las observaciones cae exponencialmente a medida que nos alejamos del momento actual t. Esto indica que existe una incidencia muy baja de observaciones a distancia. Para distinguirlo de otros casos en los que una variable se somete a una operación de suavizado doble, el calificador simple se agrega al final.

La siguiente fórmula se utiliza para determinar el pronóstico en el momento t+1 (ŷ(t+1))

$$\widehat{y(t+1)} = \alpha * y(t) + (1-\alpha) * \hat{y}(t) \qquad \textbf{(2.1)}$$

Donde:

- $\hat{y}(t+1)$ es el pronóstico en el tiempo t+1.
- $y(t)$ es la observación actual en el tiempo t.
- $\hat{y}(t)$ es el pronóstico en el tiempo t.
- α ($0 < \alpha < 1$) es el parámetro de suavizado exponencial, también conocido como el factor de suavizado. Identifica los pesos asignados a la predicción anterior y a la observación actual. Un valor más alto de da más peso a la observación actual, lo que hace que el pronóstico sea más susceptible a cambios recientes. El pronóstico se vuelve más suave y menos susceptible a los cambios recientes cuando es más pequeño porque le da más peso a la predicción anterior.

Determinar el valor óptimo de α, que minimiza el error de pronóstico en la serie de tiempo histórica, es parte del proceso de ajuste del método de Suavizado Exponencial Simple. Esto se puede lograr mediante el uso de métodos de optimización o pruebas iterativas para determinar el valor más adecuado para los datos.

Para evaluar varios valores α y elegir el que produzca el menor error de pronóstico en los datos no utilizados en el ajuste, generalmente se utiliza la validación cruzada, también conocida como método de entrenamiento y prueba.

2.3 Método de predicción temporal ARIMA

El método estadístico conocido como "*Autoregressive Integrated Moving Average*" o método ARIMA se utiliza para modelar y pronosticar series de tiempo.

Se utilizan para el análisis de datos y la predicción de patrones en muchos campos diferentes donde los datos se presentan como una secuencia de tiempo, incluidas las ventas, la meteorología, la economía y las finanzas.

Para la realización de un modelo ARIMA se suele realizar los siguientes pasos:

- Análisis de series temporales: para comprender la tendencia, la estacionalidad y otros patrones presentes en los datos, primero se realiza un análisis exploratorio.
- Se confirma si la serie temporal es estacionaria. Si no, se utiliza la diferenciación (integración) hasta que la serie se vuelve estacionaria.
- Encontrar parámetros necesarios: Los métodos como los gráficos ACF (función de autocorrelación) y PACF (función de autocorrelación parcial) se utilizan para encontrar los valores de p, d y q que mejor se ajustan a los datos.
- Ajuste del modelo: los valores elegidos de p, d y q se utilizan para ajustar el modelo ARIMA.
- Se utilizan métodos de validación cruzada y otras métricas de rendimiento para evaluar la calidad del modelo.
- Es posible hacer predicciones basadas en series temporales históricas después de que el modelo haya sido validado.

Es crucial recordar que el método ARIMA asume que la serie de tiempo es lineal y los errores de pronóstico son ruido blanco, es decir, tienen una media cero y una varianza constante. Es posible que se requieran otros modelos más complejos si la serie temporal muestra patrones más complicados o no lineales.

2.3.1 Autoregresión (AR)

Una idea crucial en el análisis de series temporales es la autoregresión, el cual es un componente del método ARIMA, así como de otros modelos de series temporales.

La autorregresión es la dependencia de la observación actual de una serie temporal con respecto a sus valores anteriores. Es decir, en un modelo AR, la variable que intentamos predecir, entonces, está relacionada con sus propias observaciones históricas.

Al examinar su estructura matemática, el modelo de autoregresión se puede entender más fácilmente. Una combinación lineal de los p valores anteriores de la serie temporal (y(t-1), y(t-2), …, y(t-p)) por sus coeficientes correspondientes (φ1, φ2, ..., φp) es cómo un modelo AR(p) expresa la observación actual (y(t)).

Es crucial tener en cuenta que la serie de tiempo debe ser estacionaria o haber sido estacionaria por diferenciación para poder aplicar un modelo AR. Dado que la estacionariedad garantiza que la relación entre las observaciones anteriores y las más recientes permanezca constante en el tiempo, es una condición necesaria para que el modelo AR funcione correctamente.

Los métodos de ajuste, como el método de los mínimos cuadrados, son utilizados para estimar los coeficientes del modelo AR. El objetivo es identificar los valores de los coeficientes que minimiza la discrepancia entre las observaciones reales y las predicciones del modelo.

2.3.2 Media Móvil (MA)

Otro elemento esencial del método ARIMA y una técnica de análisis de series temporales ampliamente utilizada es la (MA) Media Móvil también conocida como Promedio Móvil.

La Media Móvil se concentra en la relación entre una observación actual y los errores de pronóstico anteriores a diferencia del componente de AR, que modela la relación entre una observación y sus valores pasados.

Es crucial comprender que en un modelo MA, se tienen en cuenta los errores de pronóstico pasados en lugar de los valores históricos de la serie temporal en su conjunto.

La discrepancia entre las observaciones actuales y las predicciones del modelo histórico se conoce como error de pronóstico. El propósito del promedio móvil es identificar patrones en el error de pronóstico y hacer uso de este conocimiento para mejorar los pronósticos futuros. Al igual que lo es para el modelo AR, la estacionariedad de la serie temporal es necesaria para aplicar el modelo MA.

El valor de q, que denota cuántos errores de pronóstico pasados deben tenerse en cuenta para hacer la predicción en el tiempo t, sirve como representación del orden del modelo MA. Si q es igual a 1, el modelo es MA(1) y solo se tiene en cuenta el error de pronóstico más reciente. Mayor q tiene en cuenta más errores de pronóstico históricos.

Utilizando técnicas de ajuste como el método de mínimos cuadrados o el método de máxima verosimilitud, se estiman los coeficientes del modelo MA. El objetivo es encontrar los valores de los coeficientes que reducen la discrepancia entre las observaciones reales y las predicciones del modelo.

El modelo ARMA (del inglés, *Autoregressive Moving Average)* se crea fusionando los componentes de media móvil (MA) y autorregresivo (AR). El modelo ARMA puede tener en cuenta la dependencia temporal de la serie temporal, así como el impacto de los errores de pronóstico anteriores en las predicciones.

2.3.3 Integración(I)

Uno de los elementos clave del método ARIMA, es la integración, la cual es crucial para el análisis y el modelado de series temporales. Cuando una serie temporal está integrada, significa que se ha diferenciado para volverse estacionaria.

Si la media y la varianza de una serie de tiempo se mantienen constantes en el tiempo, se dice que es estacionaria. Muchos modelos estadísticos, incluido ARIMA, dependen del supuesto de estacionariedad porque facilita el análisis y permite predicciones más precisas.

Una serie de tiempo no estacionaria no exhibe tendencia y/o estacionalidad, sus características estadísticas evolucionan con el tiempo. Por ejemplo, la serie puede exhibir una tendencia constante al alza o a la baja a lo largo del tiempo (tendencia) o patrones cíclicos recurrentes durante un periodo período prolongado (estacionalidad). Dado que los patrones en una serie temporal no estacionaria pueden modificarse con el tiempo, la precisión del modelado y la predicción son más desafiantes.

2.3.4 Parámetros p, d y q

Por lo tanto, el modelo ARIMA se compone de tres parámetros: p, d y q.

- p: Orden del componente de autoregresión (AR).
- d: Número de veces que se diferencia la serie de tiempo para hacerla estacionaria.
- q: Orden del componente de media móvil (MA).

2.3.4.1 Autocorrelación (ACF)

Una herramienta crucial en el análisis de series de tiempo, la función de autocorrelación (ACF), nos permite examinar la relación entre una observación y sus valores retrasados, también conocidos como rezagos, en varios puntos en el tiempo.

El término "autocorrelación" describe la relación entre las observaciones realizadas en varios puntos en el tiempo, o la correlación que tiene una serie de tiempo consigo misma.

Primero se calcula la correlación entre una observación en un momento dado (t) y sus valores retrasados en diferentes momentos (τ) para determinar el ACF.

El retraso se muestra en el eje horizontal de un gráfico de barras o de líneas, y los valores de correlación se muestran en el eje vertical. Los valores de ACF varían de -1 a 1, donde 1 indica una correlación positiva perfecta, -1 indica una correlación inversa o negativa perfecta, y 0 indica que no hay correlación.

El orden del componente de media móvil MA en el modelo ARIMA se puede determinar analizando el ACF en un gráfico. ACF puede identificar patrones en la correlación entre la serie temporal y sus retrasos que están relacionados con MA, lo que representa la dependencia de una observación de errores de pronóstico previos en ARIMA.

Encontrar el valor ideal de "q" en un modelo ARIMA (p, d, q) es importante para determinar cuánto error de pronóstico histórico debe incluirse en el modelo MA.

Si hay una correlación significativa en el primer retraso (=1) de una gráfica ACF pero un patrón poco o nada significativo en los retrasos posteriores, esto puede sugerir que un modelo MA(1) sería apropiado. Un modelo MA de orden superior como MA(2) o MA(3), podría ser más apropiado si existen correlaciones significativas entre múltiples rezagos.

La correlación de los retrasos se "corta" después del verdadero orden MA, que es una característica distintiva importante del ACF en un modelo MA. En otras palabras, las correlaciones se vuelven insignificantes y eventualmente se aproximan a cero en el modelo MA una vez que se incorporan suficientes rezagos. En el gráfico ACF esto aparecería como una "cola" que se extiende en la dirección de cero después de la orden MA.

También puede estar presente un patrón estacional en los datos si el ACF exhibe correlaciones significativas en múltiples retrasos de un valor particular.

2.3.4.2 Autocorrelación Parcial (PACF)

La función de autocorrelación (ACF) es una herramienta crucial en el análisis de serie de tiempo, pero la PACF (Función de Autocorrelación Parcial) también es crucial.

El PACF, como el ACF, mide la correlación entre una observación y sus valores de retraso, pero se concentra en eliminar las correlaciones de los retrasos intermedios, dando una imagen más clara y directa de la dependencia del tiempo en la serie.

Para comprender cómo funciona el PACF, primero debemos comprender cómo se crea. La correlación entre la observación actual (y(t)) y sus valores de rezago (y(t-τ)), luego de las contribuciones de los rezagos intermedios (y(t-1), y(t- 2), …, y(t-τ+1)), se utiliza para calcular el PACF en un rezago dado (τ).

En otras palabras, el PACF se concentra en la observación en el tiempo t y la observación en la correlación directa del tiempo t-τ, sin considerar ninguna otra observación intermedia.

El eje horizontal de un gráfico PACF es el retraso y el eje vertical son los valores de correlación, muy parecido a un gráfico ACF. Los valores PACF varían de -1 a 1, donde -1 indica una correlación inversa perfecta, 1 indica una correlación positiva perfecta y 0 indica que no hay correlación, al igual que el ACF.

El orden del componente autorregresivo (AR) en el modelo ARIMA se puede determinar usando el PACF en particular.

En una gráfica PACF, si las correlaciones son significativas en el primer retraso (=1) pero disminuyen rápidamente y se vuelven insignificantes en los retrasos posteriores, esto podría sugerir que un modelo AR(1) sería apropiado. Considerar un modelo AR(p), donde "p" es el orden correcto del componente AR, si existen correlaciones significativas entre múltiples retrasos, pero se detienen después de un cierto número de retrasos.

La presencia de un patrón estacional en los datos también puede indicarse mediante la gráfica PACF, que exhibe correlaciones significativas en retrasos que son múltiplos de un valor particular.

