Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Análisis y aplicación de métodos y técnicas de previsión de ventas en Python

Tutor: Pablo Aparicio Ruíz
Autor: Francisco Javier Carbonell Castillo

Dpto. Organización y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Análisis y aplicación de métodos y técnicas de previsión de ventas en Python

Autor:
Francisco Javier Carbonell Castillo

Tutor:
Pablo Aparicio Ruíz
Profesor Titular de Universidad

Dpto. Organización y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Análisis y aplicación de métodos y técnicas de previsión de ventas en Python

Autor: Francisco Javier Carbonell Castillo

Tutor: Pablo Aparicio Ruíz

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Agradecer en primer lugar a mi tutor, Pablo, por su ayuda y paciencia en tutorías y sobre todo por el tiempo que me ha dedicado en cada tutoría que me brindó, agradecer también su disposición e interés en querer ayudarme en cada tutoría que le solicitaba. Agradecer a la Escuela y a todo el conjunto de profesores por su dedicación en mi formación.

Agradecer como no a mi familia, que han sido mi sostén en estos años de carrera y este último año me han demostrado la importancia de estar unidos, a mis padres, Antonio y Conchi por facilitarme tanto la vida y hacérmela sencilla. A mi hermano, Antonio Miguel, por apoyarme y ayudarme.

Por último, a los alumnos que puedan darle utilidad a este trabajo, a los cuales ojalá les sirva para aprender más sobre Python y la programación orientada al análisis de datos.

Francisco Javier Carbonell Castillo

Sevilla, 2024

Resumen

El objetivo del proyecto es el análisis y aplicación de los métodos de previsión a un caso de estudio obtenido de un problema presentado por la comunidad científica, así como la utilización de un entorno para la exposición de los métodos de resolución de previsión de la demanda en un entorno que podría facilitar la exposición de estos casos a nivel docente.

En este trabajo se presenta la resolución de métodos de previsión de demanda mediante el lenguaje de programación Python, para que de una forma más interactiva los usuarios puedan ejecutar los bloques y analizar la solución en un entorno de desarrollo más ameno para el aprendizaje, sin necesidad de desarrollar código.

El presente trabajo está desarrollado en el entorno de Colab de Google, que permite ejecutar código sin configuración previa. Dicho entorno está alojado en Jupyter Notebook, por lo que los archivos generados son ejecutables también en Jupyter y otras plataformas.

Los datos escogidos en el presente proyecto están descargados de Kaggle, una subsidaria de Google, donde se encuentra una comunidad de científicos de datos que resuelven problemas reales de empresas, las cuales recurren a Kaggle para encontrar solución a sus problemas.

Al principio se presenta en Python como cargar el dataframe donde el usuario si quisiera podría escoger tanto el supermercado como el número de artículo que desease, teniendo así mediante el código desarrollado la oportunidad de analizar el mismo producto en varias tiendas, o varios artículos en una misma tienda. Conforme se avance y se vaya ejecutando código y eligiendo diferentes parámetros se podrán visualizar resultados y gráficas de forma que se observará cómo evolucionan los métodos con sus respectivos ajustes y predicciones, a su vez el cálculo de los errores permitirá analizar los mejores modelos según que patrón se esté estudiando.

Abstract

The objective of the project is the analysis and application of forecasting methods to a case study derived from a problem presented by the scientific community, as well as the use of an environment for demonstrating demand forecasting resolution methods in a setting that could facilitate the presentation of these cases for educational purposes.

This work presents the resolution of demand forecasting methods using the Python programming language, so that users can interactively execute the blocks and analyze the solution in a more user-friendly development environment for learning, without the need to develop code.

The present work is developed in Google Colab, an environment that allows code execution without prior configuration. This environment is hosted on Jupyter Notebook, making the generated files executable in Jupyter and other platforms as well.

The data selected for this project is downloaded from Kaggle, a subsidiary of Google, which hosts a community of data scientists who solve real-world problems for companies that turn to Kaggle for solutions.

Initially, it is shown in Python how to load the dataframe, where the user could choose both the supermarket and the item number they wish to analyze. This allows the opportunity to analyze the same product in various stores or various items in the same store using the developed code. As the code is executed and different parameters are chosen, results and graphs can be visualized, showing how the methods evolve with their respective adjustments and predictions. Additionally, the calculation of errors will enable the analysis of the best models according to the pattern being studied.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 OBJETIVOS Y ALCANCE

Desarrollar un estudio basado en un caso práctico de previsión de la demanda, el caso recrea los datos de una cadena de supermercados latinoamericana, el trabajo podría servir de recurso educativo para estudiantes que analizan la previsión de la demanda mediante el lenguaje de programación Python sin ser el objetivo de este. Estableciendo un orden claro en la consecución de los métodos para que la adquisición de conceptos sea lo más ordenada posible.

El proyecto se centrará en el análisis de las ventas de una cadena de supermercados específica en un período de tiempo determinado, estableciendo como ejemplos una tienda y un producto comestible no perecedero. Se modificarán ciertos valores de venta para exponer de manera más práctica y comprensiva los distintos modelos de previsión según el patrón que se esté estudiando. La tienda sobre la que se extraerán los datos será de una zona costera de Ecuador donde debido a ciertas épocas del año puede aumentar el turismo, siendo en estos periodos de tiempo importante una buena gestión de stock para mantener el inventario en unas condiciones lo más idóneas posibles.

El proyecto se expondrá con los disintos patrones de demanda existentes, que son, patrones de demanda constante, patrones de demanda con tendencia y patrones con estacionalidad.

Además, el análisis se centrará en encontrar los mejores valores de los parámetros de cada modelo para posteriormente escoger los modelos que mejores métricas obtengan. Estas métricas se basarán en errores que se calcularán también mediante Python implementando una función.

Se usarán recursos con las funciones existentes de Python para ajustar los modelos y se complementarán con desarrollos propios cuando sea necesario desarrollar en el lenguaje usado un código para ello. También se usarán los distintos métodos y funciones existentes para conocer mediante gráficas las componentes de la demanda y tener mejor idea de la tendencia o estacionalidad de los datos que se estén estudiando, aunque como método de apoyo en ocasiones se usarán métodos matemáticos para conocer la tendencia o estacionalidad de los datos.

2 CONTEXTO

El conjunto de datos a estudiar es de una cadena de supermercados minorista hispanoamericana, perteneciente a Corporación Favorita. Dicho conjunto de datos tiene un total de 54 tiendas y se establecerá a modo de ejemplo el de una sola tienda para mayor facilidad de estudio computacional. De esta forma se ahorra en tiempo y en recursos computacionales la búsqueda de artículos idóneos para la exposición de ciertos métodos, en su lugar se modificarán los ya existentes.

Los datos serán tratados y volcados en un dataframe para su manejo, esto permite descubrir cómo se hace el volcado cuando existen grandes cantidades de datos en un solo archivo, los actuales archivos existentes en las empresas cada vez con más datos así lo requieren, pues la actual 'moda' o 'necesidad' de recopilar cantidades masivas de datos hace que sean cada vez más complejos de tratar dado su peso.

El análisis de datos mediante aplicaciones como Excel es viable, pero lenguajes como Python permite desarrollar algoritmos mas complejos y acceder a una mayor potencia computacional, a su vez la limitación existente en herramientas ofimáticas como Excel para manejar grandes volúmenes de datos es cada vez más común, teniendo ciertos profesionales como data scientists un destacable mercado laboral dada la escasez de profesionales en el actual mercado.

Aunque los nuevos algoritmos de Inteligencia Artificial y Machine Learning son mucho más potentes que los métodos que en este trabajo se exponen, no dejan de ser estos métodos de previsión el origen y lo que se ha usado durante años para predicir el comportamiento en ventas de productos.

3 INTRODUCCIÓN

La previsión de la demanda se define como el proceso de estimar la cantidad de un producto o servicio que los clientes van a necesitar en un determinado periodo de tiempo. Esto implica la identificación de factores que afectan la demanda, la construcción de modelos cuantitativos o cualitativos para predecir la demanda futura y la evaluación continua de la exactitud de las predicciones.

El objetivo común de la previsión de la demanda es una menor cantidad de inventario, minimizar el coste de comprar y mantener, disminuir o evitar los costes indebidos y conseguir una previsión fiable para prever demandas futuras (Onieva Giménez, 2017).

El horizonte temporal de planificación según (Onieva Giménez, 2017) puede ser:

A. Muy corto plazo: 1-3 meses
B. Corto plazo: 3 meses - 1año
C. Largo plazo: 3-5 años
D. Muy largo plazo: más de 5 años

Es por ello que este TFG se centra en el corto plazo dado también la naturalidad y caducidad de los productos que se encuentran en una tienda de alimentación.

Como se puede extraer de (Chase, 2014) la previsión de las ventas de una empresa tiene un impacto directo sobre la gestión de la logística y de la cadena de suministros de la que forma parte. La previsión de las ventas es un requisito principal en la toma de decisiones del resto de procesos industriales. Las ventas generan un impacto directo en los niveles de existencias y en la gestión de las actividades de todos los actores que forman la cadena de logística. Con el fin de mejorar la eficiencia de una cadena de suministro es importante predecir con una precisión elevada el número de ventas, de dicho resultado se derivan los ingresos de la empresa y los costes, como el coste de stock, los costes derivados del riesgo de generar stock obsoleto o los efectos en el tiempo de abastecimiento del proceso industrial.

La previsión de la demanda puede ser estudiada cualitativa o cuantitativamente, el presente documento se centra en esta última aportando métodos diferentes mediante el análisis de series temporales.

4 ESTADO DEL ARTE

4.1. Clasificación de los métodos de previsión

En cuanto a la clasificación de métodos de previsión, se pueden diferenciar dos tipos principales: cualitativos y cuantitativos. Ambos tipos de métodos son importantes en el análisis de un producto, pero se utilizan en momentos diferentes del ciclo de vida del producto. Los métodos cualitativos son más útiles en las etapas iniciales del lanzamiento y crecimiento, donde los datos cuantitativos pueden ser limitados. Por otro lado, los métodos cuantitativos son más apropiados en las fases de estabilidad y madurez, cuando hay datos suficientes para realizar análisis estadísticos y predictivos precisos. Esto se ilustra en la Figura 4-1, que muestra cómo los diferentes métodos se aplican a lo largo del ciclo de vida de un producto (Onieva Giménez, 2017).

Figura 4-1 Curva ciclo de vida de un producto (Onieva Giménez, 2017).

4.1.1 Estimación subjetiva y métodos cualitativos

Según (Chase, 2014) las técnicas de pronóstico cualitativo se basan en el conocimiento y juicio de expertos, y suelen implicar procesos bien definidos para quienes participan en la predicción. Estas técnicas son especialmente útiles cuando se trata de productos nuevos o cuando hay poca experiencia previa en una región nueva. Entre las técnicas cualitativas más comunes se encuentran la investigación de mercado, los grupos de consenso, la analogía histórica y el método Delphi.

1. **Investigación de mercado**: Las empresas a menudo contratan a especialistas en investigación de mercado para recopilar datos a través de encuestas y entrevistas. Estos datos permiten conocer las preferencias de los consumidores, explorar nuevas ideas de productos y evaluar productos competitivos. Por ejemplo, es común recibir llamadas telefónicas o correos electrónicos solicitando información sobre hábitos de consumo y preferencias por ciertos productos.
2. **Grupos de consenso**: Esta técnica se basa en la idea de que la colaboración entre varias personas con diferentes perspectivas puede generar un pronóstico más preciso. Sin embargo, un problema común es que los empleados de niveles inferiores pueden sentirse intimidados por la jerarquía, lo que puede limitar su participación. Para superar esta limitación, se puede utilizar el método Delphi, que asegura la igualdad en la influencia de todas las opiniones al ocultar la identidad de los participantes.
3. **Analogía histórica**: Este método utiliza productos existentes como referencia para predecir la demanda de nuevos productos. Por ejemplo, la demanda de reproductores de videodiscos digitales puede estimarse a partir de la demanda histórica de VCRs. La analogía se basa en similitudes entre productos complementarios o sustitutos y su comportamiento en el mercado.
4. **Método Delphi**: Diseñado para evitar la influencia desproporcionada de individuos de niveles superiores, este método implica la recopilación de pronósticos de expertos de manera anónima. Un moderador recoge las respuestas, las resume y las redistribuye al grupo con nuevas preguntas. Este proceso se repite varias veces hasta que se alcanza un consenso. Generalmente, se necesitan alrededor de tres rondas para obtener resultados satisfactorios.

En resumen, las técnicas cualitativas de pronóstico permiten integrar conocimientos expertos y estructurar el proceso de toma de decisiones, proporcionando estimaciones valiosas, especialmente en contextos de incertidumbre o falta de datos históricos.

4.1.2 Estimación objetiva y métodos cuantitativos

Los métodos cuantitativos, generalmente aplicados durante las fases de crecimiento y estabilidad, utilizan datos numéricos históricos que se transforman mediante modelos o algoritmos para generar predicciones explícitas (Onieva Giménez, 2017).

Se pueden clasificar de la siguiente manera:

Métodos estadísticos de extrapolación (intrínsecos): Se basan en datos históricos de la demanda. Existen distintas técnicas de pronóstico según el comportamiento de la demanda sea nivelada, con tendencia lineal o con estacionalidad.

Métodos causales (extrínsecos): Asumen que la demanda está altamente correlacionada con ciertos factores externos al producto como publicidad, calidad o percepción de 'estatus' respecto a los competidores.

Método de Box-Jenkins: Desarrollado por George E. P. Box y Gwilym M. Jenkins, este enfoque se basa en la modelización de series temporales a través de modelos Arima (Autoregressive Integrated Moving Average), los cuales son capaces de capturar la estructura estocástica subyacente en los datos históricos (Box & Jenkins, 2016).

4.2. Métodos cuantitativos

En la disciplina de la previsión de demanda, los métodos cuantitativos desempeñan un papel fundamental al permitir el análisis y la modelización de datos históricos para predecir el comportamiento futuro de las variables de interés. Esta sección se centra en explorar algunos de los métodos cuantitativos más utilizados, comenzando con el análisis de series temporales y el proceso de elaboración de un pronóstico.

4.2.1 Análisis de series temporales

Según (Chase, 2014), los modelos de pronóstico de series de tiempo se utilizan para predecir el futuro utilizando información previa. Por ejemplo, se pueden usar datos de ventas anteriores para pronosticar las ventas futuras. Hay distintos modelos de series de tiempo que se aplican dependiendo del horizonte temporal y las características específicas de los datos.

Horizonte temporal y modelos utilizados: Se distinguen entre modelos de corto, medio y largo plazo. Estos términos son relativos, pero generalmente se refieren a menos de tres meses, de tres meses a dos años y más de dos años, respectivamente. Los pronósticos de corto plazo son útiles para decisiones tácticas, como la gestión de inventario, mientras que los de mediano plazo son útiles para planificar estrategias a seis meses a un año y medio. Los modelos de largo plazo identifican tendencias generales y cambios importantes.

Factores a considerar en la clección del modelo: La elección del modelo de pronóstico depende de varios factores, como el horizonte de tiempo, la disponibilidad y precisión de los datos, el presupuesto, y la disponibilidad de personal calificado. También se deben considerar la flexibilidad de la empresa y las consecuencias de un mal pronóstico, especialmente si se basan decisiones importantes en él, como inversiones de capital.

El análisis de series temporales es una técnica estadística utilizada para comprender y modelar el comportamiento de una variable a lo largo del tiempo. Este enfoque implica examinar patrones, tendencias y ciclos en los datos históricos para identificar regularidades y estructuras subyacentes que puedan ser utilizadas para predecir valores futuros.

Las previsiones tienen una importante característica a tener en cuenta y es que los pronósticos son siempre erróneos y por lo tanto se debe incluir un análisis de error (Onieva Giménez, 2017).

4.2.2 Proceso de elaboración de un pronóstico

El proceso de elaboración de un pronóstico comprende una serie de pasos que van desde la definición del problema hasta la evaluación y mejora continua del modelo de previsión.

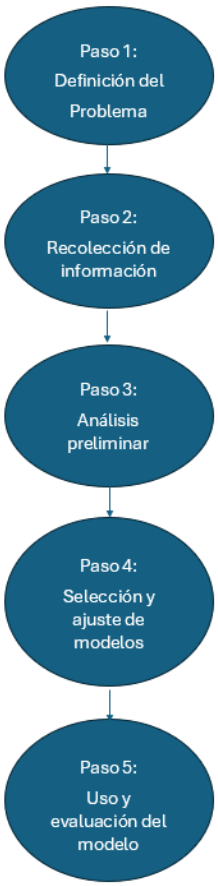

Figura 4-2 Pasos de elaboración de un pronóstico según (Hyndman, 2021).

La explicación detallada de cada paso según (Hyndman, 2021) sería:

Paso 1: Definición del problema

Es esencial comenzar por definir con precisión el problema de pronóstico que se aborda en el estudio. Este paso implica comprender a fondo cómo se utilizarán los pronósticos, quiénes serán los usuarios finales y cómo estos pronósticos se integrarán en la estructura organizativa. La colaboración con todas las partes interesadas, desde la recolección de datos hasta su aplicación en la planificación estratégica, es fundamental para garantizar un enfoque completo y relevante.

Paso 2: Recolección de información

En esta etapa, se requiere la recopilación de dos tipos principales de información: datos estadísticos y conocimiento experto acumulado por aquellos que participan en la recopilación y el uso de dichos datos. En ocasiones, los datos históricos disponibles pueden resultar insuficientes para desarrollar un modelo estadístico sólido, lo que puede requerir la aplicación de métodos de pronóstico basados en juicios expertos. Es importante valorar todos los datos históricos disponibles, ya que estos pueden proporcionar información valiosa para modelar incluso los cambios más sutiles en el sistema.

Paso 3: Análisis preliminar (exploratorio)

El análisis preliminar se inicia con la visualización y exploración de los datos para identificar patrones, tendencias, estacionalidades y posibles valores atípicos. Este análisis ayuda a comprender la relación entre las diferentes variables y a detectar posibles áreas de interés que requieran una investigación más profunda. Se pueden emplear diversas herramientas y técnicas para llevar a cabo este análisis preliminar como se expone en este trabajo.

Paso 4: Selección y ajuste de modelos

La selección del modelo más adecuado depende en gran medida de la disponibilidad y la calidad de los datos históricos, así como de la naturaleza de la relación entre la variable a pronosticar y las variables explicativas disponibles. Es común comparar varios modelos potenciales y evaluar su rendimiento relativo antes de seleccionar el más apropiado.

Paso 5: Uso y evaluación del modelo de pronóstico

Una vez seleccionado y ajustado el modelo de pronóstico, se procede a su implementación y uso para generar pronósticos futuros. La evaluación del rendimiento del modelo se realiza mediante la comparación de los pronósticos generados con los datos reales disponibles posteriormente. Se pueden emplear diversas métricas de evaluación para determinar la precisión del modelo y su capacidad para generar pronósticos útiles y confiables.

Este proceso implica la selección y aplicación de métodos adecuados, así como el estudio del error con datos reales y predichos pudiendo así modificar las variables de entrada de cada método, consiguiendo una mejora para los futuros valores predichos.

Algebraicamente una previsión constaría de lo siguiente:

$$Previsión = Patrón + aleatoriedad \tag{1}$$

La componente aleatoria es común en todas las estructuras de demanda, independientemente de si presentan nivelación, tendencia, estacionalidad u otros patrones específicos.