2.3.4.3 Diferenciación

La diferenciación es un método para convertir una serie de tiempo no estacionaria en una serie de tiempo estacionaria al eliminar la tendencia y la estacionalidad. La diferenciación es el proceso de calcular las diferencias entre observaciones posteriores en una serie de tiempo. La serie de tiempo diferenciada se conoce como la primera diferencia y se denota como "y(t) - y(t-1)". Para decirlo de manera más amplia, hasta que la serie se vuelva estacionaria, se pueden usar segundas, terceras o más diferencias.

La cantidad de veces que se diferenciaron las series de tiempo para lograr la estacionariedad se indica mediante el orden de integración, que se representa con la letra "d" en el método ARIMA. La serie ya es estacionaria si d = 0, lo que significa que no se requiere diferenciación. Se ha aplicado una primera diferencia en el caso de que d = 1, y así sucesivamente.

Durante el proceso de diferenciación, se tiene en cuenta el tiempo entre observaciones además de restar las observaciones sucesivas. Por lo general, es suficiente realizar una diferenciación de primer orden para lograr la estacionariedad cuando las observaciones se realizan a intervalos regulares (por ejemplo, datos mensuales, trimestrales o anuales). Sin embargo, podría ser necesario aplicar una diferenciación de orden superior si hay estacionalidad.

Es crucial recordar que la diferenciación puede afectar los modelos subsiguientes, aunque puede ayudar a que la serie temporal se vuelva estacionaria. Por ejemplo, una diferenciación de

primer orden implica que las observaciones sucesivas están correlacionadas, lo que se tiene en cuenta en los modelos ARIMA.

Un paso crucial en el análisis de series de tiempo es determinar el orden de integración d. Para determinar si se requiere diferenciación y en qué medida, se pueden usar pruebas estadísticas y herramientas gráficas como gráficos de series de tiempo y gráficos de autocorrelación.

Para determinar si la serie temporal es estacionaria debe cumplir que sea tanto estacionaria en varianza como en media:

- La estacionariedad de la varianza es el requisito de que la variabilidad de las observaciones en una serie de tiempo permanezca constante a lo largo de la serie. En otras palabras, a medida que pasa el tiempo, la amplitud de las fluctuaciones en la serie no debería cambiar significativamente.
- El requisito de que la media de las observaciones en una serie de tiempo permanezca constante a lo largo del tiempo se conoce como estacionariedad media. En consecuencia, no debería haber una tendencia perceptible al alza o a la baja en el nivel promedio de la serie a medida que pasa el tiempo.

2.4 Método de predicción temporal SARIMA

Una variación del modelo ARIMA (Promedio móvil integrado autorregresivo) utilizado para analizar y predecir series temporales con estacionalidad se llama SARIMA (Promedio móvil integrado autorregresivo estacional).

Los patrones como las repeticiones diarias, mensuales o estacionales se denominan estacionales. El modelo SARIMA fue creado para reconocer y modelar estos efectos estacionales porque estos patrones estacionales pueden tener una influencia significativa en el comportamiento de la serie temporal.

La notación SARIMA es representada como SARIMA(p, d, q)(P, D, Q, s), donde:

- P: Representa el orden del componente AR estacional que modela la dependencia lineal entre una observación y sus valores rezagados estacionales.
- D: Indica el número de diferencias estacionales necesarias para alcanzar la estacionariedad en media después de tomar las diferencias estacionales.
- Q: Es el orden del componente MA estacional que modela la dependencia lineal entre una observación y los errores residuales estacionales de observaciones pasadas.
- s: Representa la frecuencia de la estacionalidad, es decir, la longitud del ciclo estacional. Por ejemplo, para datos mensuales con estacionalidad anual, s sería igual a 12.

El modelo SARIMA puede manejar series de tiempo con estacionalidades múltiples, es decir, datos que exhiben más de una estacionalidad. En tales casos, el modelo SARIMA puede ajustarse adecuadamente para capturar varias estacionalidades, lo que lo hace especialmente útil para series con patrones estacionales complejos.

El modelo SARIMA es adecuado para realizar pronósticos a largo plazo en series de tiempo estacionales. Sin embargo, es importante tener en cuenta que a medida que aumenta el horizonte de pronóstico, la precisión de las predicciones puede disminuir debido a la propagación de errores a lo largo del tiempo.

La estimación de los parámetros del modelo SARIMA puede ser computacionalmente intensiva, especialmente para series de tiempo con un gran número de observaciones y órdenes altos. En tales casos, pueden requerirse algoritmos de estimación más avanzados y técnicas de optimización para obtener resultados eficientes y precisos.

2.5 Redes neuronales

Un modelo computacional llamado red neuronal, también conocido como perceptrón multicapa o artificial, se basa en la estructura y el funcionamiento del sistema nervioso humano. Está hecho para procesar datos en forma de vectores numéricos para aprender y resolver problemas complejos de aprendizaje automático. El reconocimiento de patrones, la clasificación de imágenes, el procesamiento del lenguaje natural, los juegos y muchos otros campos han tenido un gran éxito con el uso de redes neuronales.

Una red neuronal se compone de capas de unidades de procesamiento interconectadas llamadas neuronas o nodos. Hay un peso asignado a cada conexión entre neuronas que refleja cuán crítica es esa conexión para el funcionamiento de la red.

Una función de pérdida, que mide la discrepancia entre las predicciones del modelo y los resultados reales, se minimiza durante el entrenamiento ajustando los pesos de forma iterativa al valor mínimo de la función. Para determinar los pesos ideales que minimicen la pérdida, se utilizan algoritmos de optimización como el descenso de gradiente.

A continuación, describimos con más detalle los componentes clave de una red neuronal:

- Neurona: En una red neuronal, una neurona es el nodo de procesamiento fundamental. Cada neurona recibe una o más entradas (de datos de entrada o de otras neuronas), multiplica cada entrada por su peso correspondiente y luego realiza una operación de suma ponderada en las entradas.
- Capas: Una red neuronal está formada por capas de neuronas organizadas secuencialmente. Tanto la capa de entrada como la capa de salida producen las predicciones o los resultados del modelo. Una o más capas ocultas que están a cargo de descubrir patrones y características intrincados en los datos pueden existir entre la capa de entrada y la capa de salida.
- Capa de entrada: La entrada de datos está representada por la capa de entrada, la capa superior de la red. Cada neurona de la capa de entrada está asociada con una característica o atributo particular de los datos de entrada.
- Capas ocultas: Entre la capa de entrada y la capa de salida hay capas conocidas como capas ocultas. Están a cargo de descubrir las relaciones y características no lineales en los datos. Los hiperparámetros que se ajustan durante el diseño del modelo y el proceso de entrenamiento incluyen la cantidad de capas ocultas y la cantidad de neuronas en cada capa.
- Capa de salida: La capa final de la red, la capa de salida es donde se generan las predicciones o los resultados del modelo. La configuración de esta capa está determinada por el tipo de problema que se está resolviendo.
- Pesos y Conexiones: Cada neurona de la red tiene conexiones ponderadas con las neuronas de la capa superior y la capa inferior. Los pesos representan la importancia de cada conexión en el procesamiento de la información. Para reducir la pérdida y aumentar la precisión del modelo en los datos de entrenamiento, los pesos se cambian durante el entrenamiento.
- Función de Pérdida (*Loss Function*): Es posible cuantificar la diferencia entre las predicciones del modelo y los resultados reales utilizando la función de pérdida. Para que las predicciones del modelo se acerquen lo más posible a los valores reales, el objetivo del entrenamiento es minimizar la pérdida.
- Algoritmo de Optimización: Usando algoritmos de optimización como el descenso de gradiente, es posible cambiar los pesos de la red para reducir la función de pérdida. En un esfuerzo por converger a los pesos que producen las mejores predicciones, estos algoritmos cambian gradualmente los pesos en una dirección que reduce la pérdida.

- Entrenamiento y Aprendizaje: Cuando se entrena una red neuronal, los datos de entrenamiento se introducen en la red, se calculan las predicciones, se evalúan las pérdidas y se ajustan los pesos para optimizar el rendimiento. Aprender patrones y relaciones en los datos es el objetivo del entrenamiento, lo que permite que el modelo haga predicciones precisas basadas en datos nuevos.
- Validación y Evaluación: La evaluación del rendimiento de la red neuronal en los datos que no se utilizan durante el entrenamiento es crucial después de entrenar la red. Para lograr esto, se emplean métodos de validación cruzada o los datos se dividen en conjuntos de entrenamiento y prueba. Se puede obtener una estimación realista del rendimiento del modelo en circunstancias prácticas a partir de la evaluación de datos no observados.

2.5.1.1 Redes neuronales LSTM

Para resolver el problema del desvanecimiento y el estallido de gradiente, que puede ocurrir en las redes neuronales recurrentes (RNN) convencionales cuando se entrenan con secuencias largas, las redes LSTM (Long Short-Term Memory) son un tipo especial de RNN. Han demostrado tener mucho éxito en una variedad de tareas de procesamiento de datos secuenciales, incluido el análisis de series temporales, el reconocimiento de voz, el procesamiento del lenguaje natural y más.

La celda LSTM, que es la unidad fundamental de una capa LSTM, es la característica principal que distingue a las redes LSTM de las RNN convencionales. La capacidad de la celda LSTM para almacenar datos pertinentes de secuencias anteriores durante largos períodos de tiempo permite que la red reconozca la dependencia a largo plazo de los datos.

A continuación, describiremos los elementos esenciales de una célula LSTM:

Memoria a largo plazo: El elemento clave en una celda LSTM que permite que la red almacene datos de secuencias anteriores durante mucho tiempo es la memoria a largo plazo. La puerta de olvido y la puerta de entrada, que controlan qué información se debe retener y cuál se debe descartar, actualizan la memoria a largo plazo.

Puerta de entrada: La puerta de enlace decide qué nueva información debe almacenarse en la memoria a largo plazo. La puerta de enlace determina qué nueva información incluir en la memoria a largo plazo en cada paso de tiempo utilizando la entrada actual y los datos de la capa oculta anterior (estado oculto).

Puerta de olvido: La celda LSTM puede elegir qué datos de la memoria a largo plazo borrar usando la puerta de olvido. Esto es crucial para resolver el problema del desvanecimiento de gradiente que pueden experimentar los RNN tradicionales. Según la entrada actual y el estado oculto anterior, la puerta de olvido decide qué información en la memoria a largo plazo se retendrá y qué información se borrará.

Puerta de salida: La puerta de salida determina qué datos se transferirán desde la memoria a largo plazo a la capa de salida. La puerta de salida determina cómo calcular el estado oculto y qué datos se enviarán a la siguiente capa o salida de la red.

Capítulo 3. Herramientas y bibliotecas utilizadas

3.1 Python

Python es un lenguaje de programación flexible, potente y fácil de aprender que se utiliza en una amplia gama de aplicaciones y campos.

Es el favorito tanto de los desarrolladores novatos como de los experimentados debido a su sintaxis legible y su vibrante comunidad.

Debido a que se usa ampliamente en la industria, tiene excelentes bibliotecas de aprendizaje profundo y una comunidad activa, Python es una buena opción para trabajar con redes neuronales. El desarrollo de redes neuronales es más eficaz y eficiente gracias a la simplicidad y flexibilidad de Python.

3.2 Biblioteca *Pandas*

La biblioteca *Pandas* de *Python* es una herramienta crucial para manejar y analizar datos. Pandas ofrece estructuras y funciones de datos que permiten a los profesionales manejar datos de manera efectiva y eficiente, lo cual es fundamental en muchos campos e industrias diferentes.

Entre las principales características principales de la biblioteca Pandas encontramos:

- Integración de datos externos como CSV, Excel o bases de datos SQL.
- Estructuras de *DataFrames* y *Series* utilizadas para el análisis y manipulación de datos en Python
- Pandas facilita la manipulación e indexación de los datos para facilitar el uso de estos.

3.3 Biblioteca *NumPy*

Realizar cálculos numéricos y matemáticos rápidos y efectivos requiere el uso de la biblioteca esencial de Python conocida como *NumPy*. Para fines de análisis de datos, computación científica y programación general, ofrece una variedad de estructuras de datos, funciones y herramientas para trabajar con matrices y arreglos multidimensionales.

3.4 Biblioteca *StatsModels*

Para modelado y análisis estadístico, *StatsModels* es una biblioteca de Python que ofrece recursos para realizar análisis de regresión, exploración de datos estadísticos y ajuste de modelos estadísticos. En comparación con las bibliotecas de aprendizaje automático, *StatsModels* es particularmente útil para quienes desean realizar análisis estadísticos más convencionales y detallados.

3.5 Biblioteca *Matplotlib*

Para producir visualizaciones y gráficos excelentes, *Matplotlib* es una de las bibliotecas de Python más utilizadas. Ofrece herramientas adaptables y versátiles para crear una variedad de gráficos, desde diagramas lineales sencillos hasta visualizaciones 3D complejas. *Matplotlib* es crucial para expresar claramente los resultados del análisis de datos y para comprender patrones y tendencias en los datos.

3.6 Biblioteca *TensorFlow*

Se utiliza una biblioteca de código abierto desarrollada por Google llamada *TensorFlow* para crear y entrenar modelos de aprendizaje automático y redes neuronales.

Se lanzó en 2015 y desde entonces ha crecido hasta convertirse en una de las bibliotecas más utilizadas para proyectos que involucran aprendizaje automático e inteligencia artificial. *TensorFlow* destaca por su adaptabilidad y escalabilidad, permitiendo la construcción de una variedad de modelos, desde los más básicos hasta los más sofisticados.

3.7 Biblioteca Scikit-Learn

Una de las bibliotecas de aprendizaje automático más conocidas y utilizadas en Python se llama *Scikit-Learn*. Las funciones de evaluación de modelos, las herramientas de preprocesamiento de datos y los algoritmos de aprendizaje automático que proporciona son extremadamente diversos. *Scikit-Learn* permite a los desarrolladores y científicos de datos crear modelos de aprendizaje automático de manera efectiva incluso si no son expertos en el campo gracias a su énfasis en la simplicidad y la eficiencia. La biblioteca ofrece una base sólida para desarrollar modelos predictivos y de clasificación, lo que la convierte en una opción crucial para proyectos de análisis de datos y aprendizaje automático.

Capítulo 4. Desarrollo del proyecto y resultados

Se han realizado los códigos de los tres métodos que se han investigado para comprobar cuál es el modelo que menos error y por ello más se acerca a los datos reales.

En todos los códigos se ha realizado la predicción de unos pocos sensores y no de toda su totalidad, puesto que es una forma de comprobar los resultados de forma más rápida.

4.1 Tratamiento previo de los datos

En los datos obtenidos se encuentran tipos de datos que no nos interesan, datos vacíos debido a posibles errores, y posibles desfases temporales. Para ello necesitamos realizar un tratamiento de datos previo para conseguir un conjunto de datos limpio y coherente que pueda ser utilizado de manera efectiva en el análisis posterior.

Para ello lo primero que hacemos es escoger solamente aquellos datos que necesitamos, específicamente el nombre del sensor para identificar cada uno, la cantidad de tráfico que se ha captado por dicho sensor y la fecha y hora en la que se ha captado dicho valor.

```
# Cargar el archivo de Excel
data = pd.read_csv('Intensidad.csv')

nombres_columna = data.columns

# Obtener los valores de las tres columnas
it = data[nombres_columna[0]]
fecha_str = data[nombres_columna[1]]
nombre = data[nombres_columna[2]]
```

Fig 4. Código para obtener los datos útiles.

El siguiente paso es seleccionar un único sensor para realizar la prueba de los códigos de forma más rápida. En este caso seleccionaremos el primer sensor de la lista por el nombre “A1” para comprobar la eficiencia de los distintos modelos predictivos.

```
sensores_unicos = list(set(nombre))

for sensor in sensores_unicos:
    it_sensor = []
    fechas_sensor = []
    fechas=[]

    if sensor=="A1":
        # Filtrar los datos por nombre de sensor
        for i in range(len(nombre)):
            if nombre[i] == sensor:
                it_sensor.append(it[i])
                fechas.append(fecha_str[i])
    else:
        continue
```

Fig 5. Código para obtener un solo sensor.

Para ello a través de set seleccionamos la lista "nombre" en un conjunto. Un conjunto es una estructura que no permite elementos duplicados, por lo que elimina los elementos que se repitan (en este caso los nombres duplicados). Una vez eliminados los elementos duplicados, lo volvemos a convertir en una lista. Tras haber realizado la lista podemos escoger específicamente el sensor que queremos utilizar, mediante el uso de bucles, para crear dos vectores con los datos del sensor.

De esta manera, si necesitamos o queremos comprobar un método de predicción con cualquier otro sensor, podemos realizarlo simplemente cambiando el nombre que hemos colado en el bucle *for*.

Ahora vamos a establecer un intervalo estable entre los datos, puesto que el tiempo de los diferentes datos puede variar por motivos como retrasos en la publicación de los tiempos. Entre cada dato hay un tiempo medio de 3 minutos. Para disminuir el tiempo de procesamiento de los datos en los diferentes métodos, vamos a establecer intervalos de 10 minutos.

```
for fechas in fechas:
    fecha = parse(fechas)
    fechas_sensor.append(fecha)

df = pd.DataFrame({'Fecha': fechas_sensor, 'IT': it_sensor})

df.set_index('Fecha', inplace=True)

df_resample = df.resample('10T').mean()
```

Fig 6. Código para establecer un intervalo temporal estable.

Para ello debemos convertir todos los datos de las fechas en un objeto *datatime*, puesto que el programa no detecta los datos como una fecha, usando la función *parse* para posteriormente agregarse a la lista empleando la función *append*.

Creamos un *DataFrame* con las dos columnas de los datos de las fechas y del tráfico en dicha fecha con la biblioteca Pandas. Haciendo uso de *set_index* establecemos como índice la columna de las fechas para poder acceder a los datos de forma más rápida.

A través de la función *resample*, podemos establecer intervalos temporales de 10 minutos, puesto que no hay un dato exacto del valor del tráfico cada 10 minutos, debemos obtener dicho valor operando con la media de los valores.

El último problema que podemos tener es que falten datos o no existan datos en algunos periodos de tiempo debido a errores o fallos en los sensores o bien debido a fallos del proceso. Por ello debemos realizar una interpolación de los datos para asegurarnos de que ningún dato vacío pueda provocar un error en el proceso posterior.

4.2 Modelo Suavizado *Exponencial Simple*

Este es el procedimiento más fácil de aplicar puesto que existe una función llamada *ExponentialSmoothing()*, que se encarga de realizar el modelo para la predicción usando el método de suavizado Exponencial Simple.

Los elementos que debemos definir en la función *ExponentialSmoothing()* son:

- La serie temporal, debemos introducir la serie temporal que queremos que use para predecir los datos futuros. En nuestro caso en la serie faltan los últimos 1000 datos para poder comprobar posteriormente la eficacia del modelo.
- Debemos indica la tendencia de la serie a través del elemento *trend*. Podemos indicar que la tendencia cambia ligeramente con el tiempo con "*add*", si cambia exponencialmente habría que indicar "*mul*" y si no se considera ninguna tendencia "*none*".
- Además de la tendencia hay que indicar la estacionalidad de la serie de datos con el elemento *seasonal*. Para ello "*add*" indica una estacionalidad aditiva, "*mul*" una estacionalidad multiplicativa y "*none*" si no se considera ninguna estacionalidad.
- Debemos indicar la cantidad de periodos por el por cada estacionalidad con el elemento *seasonal_periods*. Por ejemplo, si se reciben datos cada hora y su estacionalidad es un día deberemos introducir *seasonal_periods*=24.
- También hay que indicar el método que debemos usar para inicializar los parámetros del modelo, usando el elemento *initialization_method*. Existen 3 métodos distintos, "*estimated*" que inicializa los parámetros a valores estimados, "*heuristic*" el cuál inicializa los parámetros utilizando heurísticas y "*known*" utiliza valores conocidos para inicializar los parámetros.

Para poder averiguar qué podemos indicar en los elementos sobre la tendencia y sobre la estacionalidad podemos dibujar las gráficas de tendencia y la estacionalidad de la serie temporal.

```
# Realizar la descomposición estacional
res = sm.tsa.seasonal_decompose(df_resample, period=672)


# Obtener las componentes
trend = res.trend
seasonality = res.seasonal
residuals = res.resid

# Visualizar las componentes
plt.subplot(411)
plt.plot(df_resample, label='Original')
plt.legend(loc='upper left')

plt.subplot(412)
plt.plot(trend, label='Tendencia')
plt.legend(loc='upper left')

plt.subplot(413)
plt.plot(seasonality, label='Estacionalidad')
plt.legend(loc='upper left')

plt.subplot(414)
plt.plot(residuals, label='Residuos')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

Fig 7.Código de obtención de la estacionalidad, tendencia y los residuos.

Usamos para ello la función *seasonal_decompose()*, la cual se utiliza para realizar la descomposición estacional de una serie temporal. La descomposición estacional es una táctica que permite separar una serie temporal en sus componentes principales: tendencia, estacionalidad y residuos. Una vez separados podemos dibujar las gráficas tanto de la serie como de cada componente

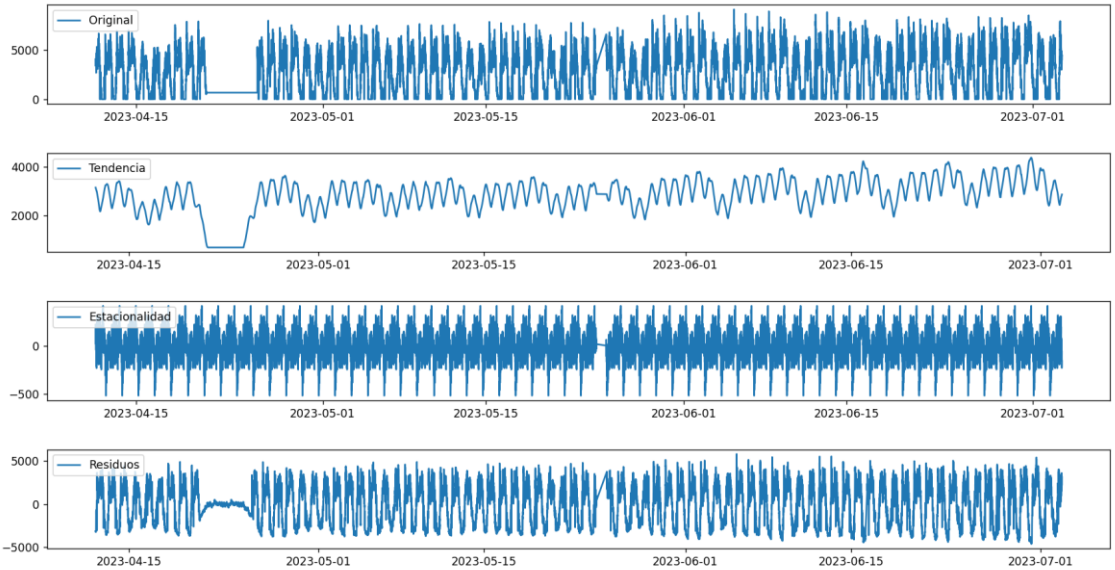

Fig 8. Gráficas para comprobar la tendencia, la estacionalidad y los residuos en comparación a los datos.

Observando las gráficas podemos percibir que tanto la tendencia como la estacionalidad podemos indicarlo como '*add*', para la cantidad de periodos podemos indicar 672 debido a que los datos se repiten cada 15 minutos y un patrón completo podría ser cada semana, lo que produce cada 7dias*24horas*4intervalos=672 muestras. Debido a la poca información de cuál puede ser mejor *initialization_method* probaremos los tres y elegiremos aquel que nos dé mejor resultado. En este caso es '*estimated*'. Dándonos así un resultado de error cuadrático medio de 990 y un error cuadrático medio raíz de 1100.

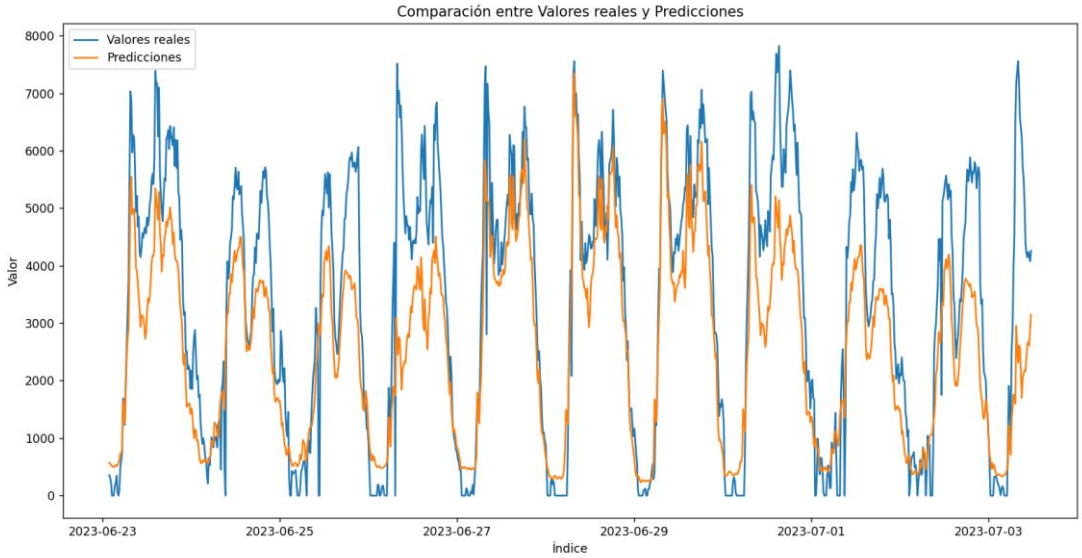

Fig 9.Gráfica de comparación de los datos reales con los datos predichos del modelo exponencial simple.

Podemos observar en la gráfica que las predicciones se asemejan bastante a los valores reales, aunque en las últimas muestras se diferencien un poco, posiblemente debido a que son los últimos datos conocidos.

4.3 Modelo ARIMA

Como ya he comentado anteriormente para el modelo de predicción temporal ARIMA lo más importante es obtener los parámetros p, q y d para que el modelo funcione de la manera que sea óptima.

Con este propósito se ha decidido realizar varias formas de obtener dichos parámetros. Una vez hallados los parámetros de las diferentes técnicas se realiza una predicción usando la función ARIMA, donde se predicen los últimos valores conocidos de la serie para comprobar la cantidad de error y la similitud entre las predicciones y los valores reales.