4.2.3 Componentes de una serie temporal

Una serie temporal puede descomponerse en varios componentes distintos, cada uno de los cuales contribuye de manera única al comportamiento general de la serie. Estos componentes incluyen el nivel, la tendencia, la estacionalidad, el ciclo, el ruido y las variaciones accidentales. La comprensión de estos componentes es fundamental para modelar y pronosticar con precisión el comportamiento de la serie temporal. (Chase, 2014)

Los componentes de la demanda según (Sánchez Fernández, 2004) son:

1. Tendencia: La tendencia se puede definir, de manera amplia, como la componente que refleja el comportamiento a largo plazo de una serie temporal. Para identificarla, es crucial que la serie cuente con un número significativo de observaciones a lo largo de muchos años, lo cual permite determinar si la serie sigue una ley de crecimiento, decrecimiento o estabilidad a largo plazo. Este comportamiento tendencial puede adoptar diferentes perfiles, como lineal, exponencial, parabólico, logístico, entre otros.
2. Estacionalidad: Las variaciones estacionales son movimientos periódicos que se repiten regularmente en la serie. Estas variaciones suelen estar influenciadas por factores climáticos (como producción o turismo) o por la organización del tiempo (como los días de la semana que afectan el comportamiento de ciertas series). Generalmente, la periodicidad es anual, aunque puede ser mensual, semanal o incluso diaria
3. Ciclos: Esta componente tiene un carácter económico marcado, ya que resulta de la alternancia de fases expansivas y recesivas de la economía. Son movimientos a medio plazo, que superan el año, y se repiten casi periódicamente, aunque no son tan regulares como las variaciones estacionales. La identificación de esta componente puede ser compleja debido a la posible superposición de ciclos de diferentes periodos o amplitudes. La amplitud se refiere a la duración total de un ciclo.
4. Variaciones Accidentales: Esta componente no sigue ningún patrón específico y resulta de factores fortuitos o aleatorios que afectan de manera aislada y no permanente a una serie.

De (Cruz Fernández, 2018) se pueden extraer las siguientes gráficas de las diferentes demandas existentes:

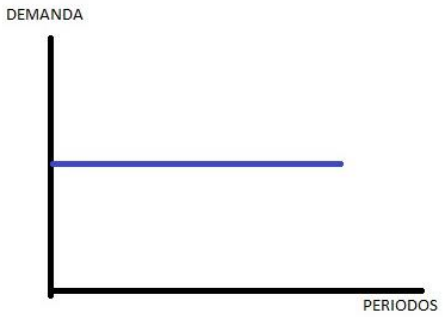

Figura 4-3 Demanda constante, basado en (Cruz Fernández, 2018).

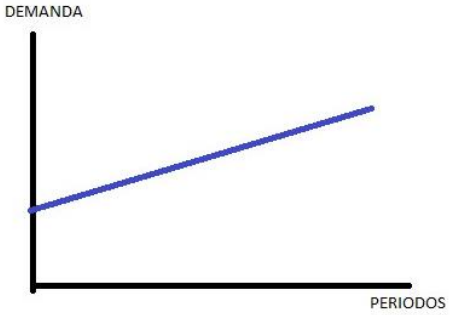

Figura 4-4 Demanda con tendencia, basado en (Cruz Fernández, 2018).

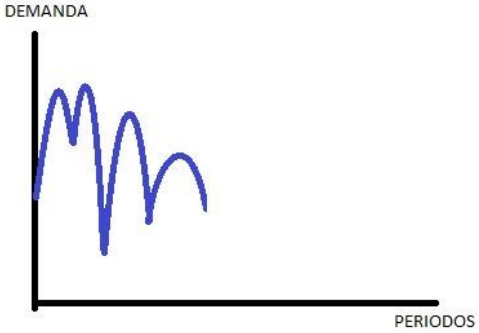

Figura 4-5 Demanda estacional, basado en (Cruz Fernández, 2018).

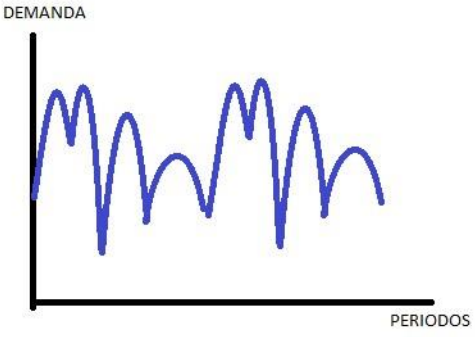

Figura 4-6 Demanda cíclica, basado en (Cruz Fernández, 2018).

Es común analizar patrones fomados por componentes donde tendencia y estacionalidad estén unidos, es fundamental comprender cómo interactúan estos componentes entre sí. En este contexto, se analizan dos tipos de variación estacional: aditiva y multiplicativa.

1. Variación Estacional Aditiva

En la variación estacional aditiva, la cantidad estacional es una constante independiente de la tendencia o la cantidad promedio. Esto significa que el efecto estacional se suma directamente a la tendencia para formar el pronóstico completo.

Pronóstico que incluye tendencia y estacional = Tendencia + Estacional

En Figura 4-7 se observa una tendencia en aumento con cantidades estacionales constantes, es decir, la estacionalidad se suma a la tendencia de manera constante a lo largo del tiempo.

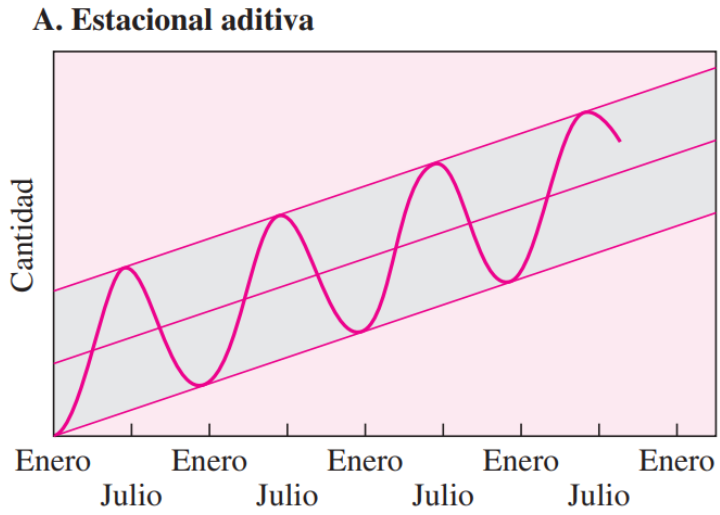

Figura 4-7 Demanda estacional Aditiva (Chase, 2014).

2. Variación Estacional Multiplicativa

Por otro lado, en la variación estacional multiplicativa, la tendencia se multiplica por los factores estacionales. En este caso, el efecto estacional se ajusta de acuerdo con la magnitud de la tendencia.

Pronóstico que incluye tendencia y estacional = Tendencia × Factor estacional

La Figura 4-8 de la variación estacional multiplicativa se observa cómo la estacionalidad aumenta con la tendencia, ya que su tamaño está directamente relacionado con esta última.

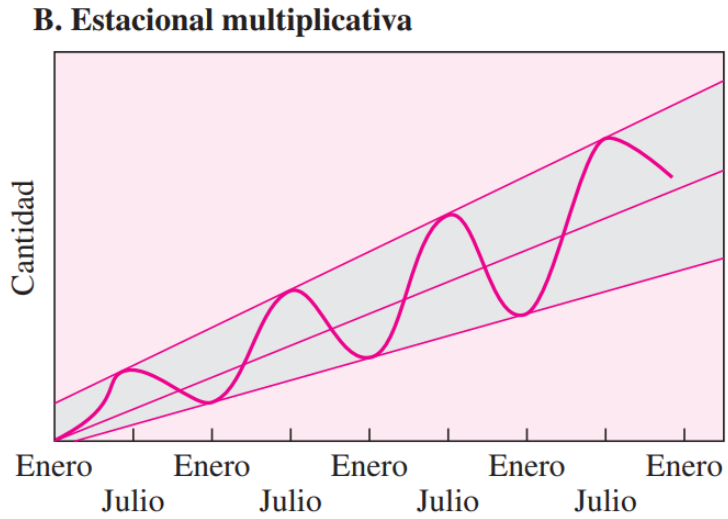

Figura 4-8 Demanda Estacional Multiplicativa (Chase, 2014).

En la práctica, la variación estacional multiplicativa es más común y establece que cuanto mayor sea la cantidad básica pronosticada, mayor será la variación estacional esperada a su alrededor. (Chase, 2014)

Factor Estacional

El factor estacional, también conocido como índice estacional, representa la cantidad de corrección necesaria en una serie temporal para ajustarse a la estacionalidad del año. Este factor se asocia generalmente con un período del año caracterizado por alguna actividad particular. A menudo, se utiliza el término "cíclico" para indicar que estos periodos no se limitan a la repetición anual de actividades. (Chase, 2014)

La descomposición de series temporales con efectos estacionales y de tendencia requiere comprender cómo estos componentes interactúan entre sí y cómo se pueden modelar tanto aditiva como multiplicativamente para lograr pronósticos precisos.

4.2.4 Métodos de previsión de series de tiempo

Las fórmulas de los métodos que a continuación se exponen se extraen de (Onieva Giménez, 2017) salvo algunas fórmulas en las que se mencionará al autor.

La notación a usar para cada previsión será la siguiente:

$$\widehat{D}_{T+\tau}(T) \tag{2}$$

Donde $\widehat{D}$ es la previsión, T es el periodo del último dato recogido, es decir en el que se realiza la previsión y τes el periodo a predecir, periodos futuros.

4.2.4.1 Métodos de previsión para patrones de demanda constante

Los patrones de demanda constante están formados por una estructura constante (D_t) y un error (ε_t).

$$D_t = D + \varepsilon_t \tag{3}$$

4.2.4.1.1 Medias Móviles

La fórmula general para T observaciones de una media móvil, siendo T el último periodo, es decir, $D_1,\ D_2\ , \dots , D_T$

$$\widehat{D}_{T+\tau}\ (T) = \frac{1}{T}\sum_{t=1}^{T} D_t \tag{4}$$

Se consideran igual de importante todas las observaciones, es por ello que ante un cambio de patrón no es un buen estimador de la demanda futura.