```
serie_completa = df_resample['IT'][:-1000]
modelo_arimadef = ARIMA(serie_completa, order=(mejor_p, mejor_d, mejor_q))
modelo_arima_fitdef = modelo_arimadef.fit()
fechas_predicciones = df_resample.index[-1000:]
predicciones = modelo_arima_fitdef.predict(start=len(df_resample), end=len(df_resample) + 999)
error = np.mean(np.abs(predicciones - muestras_conocidas))
plt.plot(fechas_predicciones,muestras_conocidas, label='Valores reales')
plt.plot(fechas_predicciones, predicciones, label='Predicciones')
plt.xlabel('Índice')
plt.ylabel('Valor')
plt.title('Comparación entre Valores reales y Predicciones')
plt.legend()
plt.show()
print(error)
```

Fig 10. Código de ejecución del modelo ARIMA una vez obtenidos los parámetros.

Usando la función ARIMA() creamos el modelo de predicción introduciendo la serie temporal menos los últimos 1000 valores puesto que son aquellos valores que queremos predecir para comprobar si nuestro método funciona correctamente. Además, debemos introducir los parámetros p, q y d hallados.

Una vez creado el modelo realizamos las predicciones de los últimos 1000 valores conocidos, para comprobar su eficacia calculamos el error medio absoluto (MAE) y creamos unas gráficas donde podamos comparar los resultados de las predicciones junto a los valores reales de la serie temporal.

4.3.1 Modelo a partir de gráficas

La primera manera de obtener los parámetros es determinar p y q a partir de diferentes gráficas y determinar d dependiendo de la cantidad de diferenciaciones que ha hecho falta para que la serie sea estacionaria.

Primero vamos a hallar el valor d para ello vamos a usar una función *adfuller* y *kpss.*

```
adfuller_result = adfuller(df_resample['IT'])
kpss_result = kpss(df_resample['IT'])
if adfuller_result[1] <= 0.05:
    print("La serie es estacionaria en media")
    print(adfuller_result[1])
else:
    print("La serie no es estacionaria en media")

if kpss_result[1] <= 0.05:
    print("La serie es estacionaria en varianza")
    print(kpss_result[1])
else:
    print("La serie no es estacionaria en varianza")
```

Fig 11. Código para comprobar si los datos son estacionarios.

La función *adfuller()* realiza la prueba de Dickey-Fuller, que determina si una serie temporal es estacionaria en la media. La prueba de Dickey-Fuller tiene una hipótesis nula (H0) que establece que la serie de tiempo tiene una raíz unitaria, lo que indica que no es estacionaria en la media (es decir, tiene una tendencia o deriva significativa). Si no se presenta una tendencia significativa en la serie de tiempo, tendrá la hipótesis alternativa (H1), según la cual la serie de tiempo es estacionaria en la media.

Para poder evaluar si es estacionaria o no usamos el valor de *adfuller_result*[1] que corresponde con el valor-p. El valor-p indica la probabilidad de que la hipótesis nula se cumpla y conduzca a los resultados observados. Se rechaza la hipótesis nula y podemos concluir que la serie es estacionaria en la media si el valor-p es menor o igual a un umbral predeterminado, en este caso 0.05 el cuál es el valor más recomendado para usar.

En el caso de *kpss*() es exactamente igual a *adfuller*() pero este evalúa si es estacionaria o no en varianza.

En ambos casos nos muestra un resultado de que es estacionario, por lo que no es necesario realizar ninguna diferenciación y el valor de d será de 0.

Para el parámetro p y q vamos a usar las gráficas de la autocorrelación, a través de las funciones *plot_acf* y *plot_pacf*, las cuales ofrecen una gráfica de la autocorrelación y de la autocorrelación parcial respectivamente:

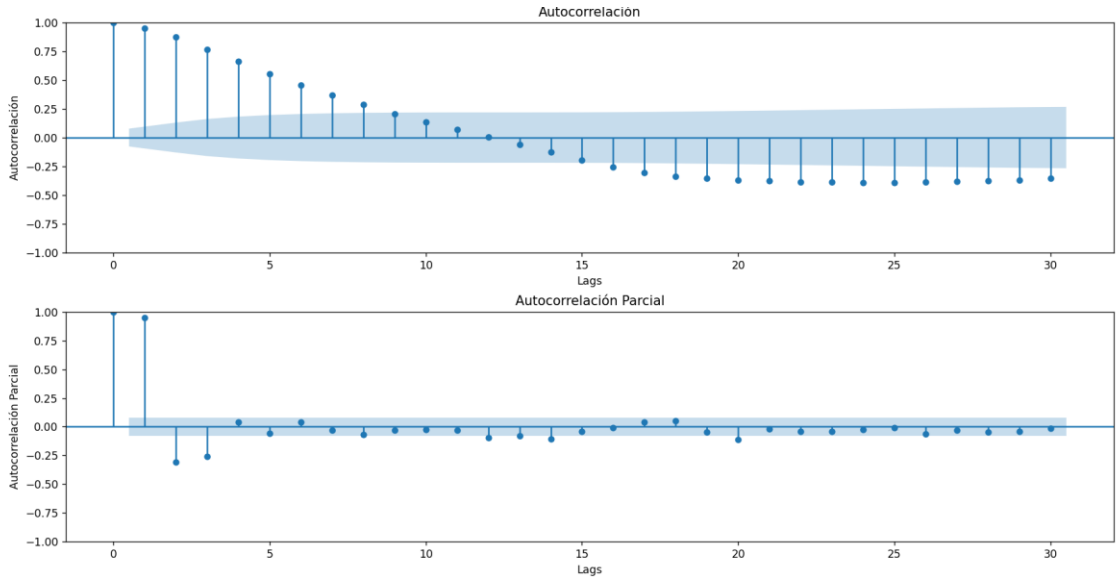

Fig 12. Gráficas de la autocorrelación y la autocorrelación parcial.

A través de la gráfica de autocorrelación podemos observar que los *lags* más significativos son alrededor de 8 y en la gráfica de autocorrelación parcial los las más significativos son 4. Por lo que supondremos un valor de p=4 y q=8.

Ahora falta comprobar qué predicciones nos produce el método ARIMA para los parámetros establecidos.

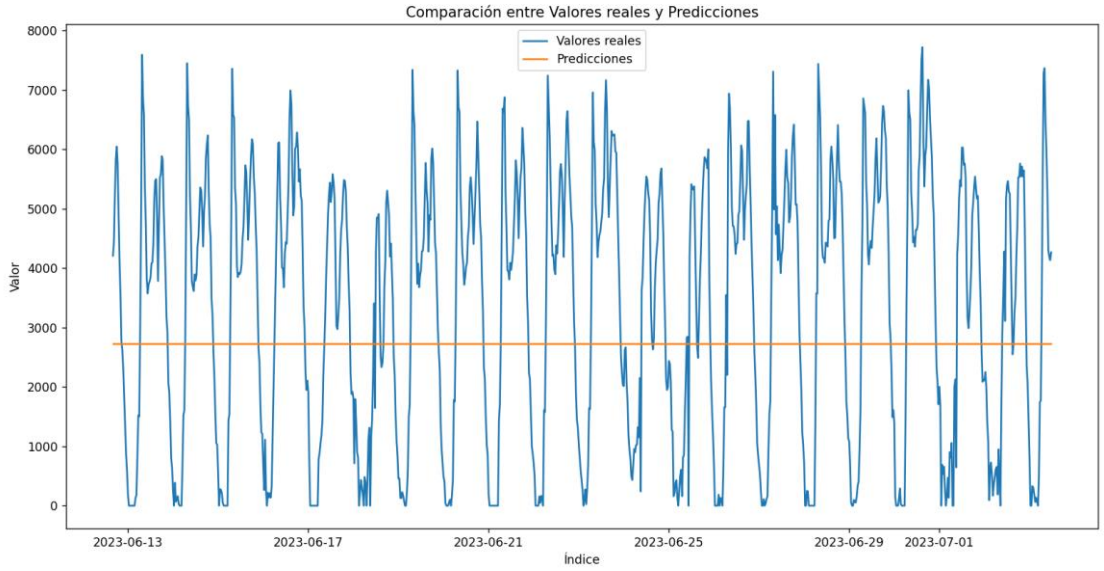

Fig 13. Gráfica de comparación de los datos reales con los datos predichos del modelo ARIMA con gráficas.

Solo observando la gráfica nos podemos dar cuenta que los parámetros elegidos no son los adecuados, puesto que las predicciones se mantienen constantes y no intentan parecerse a los valores reales. Aparte de esto aporta un MAE de 2178. Por lo tanto, podemos corroborar que no es el método óptimo para encontrar los parámetros p, q y d.

4.3.2 Modelo automático

El segundo método que vamos a investigar para obtener los parámetros es una función que consigue los valores óptimos para la serie tempo introducida. Esta función es *auto_arima*(), los elementos a introducir en la función son los siguientes:

La serie temporal de la cual se quieren obtener p, q y d. También hay que indicar los valores iniciales de p, q y d, es decir, desde que número iniciará la búsqueda de dichos parámetros. En nuestro caso el valor inicial de los tres parámetros es 0.

Los valores máximos de p, q y d, es decir cuál será el mayor valor que comprobará la función. En nuestro caso el valor máximo de p y q son 20 debido a que las gráficas de las autocorrelaciones hay una gran cantidad de *lags* significativos y el máximo del parámetro d es 3 debido a que es inusual realizar más de 3 diferenciaciones para alcanzar la estacionalidad de la serie temporal.

Puesto que estamos buscando un ARIMA tenemos que indicar que el *seasonal=False*, puesto que si indicamos *seasonal=True*, la función buscará los parámetros necesarios para un SARIMA.

En el último elemento importante de la función el cuál es *method*, es decir que método usar para obtener los parámetros, existen diversos métodos entre ellos están *'lbfgs' (Limited-memory BFGS), 'bfgs' (Broyden-Fletcher-Goldfarb-Shanno), 'nm' (Nelder-Mead) o 'cg' (Conjugate Gradient)* entre otras. En nuestro caso vamos a ir variando el método para ver cuál se ajusta más.

Una vez ejecutada la función variando el elemento *method* entre las diferentes opciones que hay podemos examinar los resultados conseguidos. Nos podemos dar cuenta que al igual que el método anterior las predicciones realizadas son una constante que no se asemeja a los valores reales de la serie temporal. Y por tanto el error sigue siendo demasiado alto.

Además los valores de los parámetros que nos proporcionaban como resultado el método *auto_arima*(), variaba dependiendo del *method* usado, aunque todas arrojaban un resultado similar.

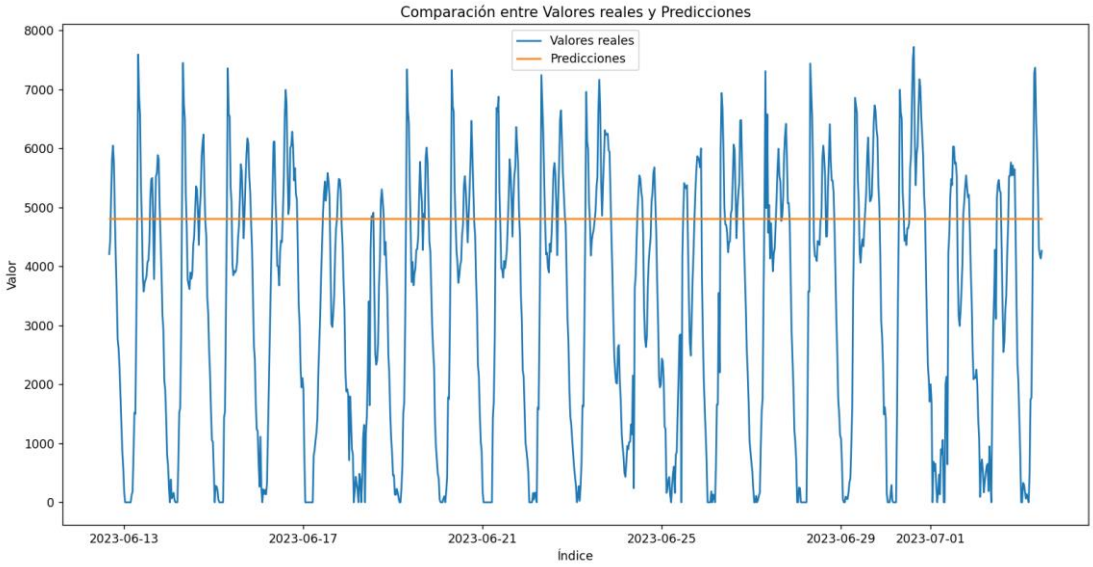

Fig 14. Gráfica de comparación de los datos reales con los datos predichos del modelo ARIMA automática.

4.3.3 Modelo ARIMA usando bucles

El último procedimiento con el que vamos a intentar obtener los parámetros del modelo ARIMA es usando bucles. Este sistema es más lento que el resto puesto que tiene que probar todas las posibles combinaciones de los parámetros.

El código consta de 3 bucles *for*, uno por cada parámetro que van desde 0 a 20 en el caso de los bucles que son en función de p y q, mientras que el otro bucle va de 0 a 3.

```
# Rangos de valores para p, d y q
rango_p = range(0, 20)
rango_d = range(0, 2)
rango_q = range(0, 20)

mejor_p = None
mejor_d = None
mejor_q = None
error_minimo = np.inf
```

Fig 15. Código de iniciación de los valores de los parámetros.

Se inician los valores de mejor_p, mejor_q y mejor_d que es donde guardaremos los parámetros que cumplan que son óptimos. Además, hay que iniciar el error mínimo a infinito, puesto que se trata de obtener el menor error, si debemos compararlo con el resto de los errores este debe ser el mayor.

```python
# Iterar sobre todos los valores de p, d y q
for p in rango_p:
    for d in rango_d:
        for q in rango_q:
            try:
                # Ajustar el modelo ARIMA con los valores actuales de p, d y q
                modelo_arima = ARIMA(serie_completa, order=(p, d, q))
                modelo_arima_fit = modelo_arima.fit()

                # Realizar las predicciones para las últimas 1000 muestras conocidas
                predicciones = modelo_arima_fit.predict(start=len(serie_completa), end=len(serie_completa) + 999)

                # Comparar las predicciones con los valores conocidos de las últimas 1000 muestras
                muestras_conocidas = df_resample['IT'][-1000:]
                error = np.mean(np.abs(predicciones - muestras_conocidas))

                # Actualizar los mejores valores si se encuentra un error mínimo menor
                if error < error_minimo:
                    error_minimo = error
                    mejor_p = p
                    mejor_d = d
                    mejor_q = q
            except:
                continue
```

Fig 16. Código para obtener los parámetros usando bucles.

Dentro de estos bucles se realiza la función ARIMA() para crear el modelo de predicción con los valores de los parámetros que corresponden en ese instante. Se realizan las predicciones de los últimos 1000 valores de la serie temporal y se calcula el error que hay entre los valores reales y los valores obtenidos de las predicciones.

Una vez se ha calculado el error este valor se comprueba con el error óptimo que se ha obtenido, es decir con el valor de error más bajo que se ha obtenido. Si se cumple que el nuevo error contiene un valor menor que el error_mínimo, se guardan los valores de los parámetros y el nuevo error pera poder comprobarlos con el resto de los errores.

Tras completar los bucles los valores guardados de las variables mejor_p, mejor_q y mejor_d son los valores óptimos para la serie temporal, en nuestro caso los valores obtenidos son p=6, q=0 y d=7 dándonos si un error medio absoluto de 900.

Al representar los valores obtenidos y los reales nos damos cuenta de que el modelo no intenta predecir los valores si no que solo realiza un sinusoidal, por lo que descartamos definitivamente el modelo ARIMA para poder predecir nuestra serie temporal.

Las casusas de que el modelo ARIMA no consiga predecir la serie temporal pueden ser muy diversas y pueden deberse a varios factores. Una de las razones puede ser la complejidad de la serie temporal, es decir, que cuando una serie temporal es compleja, ARIMA puede tener dificultades para proporcionar predicciones precisas, ya que está diseñado para capturar patrones lineales y dependencias a corto plazo.

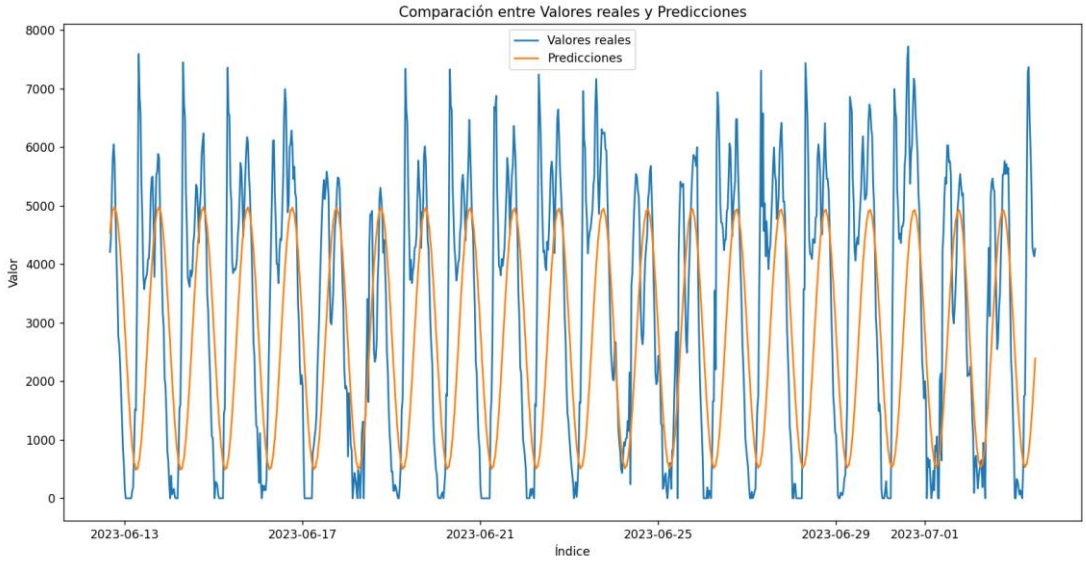

Fig 17. Gráfica de comparación de los datos reales con los datos predichos del modelo ARIMA por bucles.

4.3.4 Modelo automático para SARIMA

Puesto que no hemos conseguido que el método ARIMA funcione correctamente vamos a comprobar si a través del método SARIMA podemos acercarnos más a los resultados esperados. Para ello vamos a necesitar además de los parámetros anteriores (p, q y d) también necesitaremos los parámetros P, Q, D y s.

Para ello vamos a adaptar el método *auto_arima*() con el fin de que obtenga también estos parámetros. Para llevar a cabo este procedimiento debemos cambiar algunos elementos de la función, en particular el *seasonal* a True, esto provoca que el *auto_arima*, aparte de buscar los parámetros anteriores también busque los nuevos. Además, debemos añadir un máximo a los nuevos parámetros que será de 20 para P y Q, 3 para D, puesto que los valores más usuales para estos parámetros son algo más bajos a estos números. El valor de s corresponderá a 672, puesto que los datos se repiten cada 15 minutos y un patrón completo podría ser cada semana, puesto que 7días*24horas*4intervalos=672 muestras.

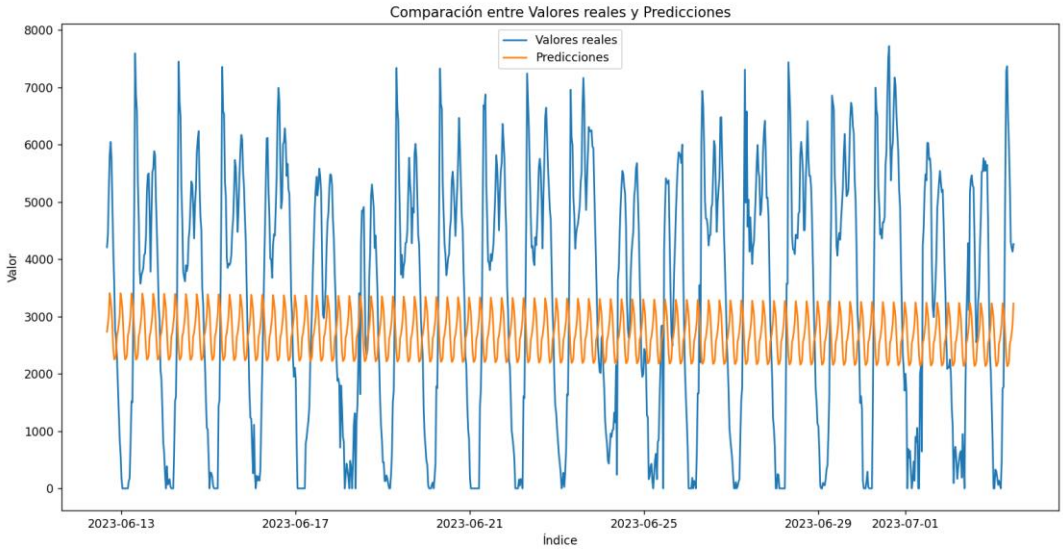

Fig 18. Gráfica de comparación de los datos reales con los datos predichos del modelo SARIMA automática.

El resultado, a diferencia del anterior *auto_arima*(), no es una constante, pero como se puede apreciar aún dista mucho del resultado esperado. Por lo tanto, el error que otorga es demasiado grande.

También vamos a probar a usar el anterior método de los bucles para realizar el método SARIMA. Para ello, de nuevo, vamos a ajustar el procedimiento para que aparte de buscar los parámetros p, q y d, también busque los parámetros P, Q y D. En este caso supondremos que s=672 puesto que es el valor que hemos calculado anteriormente para esta serie de datos.

Para adaptar este modelo a SARIMA tenemos que añadir más bucles, en este caso con los parámetros P, D y Q, con las mismas condiciones que el caso anterior.