4.2.4.1.1.1 Medias Móviles Simples

Como se expone en (Hanke, 2010) al utilizar una media móvil como estimador si el analista está más interesado en las últimas N observaciones, la fórmula anteriormente expuesta sería:

$$\widehat{D}_{T+\tau}\ (T) = \frac{1}{N}\sum_{t=T-N+1}^{T} D_t \tag{5}$$

4.2.4.1.1.2 Medias Móviles ponderadas

Si además se quiere dar más importancia a las más recientes se pueden ponderar las observaciones añadiéndose más pesos a las últimas observaciones tal y como se expone en (Onieva Giménez, 2017).

$$\widehat{D}_{T+\tau}\ (T) = \frac{1}{N}\sum_{t=T-N+1}^{T} D_t c_t \tag{6}$$

$$\sum_{t=T-N+1}^{T} c_t = N \; siendo \; c_t \; el \; peso \; del \; periodo \; t \tag{7}$$

4.2.4.1.2 Ajuste Exponencial

El ajuste exponencial se basa en obtener las nuevas estimaciones modificándolas en un porcentaje del error de previsión que se define como:

$$e(T) = D_T - \widehat{D}(T-1) \tag{8}$$

$$S_T = \alpha D_T + (1-\alpha) S_{T-1} \tag{9}$$

Donde $\widehat{D}(T-1)$ es la estimación de D realizada en T - 1

Siendo la demanda alisada S_T y la constante de alisamiento α que se encuentra entre los valores cero y uno.

$$S_T = \widehat{D}_{T+\tau}(T) \tag{10}$$

Se denomina con 'S' a la estimación de cualquier periodo futuro.

El valor inicial se suele elegir como:

$$\widehat{D}_{0+\tau}(T=0) = S_0 = \frac{1}{T}\sum_{t=1}^{T} D_t \tag{11}$$

Según (Chase, 2014), la suavización exponencial requiere elegir un valor de alfa (α) entre 0 y 1. Para demandas estables, como electricidad o alimentos, se recomienda un alfa bajo para reducir los efectos de variaciones aleatorias. Para demandas cambiantes, como productos de moda, es mejor un alfa alto para adaptarse rápidamente. Determinar la mejor alfa puede ser laborioso y debe ajustarse frecuentemente debido a cambios en la demanda. Por eso, se necesitan métodos automáticos para actualizar alfa. Dos estrategias son:

Valores predeterminados de alfa: Se mide el error entre pronóstico y demanda real. Un error grande usa un alfa de 0.8; un error pequeño, un alfa de 0.2.

Cálculo dinámico de alfa: La constante de rastreo alfa se calcula como el error real suavizado dividido por el error absoluto suavizado, ajustándose entre 0 y 1 según cambie la demanda.

4.2.4.2 Métodos de previsión para patrones de demanda con tendencia

En el análisis y pronóstico de series temporales, es fundamental considerar los patrones de demanda que presentan una tendencia. Una tendencia puede manifestarse como un incremento o un decremento sistemático en los datos a lo largo del tiempo. Los métodos de previsión que capturan estos patrones son esenciales para generar pronósticos precisos y fiables en contextos donde la demanda no es constante.

$$D_t = D + pt + \varepsilon_t \tag{12}$$

Los patrones de demanda con tendencia están formados por una estructura constante (D_t), una pendiente (p) y un error (ε_t).

4.2.4.2.1 Medias Móviles Dobles

Las medias móviles dobles también llamada promedio móvil doble son una extensión del método de medias móviles simples, diseñadas para abordar las series temporales con tendencia. Este método aplica una media móvil a la serie original y luego una segunda media móvil a la serie suavizada resultante. El objetivo es suavizar tanto las fluctuaciones aleatorias como las variaciones estacionales, resaltando la tendencia subyacente en los datos (Hanke, 2010).

El primer estimador de la media móvil doble al final del periodo T en un rango de N periodos es:

$$M_T = \frac{1}{N} \sum_{t=T-N+1}^{T} D_t \tag{13}$$

El segundo estimador de la media móvil doble es la media móvil del primer estimador:

$$M_T^{[2]} = \frac{1}{N} \sum_{t=T-N+1}^{T} M_t \tag{14}$$

Es por tanto la estimación de la observación en el periodo $T + \tau$:

$$\widehat{D}_{T+\tau}(T) = \widehat{D} + \tau\widehat{p_T} = 2M_T - M_T^{[2]} + \tau\left(\frac{2}{N-1}\left(M_T - M_T^{[2]}\right)\right) \tag{15}$$

4.2.4.2.2 Ajuste Exponencial Doble

El Ajuste Exponencial Doble es un método que se utiliza cuando los datos muestran una tendencia lineal. Este método emplea dos componentes de suavizamiento: una para el nivel y otra para la tendencia. Al dar más peso a las observaciones recientes, el ajuste exponencial doble es capaz de adaptarse rápidamente a los cambios en la tendencia, ofreciendo un pronóstico más reactivo y preciso.

Sabiendo que el primer estimador es:

$$S_T = \alpha D_T + (1-\alpha) S_{T-1} \tag{16}$$

Se expone el segundo estimador siendo:

$$S_T^{[2]} = \alpha S_T + (1-\alpha) S_{T-1}^{[2]} \tag{17}$$

Es por tanto la estimación de la observación en el periodo $T + \tau$:

$$\widehat{D}_{T+\tau}(T) = \widehat{D} + \tau\widehat{p_T} = 2S_T - S_T^{[2]} + \tau\frac{\alpha}{1-\alpha}\left(S_T - S_T^{[2]}\right) \tag{18}$$

4.2.4.2.3 Método de Holt

El método de Holt, también conocido como suavizamiento exponencial de Holt, es una extensión del ajuste exponencial simple que incorpora un componente de tendencia. Este método es adecuado para series temporales con tendencias lineales y proporciona una fórmula para actualizar tanto el nivel como la tendencia en cada periodo, haciendo uso de dos parámetros de suavizamiento (Hanke, 2010).

La fórmula de la serie temporal sigue siendo la misma salvo que se denota con P mayúscula a la pendiente.

$$D_t = D + tP_t + \varepsilon_t \tag{19}$$

El método de Holt está compuesto por dos ecuaciones, una para la ordenada y otra para la pendiente, y se usa cuando los métodos anteriores no consiguen prever la tendencia con anterioridad.

$$S_t = \alpha D_t + (1-\alpha)(S_{t-1} + P_{t-1}) \tag{20}$$

$$P_t = \beta(S_t - S_{t-1}) + (1-\beta)P_{t-1} \tag{21}$$

La previsión al final del periodo t es:

$$D_{t+\tau}(t) = S_t + \tau \cdot P_t \tag{22}$$

Las estimaciones de los parámetros iniciales pueden hacerse mediante medias móviles, aunque también es

común hacerlo mediante regresión lineal dado el conocimiento que pueda tenerse de la pendiente de la serie temporal.

4.2.4.2.4 Modelos Arima no-estacionales Arima(p,d,q)

Un modelo Arima, que significa 'Modelo Autorregresivo Integrado de Media Móvil' por sus siglas en inglés (autoregressive integrated moving average), es una herramienta estadística que se basa en la observación de variaciones y regresiones en conjuntos de datos para identificar patrones y realizar predicciones a futuro. Este tipo de modelo es dinámico y se aplica a series temporales, lo que implica que las proyecciones futuras se derivan de los datos históricos, en lugar de depender de variables independientes. Desarrollado en (Box & Jenkins, 2016) la metodología presentada en su obra, a menudo referida como el enfoque Box-Jenkins, sigue siendo una de las piedras angulares del análisis de series temporales.

Las distintas partes de un modelo Arima son las siguientes:

AR (Autoregressive): Esta parte del modelo indica que la variable que se está prediciendo depende linealmente de sus propios valores pasados y de los errores pasados. Donde el orden **p** significa que el valor actual se puede explicar como función lineal de **p** valores pasados. Se denota como **AR(p)** (Jihyun (Jenny) Kim, 2024).

MA (Moving Average): Esta parte del modelo indica que la variable depende linealmente de los errores pasados, como en un modelo de media móvil. Donde **q** indica hasta el número de rezagos de los errores pasados. Donde un rezago es una observación de la serie temporal correspondiente a un periodo anterior. En el contexto de los modelos de media móvil, los rezagos se refieren a los errores residuales de periodos anteriores que se utilizan para calcular el valor actual de la serie. En otras palabras, en un modelo MA(q), el valor actual de la serie se explica mediante una combinación lineal de los errores residuales de los últimos q periodos. (Jihyun (Jenny) Kim, 2024).

I (Integrated): Esta parte del modelo indica que los valores de la serie temporal han sido diferenciados al menos una vez para hacer que la serie sea estacionaria, es decir, para eliminar las tendencias y hacer que la media y la varianza sean constantes en el tiempo. Siendo **d** número de operaciones sucesivas para convertir la serie en estacionaria. Es por ello que para series ya estacionarias se denomina ARMA, pues es un caso particular de Arima. La diferencia entre ambos es que Arima incluye la diferenciación para hacer estacionaria una serie. La diferenciación implica calcular las diferencias entre los valores consecutivos de la serie temporal. Esto se hace restando el valor de un período de tiempo del valor del período de tiempo anterior. El resultado es una nueva serie de datos que representa los cambios entre períodos consecutivos en lugar de los valores reales de la serie (Jihyun (Jenny) Kim, 2024).

En (Hyndman, 2021) se presenta la estructura del modelo Arima con cada una de sus componentes, como se muestra a continuación:

En un modelo únicamente Autoregresivo (**AR**), se pronostica la variable de interés usando una combinación lineal de valores pasados de la variable. El término autorregresión indica que es una regresión de la variable contra sí misma.

Así, un modelo autorregresivo de orden p se puede escribir como:

$$y_t = c + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \cdots + \Phi_p y_{t-p} + \varepsilon_t \tag{23}$$

Donde y_t es el valor de la variable de interés en el tiempo t, c es una constante, $\Phi_1, \Phi_2, \ldots, \Phi_p$ son los coeficientes del modelo, $y_{t-1}, \ldots, y_{t-p}$ son los valores rezagados de la variable de interés y ε_t es el término de error en el tiempo t, que se asume como ruido blanco.

En un modelo únicamente de Media Móvil (**MA**) se utilizan errores de pronóstico pasados en un modelo similar a una regresión. Un modelo de media móvil de orden q se puede escribir como:

$$y_t = c + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_p \varepsilon_{t-q} + \varepsilon_t \tag{24}$$

Donde y_t es el valor de la variable de interés en el tiempo t, c es una constante, ε_t es el término de error en el tiempo t que se asume como ruido blanco, $\theta_1, \ldots, \theta_q$ son los coeficientes del modelo, $\varepsilon_{t-1}, \ldots, \varepsilon_{t-q}$ son los errores de pronóstico pasados.