```
# Rangos de valores para p, d, q, P, D, Q y s
rango_p = range(0, 20)
rango_d = range(0, 2)
rango_q = range(0, 20)
rango_P = range(0, 20)
rango_D = range(0, 3)
rango_Q = range(0, 20)
s = 12  # Número de pasos en la estacionalidad

# Variables para almacenar los mejores valores y el error mínimo
mejor_p = None
mejor_d = None
mejor_q = None
mejor_P = None
mejor_D = None
mejor_Q = None
error_minimo = np.inf
```

Fig 19. Código de inicialización de los parámetros del modelo SARIMA.

```
# Bucle para encontrar los mejores valores para p, d, q, P, D y Q
for p in rango_p:
    for d in rango_d:
        for q in rango_q:
            for P in rango_P:
                for D in rango_D:
                    for Q in rango_Q:
                        try:
                            # Ajustar el modelo SARIMA
                            modelo_sarima = SARIMAX(serie_completa, order=(p, d, q), seasonal_order=(P, D, Q, s))
                            modelo_sarima_fit = modelo_sarima.fit()

                            # Calcular el error medio absoluto (MAE) para el modelo ajustado
                            muestras_conocidas = df_resample['IT'][-1000:]
                            predicciones = modelo_sarima_fit.predict(start=len(df_resample), end=len(df_resample) + 999)
                            error = np.mean(np.abs(predicciones - muestras_conocidas))

                            # Actualizar los valores óptimos si se encuentra un modelo con un error menor
                            if error < error_minimo:
                                mejor_p = p
                                mejor_d = d
                                mejor_q = q
                                mejor_P = P
                                mejor_D = D
                                mejor_Q = Q
                                error_minimo = error

                        except:
                            continue
```

Fig 20. Código para obtener los parámetros usando bucles en el modelo SARIMA.

El resultado de este modelo es notoriamente mejor que los anteriores, pero se puede comprobar que aún se encuentra muy alejado de los resultados reales. A diferencia del método ARIMA el método SARIMA se acerca más a la forma de los datos reales, pero no consigue acercarse a los máximos

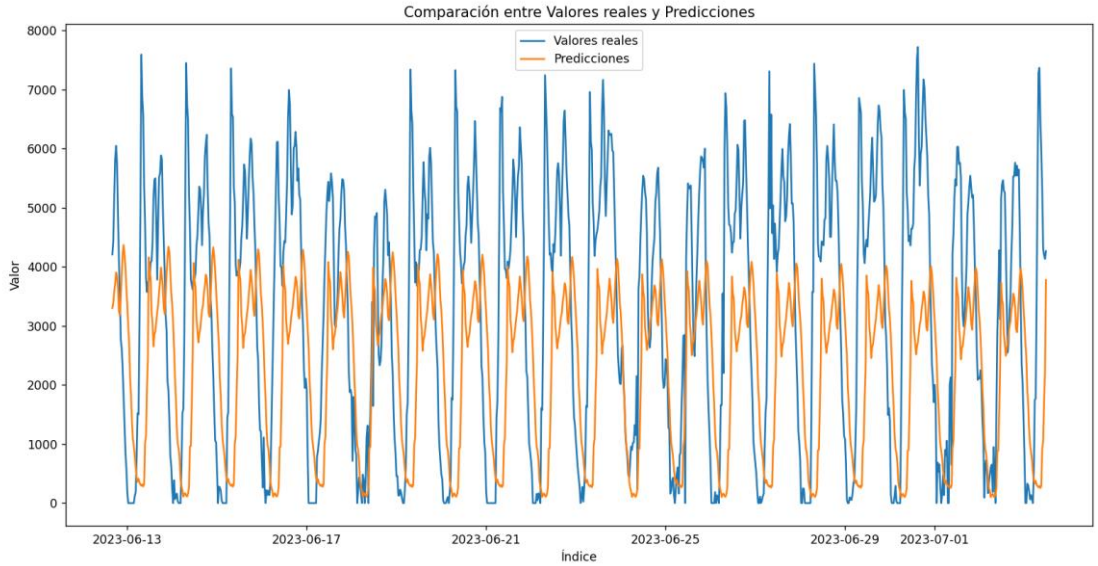

Fig 21. Gráfica de comparación de los datos reales con los datos predichos del modelo SARIMA realizado por bucles.

4.3.5 Redes neuronales LSTM

Después de investigar diferentes redes neuronales para comprobar cuál es la que puede ser óptima para una serie temporal, se decide usar una red neuronal LSTM. Debido a su capacidad para reconocer dependencias a largo plazo, administrar secuencias de longitud variable y descubrir patrones intrincados en los datos, las redes LSTM son las adecuadas para problemas de series temporales.

Lo primero que debemos hacer es el escalamiento de los datos que se trata de un método para ajustar las características o variables de un conjunto de datos dentro de un rango predeterminado. El objetivo principal del escalado es cambiar las variables para que tengan una escala similar, evitando así problemas causados por la disparidad en la magnitud de las características. Para ello usamos el objeto de *MinMaxScaler* de la biblioteca de *Sklearn.*

```
# Escalar los datos
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset.values.reshape(-1, 1))
```

Fig 22. Código escalamiento de datos.

El objeto *MinMaxScaler* es una técnica de normalización que modifica y transforma los datos para que los valores mínimo y máximo se escalen proporcionalmente dentro del rango (0, 1).

La función *fit_transform()* es utilizada para realizar el escalado de los datos de la serie temporal, sin embargo se espera recibir los datos en una columna en una matriz 2D, para ello se usa el método *values.reshape*(-1, 1).

Los datos escalados, de manera que el valor mínimo de los datos se transforme en 0 y el valor máximo se transforme en 1, mientras que el resto de los valores se escalan proporcionalmente en el rango (0, 1), se guardan en la variable *dataset.*

Una vez hemos escalado los datos de la serie temporal debemos realizar la división de los datos en el conjunto de entrenamiento y en el conjunto de test.

```
# División en conjunto de entrenamiento y prueba
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size], dataset[train_size:len(dataset)]
```

Fig 23. Código de división del dataset para el entrenamiento y el test.

Cuando entrenamos un modelo, queremos que descubra relaciones y patrones en los datos para que pueda hacer predicciones precisas sobre datos nuevos. Como resultado, el conjunto de entrenamiento se usa para ajustar los parámetros del modelo y determinar cómo representar mejor la relación entre las variables de entrada y salida.

Por otro lado, el rendimiento del modelo después del entrenamiento se evalúa mediante el conjunto de pruebas. De esta manera podemos probar la capacidad del modelo para generalizar correctamente a nuevos datos y no solo memorizar los ejemplos de entrenamiento.

Para ello, primero creamos las longitudes de los dos conjuntos, estableciendo una longitud para el entrenamiento del 67% de la totalidad de los datos, mientras que para la prueba el 33% de la totalidad de los datos. Una vez estipulado la longitud introducimos los valores de la serie temporal correspondientes en las variables de *train* y test.

Tras crear los dos conjuntos, debemos crear las secuencias de entrenamiento para entrenar un modelo de red neuronal LSTM en una serie temporal. La secuenciación para el entrenamiento es un paso crucial en el proceso de procesamiento de datos.

El objetivo de este procedimiento es dividir la serie temporal en partes manejables que el modelo pueda usar para comprender las relaciones temporales y los patrones en los datos.

```
def create_dataset(dataset, look_back):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        a = dataset[i:(i + look_back)]
        dataX.append(a)
        dataY.append(dataset[i + look_back])
    return numpy.array(dataX), numpy.array(dataY)

trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], trainX.shape[2]))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], testX.shape[2]))
```

Fig 24. Código de creación de las secuencias.

El proceso de creación de secuencias se realiza en la función *create_dataset*. En esta función se toman dos argumentos, *dataset*, que se trata de la serie de datos y *look_back* Es una variable que controla qué tan grande se asignará una ventana de tiempo para producir las secuencias de entrenamiento.

Las secuencias de entrenamiento se mantendrán en estas listas, *dataX* y *dataY*. *dataX* contendrá las secuencias de entrada, que son ventanas de tiempo secuenciales de tamaño *look_back*. Mientras que *dataY* contendrá los valores de series temporal para el paso de tiempo que sigue inmediatamente a cada secuencia de *dataX*.

Ahora podemos configurar la red LSTM. Con este propósito, usamos la función *Sequential*(), esta función se usa para crear un objeto de modelo secuencial en *Keras*. Un modelo secuencial es aquel

en el que las capas de la red neuronal se apilan secuencialmente, una encima de la otra. Cada capa se conecta a la capa superior a través de una entrada y una salida. Es el tipo de modelo más típico y directo en *Keras* y funciona bien para muchas aplicaciones de aprendizaje profundo.

```
model = Sequential()
model.add(LSTM(55, input_shape=(look_back, 1), return_sequences=True))
model.add(Dropout(0.28))
model.add(LSTM(55, return_sequences=True))
model.add(Dropout(0.43))
model.add(LSTM(126))
model.add(Dropout(0.44))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer=Adam(lr=0.001))
history = model.fit(trainX, trainY, epochs=150, batch_size=16, validation_data=(testX, testY), verbose=1)
```

Fig 25. Código de la estructura de la red neuronal.

En nuestro caso hemos añadido un total de 3 capas de neuronas usando *model.add*. En la primera y la segunda capa hemos introducido 55 neuronas en cada una, mientras que en la tercera capa hemos introducido 126 neuronas.

Cuando *return_sequences* se establece en True, la capa LSTM devuelve la secuencia completa de salidas en lugar de solo la última salida. Esto es útil cuando se apilan varias capas LSTM, ya que permite pasar la secuencia completa de salidas a la siguiente capa LSTM en lugar de solo la salida final.

En cuanto a *input_shape=(look_back, 1)* se refiere a la forma de entrada que espera la capa LSTM. En este caso, *look_back* representa el tamaño de las ventanas de tiempo que se utilizarán como entrada para el modelo. Cada ventana de tiempo tiene una longitud de *look_back* y contiene valores de la serie temporal consecutivos. La dimensión 1 indica que cada punto de datos en la ventana de tiempo tiene una sola característica.

Después de cada capa de neuronas hemos aplicado un *Dropout*. *Dropout* es una técnica de regularización utilizada para reducir el sobreajuste (*overfitting*) en el modelo. El *overfitting* ocurre cuando el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a datos nuevos o de prueba.

El *Dropout* es una capa que se aplica después de otra capa. Su propósito es "apagar" aleatoriamente un porcentaje de neuronas en la capa anterior durante el entrenamiento. Esto significa que, durante cada paso de entrenamiento, algunas neuronas no se activarán y no contribuirán al cálculo de las salidas de la capa. Como resultado, el modelo se vuelve más robusto y evita depender demasiado de ciertas neuronas específicas.

Por ejemplo, en el primer *Dropout* de 0.28 indica que el 28% de las neuronas de la primera capa se apagarán aleatoriamente en cada paso.

Al final de todas las capas añadimos *model.add(Dense(1))*. Se refiere a la adición de una capa densa (*Dense*) a la red neuronal. La capa densa es también conocida como capa totalmente conectada, y es una de las capas fundamentales en una red neuronal.

En una capa densa, todas las neuronas de la capa están conectadas a todas las neuronas de la capa anterior. Es decir, cada entrada de la capa densa tiene una conexión ponderada con todas las salidas de la capa anterior. En este caso a una única neurona.

Por último, añadimos la función *model.compile*() que se encarga de la compilación del modelo antes del entrenamiento. En la función debemos introducir dos argumentos:

- *Loss*: Especifica la función de pérdida *(loss function*) que se utilizará durante el entrenamiento para medir la discrepancia entre las predicciones del modelo y los valores reales. En este caso, se está utilizando la función de pérdida "*mean_squared_error*", que también se conoce como MSE (Error Cuadrático Medio). El MSE es una función de pérdida comúnmente utilizada en problemas de regresión, ya que penaliza más

fuertemente las predicciones que están más lejos de los valores reales. El objetivo del entrenamiento será minimizar esta función de pérdida.

- *Optimizer:* Especifica el algoritmo de optimización que se utilizará para ajustar los pesos de la red neuronal durante el entrenamiento. Aquí se está utilizando el optimizador Adam, que es una variante del descenso de gradiente estocástico (SGD) que adapta automáticamente la tasa de aprendizaje durante el entrenamiento. El argumento lr=0.001 establece la tasa de aprendizaje del optimizador en 0.001, lo que controla el tamaño de los pasos que se dan para ajustar los pesos.

Los valores usados en la configuración de la red han sido escogidos probando diferentes combinaciones y ajustándolos a través de experimentación y validación.

Ahora ya podemos realizar predicciones con el modelo entrenado y luego de transformar nuevamente las predicciones escaladas a su escala original para poder compararlas con los valores reales de la serie temporal.

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

trainPredict = trainPredict.reshape(-1, 1)
testPredict = testPredict.reshape(-1, 1)

trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform(trainY)
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform(testY)
```

Fig 26. Código para escalar los datos a la versión original.

Se procede a realizar predicciones tanto en el conjunto de entrenamiento (*trainX*) como en el conjunto de prueba (*testX*). Para esto, se utiliza el método *model.predict*() que recibe como entrada las secuencias de entrada *trainX* y *testX*, y devuelve las predicciones correspondientes para cada paso de tiempo. Las predicciones se almacenan en las variables *trainPredict* y *testPredict*, respectivamente.

Dado que las predicciones fueron escaladas previamente durante el proceso de preparación de datos, es necesario transformarlas nuevamente a su escala original para poder compararlas con los valores reales de la serie temporal.

Primero, se utiliza el objeto *scaler* que fue previamente creado con *MinMaxScaler* para realizar la transformación inversa (*scaler.inverse_transform*()) de las predicciones. Esto revierte la escala realizada previamente y devuelve las predicciones en la escala original.

Las predicciones escaladas se transforman de nuevo en matrices 1D utilizando *reshape*(-1, 1), ya que el modelo espera que las predicciones tengan esta forma para poder compararlas con los valores reales.

Las variables *trainPredict* y *testPredict* ahora contienen las predicciones transformadas en su escala original.

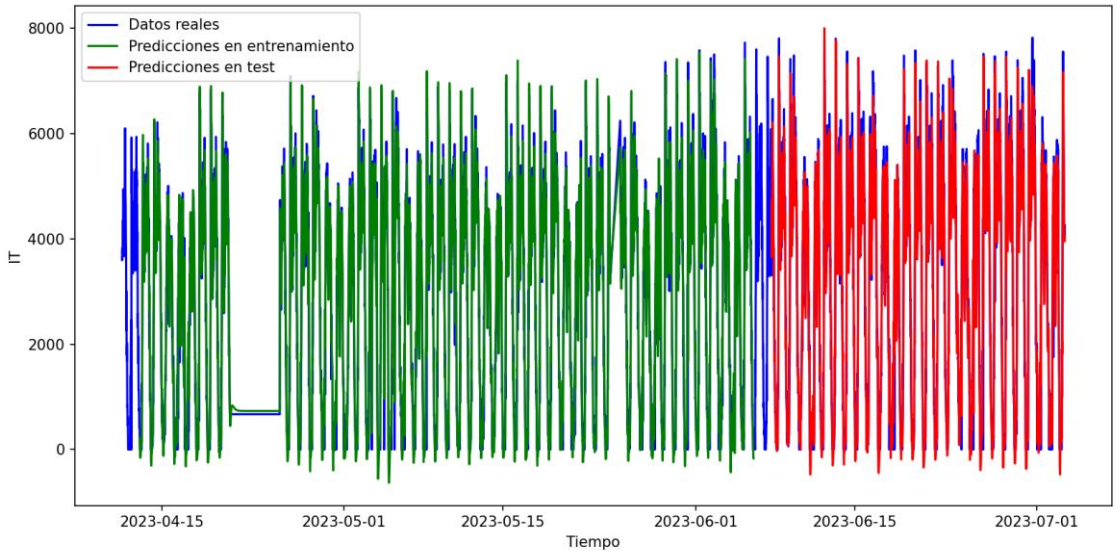

Fig 27. Gráfica de comparación de los datos reales con los datos predichos del modelo red neuronal LSTM.

Podemos observar que la predicción tanto del conjunto de entrenamiento como el de prueba es casi perfecta, otorgando un RMSE de 250, obteniendo así el mejor resultado de los diferentes modelos.

También podemos realizar la gráfica de las curvas de aprendizaje del conjunto de entrenamiento y de prueba para poder evaluar el rendimiento del modelo. Podemos comprobar que el modelo funciona perfectamente, puesto que nuestras curvas de aprendizaje muestran que tanto el conjunto de entrenamiento como el conjunto de prueba convergen rápidamente hacia un bajo error y se mantienen estables a medida que aumenta el número de épocas de entrenamiento. Esto indica que el modelo es capaz de capturar de manera efectiva los patrones y relaciones en los datos de entrenamiento y generalizar adecuadamente a datos no vistos durante el entrenamiento.

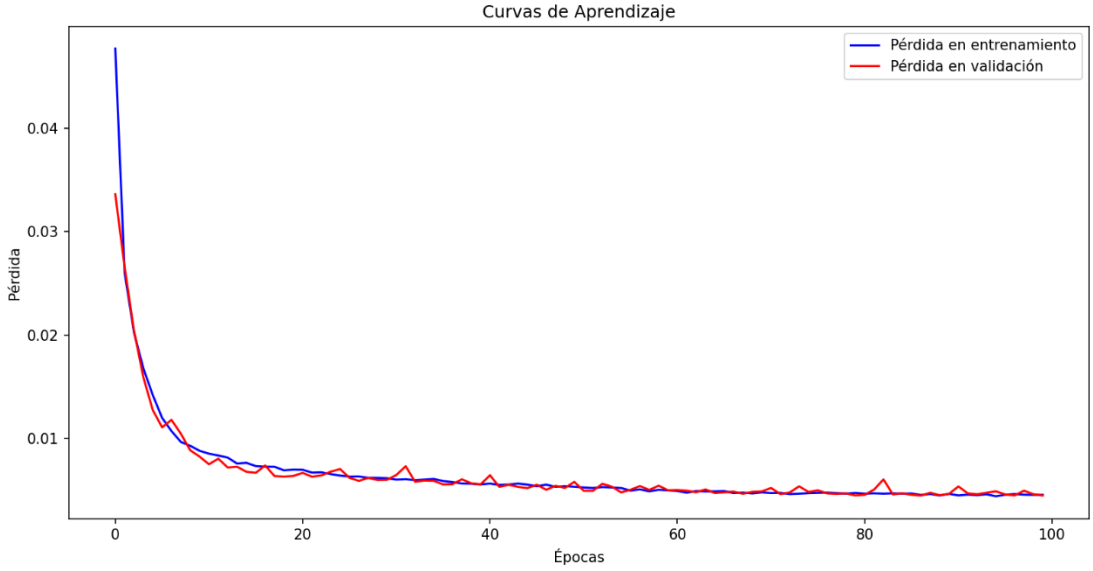

Fig 28. Gráfica para comprobar el sobreajuste de la red neuronal.

Para comprobar que el código funciona con cualquier sensor disponible de la ciudad de Valencia, vamos a ejecutar el programa con varios sensores aleatorios, en este caso con los sensores “A109”, “A111”, “A120” y “A122”.