Siendo por tanto la ecuación general de un modelo Arima no-estacional según (Hyndman, 2021):

$$y'_t = c + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \cdots + \Phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_p \varepsilon_{t-q} + \varepsilon_t \quad (25)$$

Donde y' sería la serie diferenciada, en este caso una sola vez.

Como expone (Jihyun (Jenny) Kim, 2024) si la serie no es estacionaria, se aplica diferenciación para hacerla estacionaria. Por tanto, una serie con tendencia o estacionalidad debe ser diferenciada.

Es por ello que una serie que en su forma original carece de tendencia y estacionalidad se puede considerar estacionaria. Es decir, si una serie temporal no tiene ni tendencia ni estacionalidad, entonces ya se considera estacionaria en su forma original.

4.2.4.3 Métodos de previsión para patrones de demanda con estacionalidad

Los patrones de demanda con estacionalidad son comunes en muchas series temporales y se caracterizan por fluctuaciones regulares que se repiten en intervalos específicos, como días, meses o años. Para abordar estos patrones y realizar previsiones precisas, existen varios métodos diseñados específicamente para manejar la estacionalidad. Los métodos de previsión con componente estacional se pueden representar por modelos aditivos o multiplicativos, dependiendo de cómo interactúan los componentes de la serie temporal.

Para multiplicativos tal y como se expone en (Onieva Giménez, 2017):

$$D_t = (D + pt)F_t + \varepsilon_t \quad (26)$$

Para aditivos, según (Hyndman, 2021) aunque en la fórmula se ha seguido manteniendo la nomenclatura que se expuso al principio y se le ha añadido el error:

$$D_t = D + pt + F_t + \varepsilon_t \quad (27)$$

En donde D es la componente permanente, p es componente con tendencia o pendiente, F_t es la componente estacional y ε_t componente aleatoria.

4.2.4.3.1 Holt-Winters

El modelo de Holt-Winters también conocido como suavizamiento exponencial triple, fue desarrollado por dos estadísticos: Charles Holt y Peter Winters.

- **Charles C. Holt** introdujo inicialmente los conceptos de suavizamiento exponencial en 1957, con un enfoque en el tratamiento de datos con tendencia (Holt, 1957).
- **Peter R. Winters** amplió el trabajo de Holt en 1960 para incluir componentes estacionales, resultando en el método que ahora conocemos como el método de Holt-Winters (Winters, 1960).

En (Hyndman, 2021) se presentan los modelos existentes para Holt-Winters en forma aditiva y en forma multiplicativa.

El modelo aditivo de Holt-Winters es un modelo de pronóstico que descompone una serie temporal en tres componentes: nivel, tendencia y estacionalidad. La forma componente para el método aditivo se expresa como:

$$\hat{y}_{t+h} = l_t + hb_t + s_{t+h-m(k+1)} \quad (28)$$

Donde $\hat{y}_{t+h}$ es el pronóstico para h periodos hacia adelante desde el tiempo t, l_t es el nivel en el tiempo t, b_t es la tendencia en el tiempo t, s_t es el componente estacional en el tiempo t, m es la cantidad de periodos en una temporada y k es la parte entera de $(h-1)/m$.

Las ecuaciones de nivel, tendencia y estacionalidad son:

$$l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \quad (29)$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1} \tag{30}$$

$$s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1 - \gamma)s_{t-m} \tag{31}$$

Donde y_t es el valor observado en el tiempo t, α es el parámetro de suavización para el nivel, β^* es el parámetro de suavización para la tendencia y γ es el parámetro de suavización para la estacionalidad.

El método multiplicativo de Holt-Winters se utiliza cuando la estacionalidad tiene un efecto multiplicativo en la serie temporal. La forma componente para el método multiplicativo es:

$$\hat{y}_{t+h} = (l_t + hb_t)s_{t+h-m(k+1)} \tag{32}$$

Las ecuaciones de nivel, tendencia y estacionalidad son:

$$l_t = \alpha \frac{y_t}{s_{t-m}} + (1 - \alpha)(l_{t-1} + b_{t-1}) \tag{33}$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1} \tag{34}$$

$$s_t = \gamma \frac{y_t}{l_{t-1} - b_{t-1}} + (1 - \gamma)s_{t-m} \tag{35}$$

4.2.4.3.2 Modelos Arima estacionales Arima (p, d, q) (P, D, Q), m

En el apartado 4.2.4.2.4 en el caso de que exista comportamiento de tendencia es necesario aplicar diferenciación en el método Arima, siendo de esta manera ***d*** distinto de cero. Para esta sección dado que el modelo es estacional sería necesario aplicar también diferenciación estacional, siendo ***D*** distinto a cero.

Al aplicar diferenciación a una serie temporal, el objetivo es eliminar gradualmente la tendencia y la estacionalidad presentes en los datos.

Una vez que se ha logrado la estacionariedad, se puede proceder a modelar la serie temporal con un modelo Arima, que incluirá términos para la componente autoregresiva (AR), la componente de media móvil (MA) y, en el caso de datos estacionalizados, la componente de estacionalidad (S), denominándose entonces SARIMA (Seasonal Autoregressive Integrated Moving Average), ó, Arima (p, d, q) (P, D, Q), m. Dónde p es el número de términos autorregresivos, d es el grado de diferenciación, q es el número de términos de media móvil, m es se refiere al período de cada temporada o estación y (P, D, Q) representa el (p, d, q) para la parte estacional de la serie temporal (Jihyun (Jenny) Kim, 2024).

Desarrollando el modelo no-estacional de (Hyndman, 2021), la fórmula para un modelo Arima estacional sería incluir los términos autoregresivos y de media móvil, es decir, P y Q, en la formula, quedando de la siguiente manera:

$$(1 - B)^d(1 - B^m)^D y_t = c + \sum_{n=1}^{p} \Phi_n y_{t-n} + \varepsilon_t + \sum_{n=1}^{q} \theta_n e_{(t-n)} + \sum_{n=1}^{P} \hat{\Phi}_n y_{t-mn} + \sum_{n=1}^{Q} \hat{\theta}_n e_{(t-mn)}$$

Donde:

Diferenciación no estacional $(1 - B)^d$:

- Elimina la tendencia de la serie temporal, hace estacionaria a la serie en tendencia.

Diferenciación estacional $(1 - B^m)^D$:

- Elimina la estacionalidad de la serie temporal, hace la serie estacionaria en estacionalidad.

Componente autorregresiva no estacional $\sum_{n=1}^{p} \Phi_n y_{t-n}$:

- Relaciona los valores pasados de la serie temporal diferenciada con el valor actual.

Componente de media móvil no estacional $\sum_{n=1}^{q} \theta_n e_{(t-n)}$:

- Relaciona los errores pasados con el valor actual.

Componente autorregresiva estacional $\sum_{n=1}^{P} \widehat{\Phi}_n y_{t-mn}$

- Relaciona los valores pasados de la serie temporal diferenciada con lag estacional con el valor actual.

Componente de media móvil estacional $\sum_{n=1}^{Q} \hat{\theta}_n e_{(t-mn)}$:

- Relaciona los errores pasados con lag estacional con el valor actual.

Error ε_t:

- Representa el término de error aleatorio en el tiempo t.

En (Hyndman, 2021) se presenta la notación "Backshift" que significa operador de retardo (también conocido como operador de desplazamiento hacia atrás) es una herramienta útil en la notación de series temporales, especialmente cuando se trabaja con retrasos en los datos $By_t = y_{t-1}$

Por tanto $y'_t = y_t - y_{t-1} = (1 - B)y_t$, y de forma general una diferencia de orden "d" sería $= (1 - B)^d y_t$, encaso de diferenciación estacional sería con "D".

La fórmula de un modelo SARIMA con diferenciación explícita muestra cómo se aplican las diferenciaciones no estacional y estacional a la serie temporal antes de ajustar las componentes autorregresivas y de media móvil. Esto permite capturar adecuadamente las tendencias y estacionalidades en los datos, proporcionando un modelo robusto para la previsión de series temporales complejas.

4.2.4.4 Errores aplicados a todos los métodos para su análisis

Se exponen a continuación los errores usados de (Hanke, 2010) para hacer el análisis en cada método existente, siendo D_t el valor de una serie de tiempo en t y $\widehat{D}_t$ el valor pronosticado.

- ME (Mean Error): Es el promedio de los errores (diferencias entre valores reales y valores pronosticados)

$$ME = \frac{1}{N}\sum_{t=1}^{N}(D_t - \widehat{D}_t) \quad (36)$$

- MAE (Mean Absolute Error): Promedio de los errores absolutos. Mide la precisión del pronóstico al promediar la magnitud de los errores, sin considerar su dirección.

$$MAE = \frac{1}{N}\sum_{t=1}^{N}|D_t - \widehat{D}_t| \quad (37)$$

- MPE (Mean Percentage Error): Promedio de los errores porcentuales. Detecta si el método de pronóstico está sesgado (si los pronósticos tienden a ser consistentemente altos o bajos).

$$MPE = \frac{100}{N}\sum_{t=1}^{N}\frac{(D_t - \widehat{D}_t)}{D_t} \quad (38)$$

- MAPE (Mean Absolute Percentage Error): Promedio de los errores absolutos expresados como porcentaje del valor real. Útil para comparar la precisión de diferentes técnicas o series, ya que es un porcentaje y no depende de las unidades de medida.

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\frac{|D_t - \widehat{D}_t|}{D_t} \quad (39)$$

- RMSE (Root Mean Square Error): Raíz cuadrada del MSE. Penaliza más los errores grandes y es útil

para evaluar la precisión global del modelo en las mismas unidades de la serie original.

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\frac{(D_t - \hat{D}_t)}{D_t}} \quad (40)$$

A continuación se presentan las formulas de correlación y minmax (Peña, 2005):

- CORR (Correlación) entre los valores reales y los valores pronosticados se calcula utilizando el coeficiente de correlación de Pearson. Este coeficiente mide la fuerza y la dirección de la relación lineal entre dos variables.

$$CORR = \frac{\Sigma_{t=1}^{N}(D_t - \bar{D}_t)(\hat{D}_t - \bar{\hat{D}}_t)}{\sqrt{\Sigma_{t=1}^{N}(D_t - \bar{D}_t)\,\Sigma_{t=1}^{N}(\hat{D}_t - \bar{\hat{D}}_t)}} \quad (41)$$

- MinMax: Evalúa el error relativo considerando los extremos de los valores observados, útil para identificar el peor error relativo en proporción al valor mínimo observado.

$$MinMax = \frac{max(|D_t - \hat{D}_t|)}{min(D_t)} \quad (42)$$

4.2.4.5 Gráficos ACF y PACF

Como se expone en (Hyndman, 2021), del gráfico de la Función de Autocorrelación (ACF) se extrae cómo los valores de una serie temporal están correlacionados con sus valores pasados a diferentes desfases (lags). Cuando una serie temporal presenta una tendencia, significa que hay un patrón persistente en el tiempo, ya sea creciente o decreciente. En estos casos:

- Las autocorrelaciones para desfases pequeños (lags cortos) tienden a ser grandes y positivas.
- Esto ocurre porque las observaciones cercanas en el tiempo tienden a tener valores similares.
- Como resultado, el gráfico ACF de una serie temporal con tendencia muestra valores positivos que disminuyen lentamente a medida que aumenta el desfase.

La estacionalidad en una serie temporal se refiere a patrones que se repiten a intervalos regulares, como estacionalidad mensual, trimestral, etc. En estos casos:

Las autocorrelaciones serán mayores en los desfases que corresponden a los periodos estacionales (multiplicando el periodo estacional por un número entero).

Esto significa que en un gráfico ACF, se observan picos significativos en intervalos regulares, correspondientes a estos periodos estacionales.

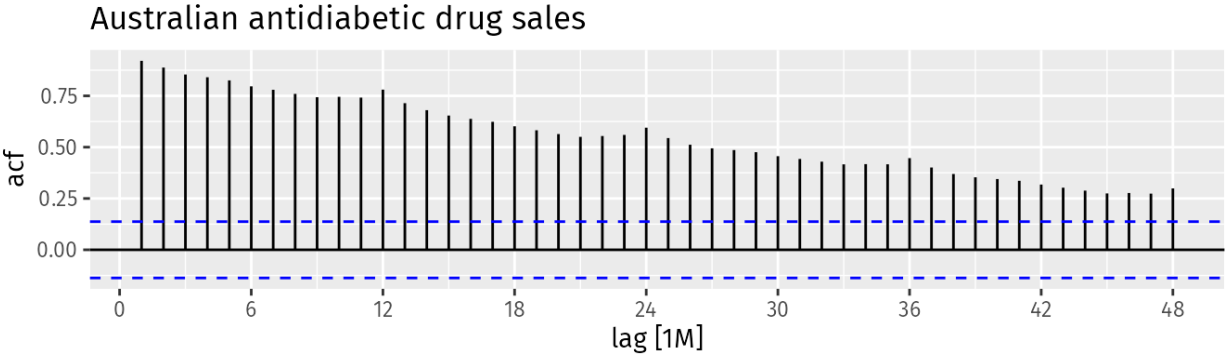

Figura 4-9 Gráfica ACF (Hyndman, 2021).

En la Figura 4-9 se presenta ambas características (tendencia y estacionalidad). Al observar el gráfico ACF de

esta serie:

- Se aprecia una disminución lenta y constante en las autocorrelaciones debido a la tendencia.
- Se observa una forma "escalonada" en el gráfico, indicando la presencia de estacionalidad.

El gráfico de la Función de Autocorrelación Parcial (PACF) es similar al ACF, pero proporciona una visión más refinada de la relación entre una observación y sus rezagos pasados, eliminando el efecto de las correlaciones indirectas a través de los rezagos intermedios.

También de (Hyndman, 2021) se puede extraer cómo interpretar los gráficos para obtener los mejores valores para los modelos Arima.

Para el caso no-estacional:

- Arima (p, d,0) (p, d, 0): Si el ACF muestra una disminución exponencial o sinusoidal y la PACF tiene un pico significativo en el rezago p, entonces p podría ser apropiado.
- Arima (0, d, q) (0, d, q): Si la PACF muestra una disminución exponencial o sinusoidal y el ACF tiene un pico significativo en el rezago q, entonces q podría ser apropiado.

Para el caso estacional:

- Los valores P y Q indican el orden de los términos de autocorrelación (AR) y de media móvil (MA) estacionales, respectivamente. Por ejemplo, si P=1, un pico significativo en el lag estacional 12 en el PACF sugiere un término AR estacional de orden 1. Por el contrario, si Q=1, un pico significativo en el lag estacional 12 en el ACF sugiere un término MA estacional de orden 1.
- Los valores p y q indican el orden de los términos AR y MA no estacionales, respectivamente. Mismo procedimiento que en el caso no-estacional con solo tendencia.

4.2.4.6 Prueba de Dickey-Fuller Aumentada (ADF)

Teniendo en cuenta que para los modelos Arima se deben diferenciar las series, ya sean no-estacionales ‘d’ o estacionales ‘D y d’ y que dichos valores se alcanzan cuando la serie es estacionaria, se debe entonces saber cuando una serie es estacionaria y para ello se usa la prueba de Dickey-Fuller o Dickey-Fuller aumentada.

La prueba de Dickey-Fuller es una técnica comúnmente utilizada en econometría para evaluar la estacionariedad de una serie temporal. La estacionariedad es una propiedad importante en el análisis de series temporales, ya que implica que las propiedades estadísticas de la serie no cambian con el tiempo. La prueba de Dickey-Fuller evalúa la presencia de raíces unitarias en una serie temporal, lo que sugiere que la serie no es estacionaria, de (Gujarati & Porter, 2010) se puede extraer:

1. **P-value bajo (menor a 0.05):** En la prueba de Dickey-Fuller, el p-value representa la probabilidad de observar los resultados de la prueba si la hipótesis nula fuera verdadera. La hipótesis nula en este caso es que la serie temporal tiene una raíz unitaria, lo que implica que no es estacionaria. Un p-value bajo (por ejemplo, menor a 0.05) indica que hay suficiente evidencia para rechazar la hipótesis nula, lo que sugiere que la serie es estacionaria.
2. **ADF más negativo:** La estadística de la prueba de Dickey-Fuller aumentada (ADF) es una medida de la presencia de una raíz unitaria en una serie temporal. Cuando esta estadística es más negativa, indica una mayor evidencia en contra de la hipótesis nula de no estacionariedad. Por lo tanto, un valor más negativo de la estadística ADF sugiere una mayor probabilidad de que la serie sea estacionaria.

5 CASO DE ESTUDIO

El caso a resolver se centra en el área comercial de 'Corporación Favorita', un conglomerado formado por un conjunto de cadenas de supermercados y tiendas minoristas. Este estudio se focaliza exclusivamente en una de sus cadenas.

5.1. Descripción del caso

La cadena denominada 'Supermaxi'(*Corporación Favorita*, 2024) se encuentra en Salinas, un cantón de la provincia de Santa Elena, ubicada en la costa de Ecuador.

5.1.1 Contexto de la tienda

Supermaxi es la cadena de supermercados más famosa y extensa de Ecuador, ofreciendo una amplia gama de más de 45,000 ítems que incluyen productos de alimentación, artículos para el hogar y productos de limpieza. Esta cadena se caracteriza por ser un autoservicio, lo que permite a los clientes seleccionar y adquirir productos de manera directa (*Supermaxi*, 2024).

Importancia de la previsión de ventas:

La previsión de ventas en la tienda Supermaxi de Salinas es importante por varias razones:

- **Optimización de inventario**: Una previsión precisa permite mantener niveles óptimos de inventario, evitando tanto el exceso como la falta de productos. Esto es especialmente importante en una tienda con un catálogo tan amplio como Supermaxi.
- **Reducción de costos de almacenamiento**: Controlar adecuadamente el inventario ayuda a reducir los costos asociados al almacenamiento de productos, liberando recursos financieros para otras áreas del negocio.
- **Gestión de la cadena de suministro**: Una buena previsión de ventas facilita la planificación y gestión de la cadena de suministro, asegurando que los productos lleguen a tiempo y en las cantidades necesarias.
- **Adaptación a la variabilidad de la demanda**: Dada la ubicación cercana a la costa y la cantidad de personas no censadas que podrían alojarse en la ciudad durante las semanas de vacaciones, es fundamental tener un control de stock y de previsión alto para evitar roturas del mismo y satisfacer la demanda fluctuante.
- **Mejora de la satisfacción del cliente**: Al garantizar que los productos más demandados estén siempre disponibles, se mejora la experiencia del cliente, incrementando así la lealtad y las ventas.

Contexto geográfico y demográfico:

Geografía: Salinas es una ciudad costera conocida por sus playas y actividades turísticas. Su proximidad al mar influye en la demanda de ciertos productos, especialmente durante la temporada alta de turismo.

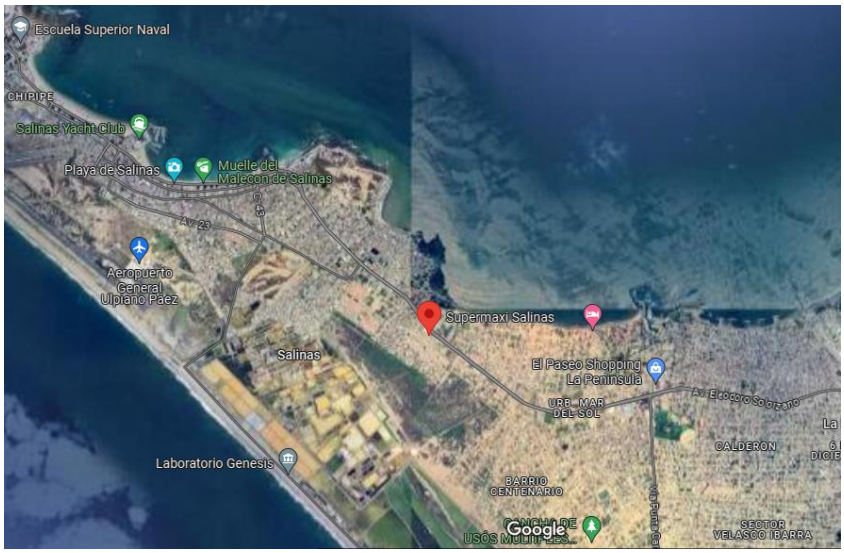

Figura 5-1 Ubicación del local en la ciudad de Salinas (*Google Maps*, 2024)

Figura 5-2 Vista del supermercado desde fuera.