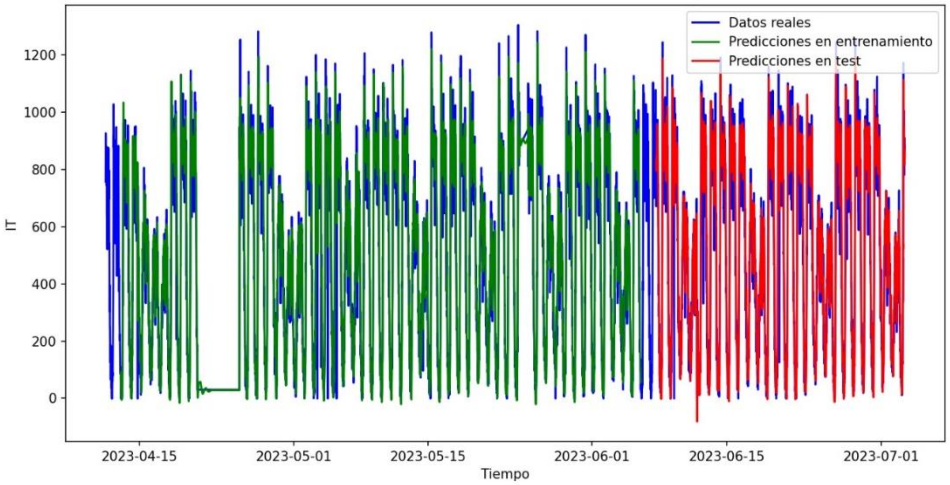

Fig 29. Resultados obtenidos del sensor A109

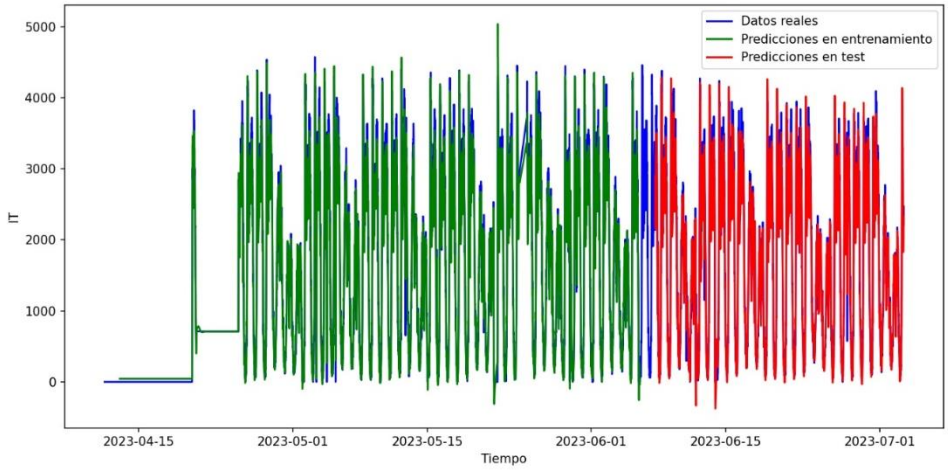

Fig 30. Resultados obtenidos del sensor A111

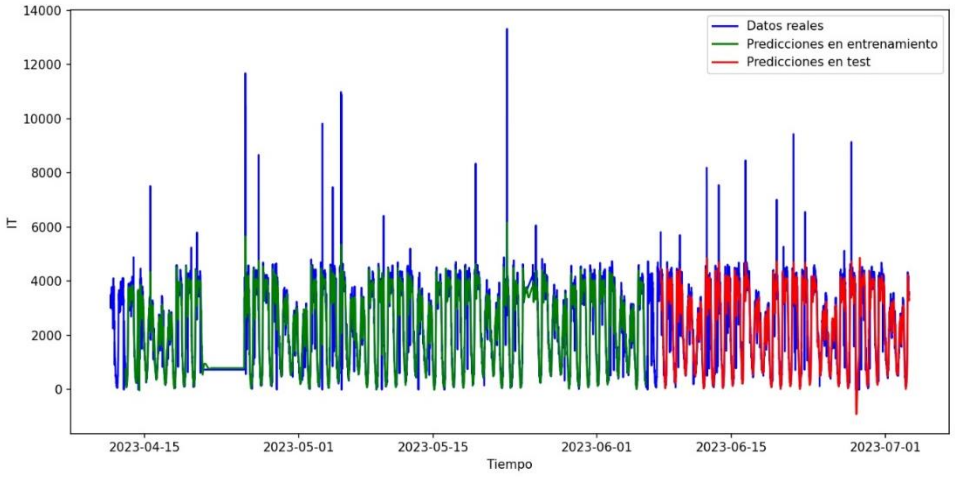

Fig 31. Resultados obtenidos del sensor A120

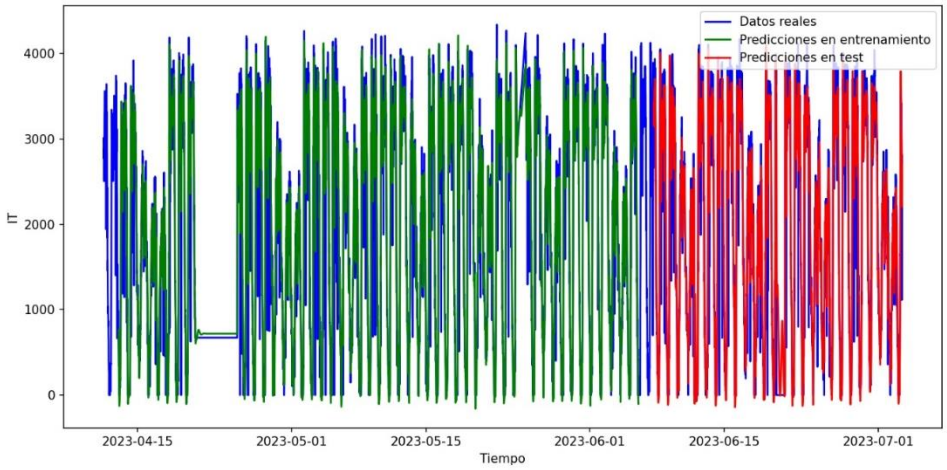

Fig 32. Resultados obtenidos del sensor A122

Como podemos observar en todos los sensores, los datos que hemos predicho se asemejan a los datos reales, además de otorgar un error bastante bajo en comparación a los otros métodos. En la gráfica de A120 es notable unos picos de tráfico demasiado altos, habría que estudiar el comportamiento de estos picos para comprobar su procedencia puesto que la red no es capaz de predecirlo.

Capítulo 5. Conclusiones y trabajos futuros

5.1 Conclusión

En la conclusión se destaca la importancia de la selección de la metodología en el análisis de series de tiempo. Durante este extenso estudio se examinaron el método ARIMA, el modelo de suavizado exponencial simple y una red neuronal.

Los resultados mostraron que, si bien ARIMA (además del método SARIMA) y los métodos de suavizado exponencial simple son enfoques convencionales valiosos, no se adaptaban bien al tipo de datos en cuestión. Sus deficiencias aparecieron al lidiar con los intrincados patrones de variación de los datos.

Sin embargo, el enfoque de las redes neuronales resultó ser muy eficiente y adaptable. La red neuronal pudo abordar patrones más complejos que los enfoques tradicionales, lo que le permitió no solo capturar y modelar con precisión las relaciones subyacentes en los datos, sino también sortear sus inconvenientes.

Estos hallazgos demuestran el valor de tener en cuenta métodos más sofisticados, como las redes neuronales, para el análisis de series temporales, particularmente cuando se trabaja con datos dinámicos y complejos.

En cuanto al método SARIMA, es posible que existan un patrón de parámetros p, q, d, P, Q y D que produzcan un resultado más cercano al actual. Sin embargo, no se ha podido conseguir una combinación con un resultado mejor al mostrado.

5.2 Trabajo futuro

Esta investigación allana el camino para una serie de posibles investigaciones y avances futuros en la gestión de infraestructuras urbanas. La ampliación del análisis a otras vías, como las diseñadas para ciclistas, sería una dirección prometedora.

Es necesaria una comprensión profunda de la demanda y el uso de carriles bici, dada la creciente popularidad de modos de transporte respetuosos con el medio ambiente, como la bicicleta. Se pueden utilizar modelos similares para estudiar los patrones de tráfico, la congestión y los requisitos de expansión en las carreteras para vehículos motorizados.

Sería ventajoso hacer accesibles los datos recopilados en un esfuerzo por aumentar la transparencia y la participación ciudadana. Los ciudadanos podrían tomar mejores decisiones sobre sus movimientos si se implementaran en las plataformas ya creadas por el ayuntamiento de València como puede ser “València al minut” o en gemelos digitales urbanos, que reflejaran el movimiento del tráfico en tiempo real y futuro. Además, hacer que estos datos estén disponibles públicamente en línea los convertiría en un recurso invaluable para los investigadores y el público en general.

Fig 33. Grafica del tráfico actual de València al minut

Una de las repercusiones importantes a considerar al hacer públicos los posibles datos futuros es la potencial modificación en el comportamiento de la población. Esta situación puede ser el resultado de la anticipación de eventos o cambios previstos, y a menudo es una respuesta natural ante la disponibilidad de información. Esta posible alteración del comportamiento de la ciudadanía plantea desafíos en términos de la eficiencia y precisión de los datos obtenidos. Si los datos se recopilan bajo la suposición de que las acciones de las personas seguirán un patrón específico, pero ese patrón se ve alterado debido a la información anticipada.

Se pueden incorporar datos más complejos y a más largo plazo a los modelos para proporcionar conocimiento histórico vital. Esto permitiría comprender más a fondo las tendencias a lo largo del tiempo y podría ser especialmente beneficioso para la planificación de proyectos de infraestructura a largo plazo. Es importante sobre todo para días que no se rigen por horarios normales como pueden ser festivos donde las calles se encuentren cortadas.

En la planificación de predicciones futuras, es fundamental no solo tener en cuenta los días festivos, sino también integrar información sobre las condiciones meteorológicas diarias. Esto se debe a que el clima desempeña un papel crucial en las decisiones de movilidad de la ciudadanía. En días lluviosos, por ejemplo, es más probable que las personas opten por utilizar el coche en lugar de caminar o utilizar medios de transporte alternativos. Del mismo modo, en días soleados, las personas pueden estar más inclinadas a disfrutar del aire libre o utilizar bicicletas y transportes públicos. Por lo tanto, incorporar datos meteorológicos en los modelos de predicción puede mejorar significativamente la precisión de las previsiones de tráfico y ayudar en la planificación de la movilidad urbana, lo que, a su vez, contribuiría a la reducción de la congestión del tráfico y la optimización de los recursos de transporte.

Los patrones de tráfico estacionales y cíclicos, así como las tendencias a largo plazo, pueden revelarse combinando datos de varias décadas con técnicas de modelización. Esto ofrece una visión detallada e invaluable de la dinámica del tráfico a lo largo del tiempo, lo que permite una planificación más precisa y efectiva de la infraestructura vial y los sistemas de transporte, así como una comprensión más completa de cómo cambian las necesidades. movilidad en una sociedad que está en constante evolución.

Capítulo 6. Bibliografía

[1] «Capítulo 1 Tipos de modelos: Predictivos vs Explicativos,» 19 05 2020. [En línea]. Available: https://derek-corcoran-barrios.github.io/CursoMulti/_book/CriteriosInfo.html. [Último acceso: 19 08 2023].

[2] «Series Temporales,» 2014. [En línea]. Available: https://bookdown.org/content/2274/series-temporales.html. [Último acceso: 29 08 2023].

[3] «Python España,» 2023. [En línea]. Available: https://es.python.org/. [Último acceso: 29 08 2023].

[4] «Datos abiertos del Gobierno de España,» Datos.gob.es, [En línea]. Available: https://datos.gob.es/es. [Último acceso: 29 08 2023].

[5] «Portal de Datos Abiertos del Ayuntamiento de València,» Portal de l'Ajuntament de la ciutat de València, 2019. [En línea]. Available: https://valencia.opendatasoft.com/pages/home/. [Último acceso: 29 08 2023].

[6] «TensorFlow,» TensorFlow, 2023. [En línea]. Available: https://www.tensorflow.org/?hl=es-419. [Último acceso: 29 08 2023].

[7] «statsmodels.tsa.holtwinters.ExponentialSmoothing¶,» statsmodel.org, 2014. [En línea]. Available: https://www.statsmodels.org/dev/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html. [Último acceso: 29 08 2023].

[8] «statsmodels.tsa.seasonal.seasonal_decompose,» Statsmodel.org, 2014. [En línea]. Available: https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html. [Último acceso: 29 08 2023].

[9] «pmdarima.arima.auto_arima,» Statsmodels.org, 2017. [En línea]. Available: https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html. [Último acceso: 29 08 2023].

[10] B. Etienne, «Time Series in Python — Exponential Smoothing and ARIMA processes,» Towards Data Science, 19 02 2019. [En línea]. Available: https://towardsdatascience.com/time-series-in-python-exponential-smoothing-and-arima-processes-2c67f2a52788. [Último acceso: 29 08 2023].

[11] «Python | Modelo ARIMA para Pronóstico de Series de Tiempo,» Barcelona Geeks, 05 07 2022. [En línea]. Available: https://barcelonageeks.com/python-modelo-arima-para-el-pronostico-de-series-de-tiempo/. [Último acceso: 29 08 2023].

[12] A. S. Alberca, «La librería Numpy,» Aprende con Alf., 2022. [En línea]. Available: https://aprendeconalf.es/docencia/python/manual/numpy/. [Último acceso: 29 08 2023].

[13] «Statsmodels,» Statsmodels.org, 2023. [En línea]. Available: https://www.statsmodels.org/stable/index.html. [Último acceso: 29 08 2023].

[14] «Modelos de aprendizaje automático: diagnosticando su rendimiento,» 24 12 20219. [En línea]. Available: https://sitiobigdata.com/2019/12/24/una-introduccion-suave-a-las-curvas-de-aprendizaje-para-diagnosticar-el-rendimiento-del-modelo-de-aprendizaje-automatico. [Último acceso: 29 08 2023].

[15] «MODELO ARIMA (P, D, Q)(P, D, Q) S.,» [En línea]. Available: https://www.estadistica.net/ECONOMETRIA/SERIES-TEMPORALES/modelo-arima.pdf. [Último acceso: 29 08 2023].

[16] M. P. G. Casimiro, «Análisis de series temporales:,» [En línea]. Available: https://addi.ehu.es/bitstream/handle/10810/12492/04-09gon.pdf. [Último acceso: 29 08 2023].

[17] «Series temporales,» Catalán C., [En línea]. Available: http://humanidades.cchs.csic.es/cchs/web_UAE/tutoriales/PDF/SeriesTemporales.pdf. [Último acceso: 29 08 2023].

[18] «TensorFlow LSTM,» Keras, 2023. [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM. [Último acceso: 29 08 2023].

[19] «TensorFlow LSTM What is TensorFlow?,» 2021. [En línea]. Available: https://www.educba.com/tensorflow-lstm. [Último acceso: 29 08 2023].

[20] A. Sánchez, «La libreria Pandas,» Aprende con Alf., 2022. [En línea]. Available: https://aprendeconalf.es/docencia/python/manual/pandas/. [Último acceso: 29 08 2023].

[21] «Matplotlib-Visualization with Pyton,» Matplotlib, 2023. [En línea]. Available: https://matplotlib.org. [Último acceso: 29 08 2023].