- **Demografía**: Según datos (Gobierno de Ecuador, 2023), Salinas tiene una población cantonal de aproximadamente 86,000 habitantes. Sin embargo, durante las semanas de vacaciones y temporadas altas, la población puede aumentar significativamente debido al flujo de turistas.

Desafíos específicos

- **Variabilidad de la demanda**: La tienda debe lidiar con una demanda altamente variable, influenciada por factores estacionales y eventos locales.
- **Stock y promociones**: Es vital prever con precisión la demanda de productos que están en promoción y aquellos que son populares durante las vacaciones. Aunque esto se expone como un apartado de mejoras futuras dado que modelos como Arima permite incluir variables exógenas que tengan en cuenta esto.
- **Gestión del espacio**: Debido a la amplia variedad de productos, la tienda debe gestionar eficientemente el espacio en los estantes y el almacén, asegurando que los productos más importantes estén disponibles sin sobrecargar las instalaciones.

Figura 5-3 Vista del espacio interior del supermercado.

5.1.2 Producto analizado

El producto analizado en este caso es un artículo comestible no perecedero, lo que implica que no requiere almacenamiento en condiciones de refrigeración debido a su envasado y naturaleza de producto procesado. Este tipo de producto es común en los supermercados debido a su durabilidad y menor necesidad de una gestión rigurosa de la cadena de frío.

Comportamiento de la demanda:

Al observar el comportamiento de la demanda mensual del producto, se puede apreciar que las ventas nunca caen a cero. La Figura 7-1 muestra que, aunque existen picos de máxima demanda, no se identifica un componente de tendencia claro en el periodo analizado. Este comportamiento sugiere que el ítem escogido tiene una demanda constante a lo largo del tiempo, con variaciones que pueden deberse a factores externos como promociones o cambios estacionales menores.

Análisis de patrones de demanda

- **Demanda constante**: Para resolver el problema de previsión de ventas con patrones de demanda constante, se utilizarán los datos originales del producto. Estos datos reflejan una estabilidad en la demanda mensual, permitiendo aplicar métodos de previsión adecuados para escenarios con baja variabilidad. Este análisis permitirá determinar la precisión de los modelos en condiciones de estabilidad relativa.
- **Demanda con tendencia**: Para abordar patrones de demanda con tendencia, se modificarán los datos originales añadiendo una pendiente creciente. Esta modificación simulará un escenario donde el producto experimenta un aumento sostenido en la demanda a lo largo del tiempo. En la Figura 7-7 se puede observar cómo se comporta el producto bajo esta nueva configuración. Este análisis es crucial para entender cómo responderían los modelos de previsión si el producto comenzara a ganar popularidad o fuera objeto de campañas de marketing que incrementaran su demanda.
- **Demanda estacional**: Finalmente, para patrones de demanda con estacionalidad, se realizará una descomposición de los datos originales. Se tomarán los datos de los 12 meses del primer año como referencia para extraer la componente estacional. Esta componente se utilizará para generar nuevos datos que incorporen tanto una tendencia como estacionalidad. Este enfoque permitirá simular un escenario donde la demanda del producto varía significativamente en función de las estaciones o eventos recurrentes, como por ejemplo festividades. En la Figura 7-16 se puede observar cómo sería la demanda en ese caso.

5.1.3 Horizontes temporales del análisis

5.1.3.1 Frecuencia de datos y agrupación

El análisis se desarrolla utilizando un periodo de tres años y siete meses de datos históricos obtenidos de las ventas diarias de la tienda. Este extenso horizonte temporal permite un estudio exhaustivo y detallado de los patrones de demanda del producto a lo largo del tiempo.

El periodo específico de análisis abarca desde enero de 2013 hasta julio de 2017. Esta duración proporciona una base sólida para la identificación de tendencias, estacionalidades y otras variaciones significativas en la

demanda del producto. Analizar los datos a lo largo de varios años permite observar cómo las ventas responden a diferentes factores estacionales y eventos específicos, como promociones o festivos.

Los datos disponibles están registrados con una frecuencia diaria, lo que permite un análisis muy detallado de las ventas. La granularidad diaria es particularmente útil para detectar patrones de corto plazo y responder a eventos imprevistos. Sin embargo, para el propósito de este análisis y para simplificar la identificación de patrones estacionales, los datos diarios serán necesario reagruparlos en periodos mensuales.

Para convertir los datos diarios en datos mensuales, sería conveniente sumar las ventas diarias de cada mes. Este proceso de reagrupación mensual no solo facilita la identificación de patrones estacionales, sino que también suaviza las fluctuaciones diarias que podrían complicar el análisis a nivel macro.

5.1.3.2 División de datos para entrenamiento y prueba

Para asegurar la robustez y la validez del modelo de previsión, los datos se dividen en dos conjuntos: uno de entrenamiento y otro de pruebas. El 80% de los datos se utilizan para entrenar el modelo, mientras que el porcentaje restante como valores de predicción.

Entrenamiento del modelo: Los primeros tres años y siete meses de datos se emplean para entrenar los modelos de previsión. Durante este periodo, se ajustan los parámetros del modelo y se realiza la calibración necesaria para capturar los patrones de demanda. Con este porcentaje se analizan los errores estudiados que son los que sirven para calibrar el modelo.

Pruebas y validación: El periodo final de un año se utiliza para probar la capacidad predictiva del modelo y evaluar visualmente como se comporta ante los valores reales.

La elección de un horizonte temporal de cuatro años y siete meses proporciona un equilibrio adecuado entre la complejidad del modelo y la cantidad de datos necesarios para una previsión precisa. Un periodo menor podría no capturar adecuadamente las variaciones estacionales, mientras que un periodo excesivamente largo podría introducir ruido innecesario o cambios estructurales que compliquen el modelo.

La reagrupación de los datos en periodos mensuales permite una mejor identificación de patrones estacionales. Por ejemplo, se pueden observar aumentos en la demanda durante ciertos meses del año, que podrían corresponder a eventos recurrentes o temporadas específicas. Si se hiciese durante días o semanas existiría mayor cantidad de ruído. Además, la preparación de datos aporta beneficios como una mayor simplificación del análisis con datos mensuale, con los datos mensuales también es posible detectar los ciclos estacionales que puedan darse en un año.

5.2. Bases de datos del estudio

Los datos que se han usado para este estudio son extraídos de Kaggle (Kaggle, 2017) y forman parte de una competición que plantearon a la comunidad, el objetivo de la competición era plasmar el mejor código para predecir las ventas futuras de la compañía.

Los archivos principales para este estudio son los siguientes:

- **train.csv**: Datos de entrenamiento que incluyen las ventas unitarias diarias por tienda y producto.
- **stores.csv**: Metadatos de las tiendas, incluyendo ciudad, estado, tipo y cluster.
- **items.csv**: Metadatos de los productos, incluyendo familia, clase y perecibilidad.
- **transactions.csv**: Número de transacciones de ventas por fecha y tienda.
- **oil.csv**: Precio diario del petróleo.
- **holidays_events.csv**: Días festivos y eventos, con metadatos relevantes.

Dado que se quieren realizar previsiones con los datos existentes de un producto pertenecientes a una tienda, sólo se requieren los archivos de entrenamiento (**train.csv**) para extraer los datos de venta, datos de tiendas (**stores.csv**) para conocer la ubicación de la tienda que se pretende analizar, el estudio e información del producto (**items.csv**) que incluye el tipo de producto que es y si es perecedero o no.

5.3. Caracterización del problema

Se pretende predecir las ventas utilizando los modelos presentados en el estado del arte. El producto a estudiar es no perecedero y comestible, por lo que el horizonte temporal de estudio será mensual.

Para un análisis completo que contemple todos los patrones de comportamiento, es necesario modelar el producto considerando tanto la tendencia como la estacionalidad.

Los resultados de este análisis pueden integrarse en la estructura organizativa de la empresa para mejorar la gestión del inventario en el almacén y optimizar la compra de dicho producto. Estos resultados serán valiosos para las áreas responsables, ya que permiten tomar decisiones informadas sobre el producto. Así, la empresa puede evitar roturas de stock o escasez debido a una mala planificación de adquisiciones.

5.1.4 Recolección de la información

Los datos deben ser recolectados correctamente de los archivos. Se exponen a continuación los diferentes componentes de cada columna de los archivos.

El archivo **train.csv** posee las siguientes columnas:

- **id**: Es un identificador único para cada registro de venta. Este número es simplemente una clave primaria para referenciar cada línea de datos.
- **date**: Indica la fecha en que se realizó la venta. El formato de la fecha es '**YYYY-MM-DD**'.
- **store_nbr**: Es el número de la tienda en la que se realizó la venta. Cada tienda tiene un identificador unívoco en de la cadena.
- **item_nbr**: Es el número del artículo que se vendió. Cada producto tiene un identificador unívoco.
- **unit_sales**: Indica el número de unidades vendidas del artículo específico en la fecha y tienda dadas. Este número puede ser decimal, lo que indica que puede incluir fracciones de unidades, posiblemente debido a ventas a granel.
- **on_promotion**: Es un indicador que muestra si el artículo estaba en promoción en la fecha de la venta. Puede tener valores:
    (1) '**True**': Si el artículo estaba en promoción.
    (2) '**False**': Si el artículo no estaba en promoción.
    (3) Puede estar vacío si la información no está disponible o no aplica.

Estos datos se pueden usar para analizar el rendimiento de ventas, entender patrones de compra, evaluar la efectividad de promociones, y otros análisis relacionados con la gestión de inventarios y ventas. Para el caso de estudio que se expone tan solo será necesario usar la columna **date**, **store_nbr**, **item_nbr** y **unit_sales**.

Las columnas del archivo **stores.csv** son las siguientes:

- **store_nbr**: Es el número de la tienda, un identificador único para cada tienda en la cadena.
- **city**: Indica la ciudad en la que se encuentra la tienda.
- **state**: Indica la provincia o estado en el que se encuentra la tienda.
- **type**: Clasifica la tienda en diferentes tipos. Esta clasificación puede basarse en el tamaño, el formato, la oferta de productos u otros criterios específicos de la cadena de tiendas.
- **cluster**: Asigna la tienda a un grupo o clúster. Los clústeres se utilizan para agrupar tiendas con características similares, lo cual puede ser útil para análisis de ventas, marketing y gestión de inventarios.

Las columnas usadas de este archivo han sido **store_nbr**, **city** y **state**, fundamentalmente este archivo se ha usado para conocer la ubicación del supermercado a analizar.

El archivo **items.csv** también ha sido usado y posee los siguientes componentes:

- **item_nbr**: Es el número del artículo, un identificador único para cada producto.
- **family**: Indica la familia o categoría a la que pertenece el artículo. Esta clasificación agrupa productos similares o relacionados.
- **class**: Es una subcategoría o clase específica dentro de la familia del producto. Proporciona un nivel adicional de detalle en la clasificación del producto.
- **perishable**: Indica si el artículo es perecedero o no. Puede tener los valores:

  1: Si el artículo es perecedero.

  0: Si el artículo no es perecedero.

Para el caso de estudio que se expone, de (**stores.csv**) se analiza la tienda número 25 se encuentra en la ciudad de Salinas, perteneciente a Santa Elena, que es del tipo D y pertenece al grupo 1.

Se analizará el ítem 129758 que pertenece a la familia de Comestibles I de carácter no perecedero (**items.csv**).

La información de los datos diarios del ítem 129758 será agregada en datos mensuales, este horizonte temporal de estudio permitirá una toma de decisión de interés para la empresa.

Así ante días donde pueda ocurrir que haya habido un error en la toma o adquisición de datos por parte del personal, dado que la demanda agregada de un mes completo generará menos errores que la desagregada.

5.1.5 Análisis preliminar

En esta sección, se examinan los desafíos iniciales en la previsión de ventas de un producto comestible no perecedero en la tienda Supermaxi de Salinas. El análisis preliminar pretende identificar y comprender las posibles dificultades para adaptar adecuadamente los modelos de previsión a los datos disponibles.

Uno de los principales retos en la previsión de ventas es la variabilidad inherente en los datos de ventas del producto. Esta variabilidad puede manifestarse de diferentes maneras:

1. **Patrones estacionales**:
   - **Impacto del turismo**: Salinas, como ciudad costera, experimenta un aumento significativo en su población durante la temporada alta de turismo. Este incremento puede generar picos en la demanda de ciertos productos, especialmente aquellos relacionados con la alimentación.
   - **Festividades y feriados**: Las ventas pueden verse afectadas por eventos estacionales como festividades nacionales o locales, feriados y vacaciones escolares. Estos eventos pueden crear aumentos o disminuciones en la demanda que deben ser anticipados en los modelos de previsión.
2. **Tendencias a largo plazo**:
   - **Crecimiento o disminución sostenida**: Es posible que el producto analizado experimente una tendencia al alza o a la baja en la demanda a lo largo del tiempo. Identificar y modelar estas tendencias es crucial para realizar previsiones precisas.
   - **Cambios en el comportamiento del consumidor**: Variaciones en las preferencias y comportamientos de los consumidores, influenciadas por factores externos como cambios económicos o campañas de marketing.

Para abordar estos desafíos, se implementan varias estrategias que permiten ajustar y adaptar los datos a diferentes patrones de demanda. Estas estrategias se centran en modificar los datos originales para simular diferentes escenarios y evaluar la efectividad de los modelos de previsión en cada caso.

1. **Demanda constante**:
   - **Uso de datos originales**: Los datos originales del producto, que muestran una demanda relativamente constante, se utilizan para evaluar la capacidad de los modelos en escenarios de baja variabilidad. Esto proporciona una línea base para la comparación con otros escenarios más complejos.

2. **Demanda con tendencia**:
    - **Añadir pendiente a los datos**: Para simular un escenario con tendencia, se modifican los datos originales añadiendo una pendiente creciente. Esta simulación permite analizar cómo responderían los modelos de previsión si el producto experimentara un aumento sostenido en la demanda. Este enfoque es útil para prever cómo manejaría la tienda un incremento en la popularidad del producto.
3. **Demanda estacional**:
    - **Descomposición estacional**: Se realiza una descomposición de los datos originales para extraer la componente estacional, tomando como referencia los primeros 12 meses de datos. Esta componente estacional se utiliza para generar nuevos datos que incorporen tanto una tendencia como variaciones estacionales. Este método permite evaluar cómo los modelos de previsión pueden adaptarse a escenarios donde la demanda varía significativamente en función de las estaciones o eventos recurrentes.

Será necesario por tanto usar librerías de Python que sirvan para visualizar y obtener una vision de cómo se comportan los datos en cuanto a tendencia y estacionalidad, y para simular el comportamiento estacional tomar los datos de los primeros doce meses y amplificarlos junto a una tendencia para representar lo mejor posible un modelo multiplicativo.

5.1.6 Selección de variables y ajuste de modelos

En esta sección se detallará el proceso de selección y aplicación que debe existir para los distintos modelos de previsión en los diversos patrones de demanda. Una vez que se han obtenido los datos y determinado el tipo de patrón de demanda a estudiar, será necesario desarrollar las previsiones utilizando Python. En algunas ocasiones, este proceso será más sencillo debido a la disponibilidad de librerías específicas, mientras que, en otros casos, será necesario implementar el código que resuelva el problema paso a paso.

Modelos para demanda constante: Para abordar la demanda constante, se emplearán dos modelos principales utilizando librerías existentes:

- **Media móvil simple**: Este modelo requiere el ajuste de parámetros como el número de elementos utilizados para calcular la media y, en el caso de la media ponderada, los pesos asignados a cada elemento. La implementación en Python puede aprovechar funciones de librerías como **pandas** o **numpy** para manejar estos cálculos de manera eficiente.
- **Ajuste exponencial simple**: Este modelo se enfoca en el ajuste de un único parámetro, alfa. Las funciones de ajuste exponencial simple están disponibles en librerías como statsmodels, facilitando el proceso de implementación y ajuste del modelo.

Modelos para demanda con tendencia: Para los datos de ventas que muestran una tendencia clara, se desarrollarán y aplicarán varios modelos específicos:

- **Media móvil doble y ajuste exponencial doble**: Estos modelos deberán ser desarrollados desde cero en Python. Implicará calcular los estimadores y previsiones para cada periodo. En el caso del ajuste exponencial doble, será necesario incluir la recta de regresión para inicializar los estimadores. El código debe permitir la variación del valor de alfa para ajustar el modelo de manera precisa.
- **Método de Holt**: Este modelo será implementado utilizando funciones disponibles en la librería statsmodels. Los valores de alfa y beta serán incluidos como argumentos en la función de ajuste, permitiendo un ajuste dinámico del modelo según los datos de entrada.
- **Modelo Arima no estacional**: Para este modelo, es necesario verificar la estacionariedad de los datos. Se analizarán las gráficas de autocorrelación para determinar los mejores valores de los parámetros p, d y q. La función Arima de la librería pmdArima será utilizada para ajustar el modelo con estos valores óptimos.

Modelos para demanda estacional: Para los patrones de demanda estacional, se aplicarán los modelos Holt-Winters y Arima estacional:

- **Modelo Holt-Winters**: Este modelo será implementado en sus dos variantes: aditivo y multiplicativo.

Ambos requerirán ajustar los parámetros alfa, beta y gamma. Las funciones correspondientes en la librería statsmodels serán utilizadas para este propósito, permitiendo ajustar los datos de manera precisa según el tipo de estacionalidad presente.

- **Modelo Arima Estacional (SARIMA)**: Similar al modelo Arima no estacional, pero incluye parámetros adicionales para manejar la estacionalidad: P, D y Q, además del valor estacional m, que representa el número de periodos estacionales. Los mejores valores para estos parámetros se determinarán a través del análisis de gráficas de autocorrelación y autocorrelación parcial.

La selección y aplicación de modelos de previsión adecuados es esencial para capturar correctamente los patrones de demanda en la tienda Supermaxi de Salinas. Cada modelo ofrece beneficios específicos según la naturaleza de la demanda, desde la estabilidad proporcionada por las medias móviles simples hasta la complejidad de los modelos SARIMA, que pueden manejar tanto tendencias como estacionalidades. La combinación de estos modelos permite una previsión robusta y precisa, que es crucial para la optimización del inventario, la reducción de costos y la mejora de la satisfacción del cliente. Mediante el uso adecuado de estos modelos, se espera mejorar significativamente la gestión de inventarios, evitando tanto el exceso como la falta de productos, lo que se traduce en una operación más eficiente y rentable para la tienda.

5.1.7 Uso y evaluación de modelos

La evaluación de los modelos con los parámetros seleccionados requiere un correcto ajuste mediante la evaluación de errores expuestos en el estado del arte, por lo que será necesario interpretar el significado global de los resultados obtenidos y hacer un correcto análisis para la selección de los mejores valores de parámetros existentes en los modelos.

La evaluación de los modelos de previsión es un paso fundamental para determinar la efectividad y precisión de cada uno en predecir las ventas del producto analizado. Para determinar el modelo que ofrece la mejor precisión para el producto en la tienda Supermaxi de Salinas, se comparan los resultados de los diferentes modelos aplicados, cada uno de estos modelos se ajusta y evalúa utilizando los datos de entrenamiento, las métricas de evaluación (ME, MAE, MPE, MAPE, RMSE, CORR y MinMax) se calculan para cada modelo y se presentarán en una tabla comparativa. Una vez que se tengan las tablas de errores con los diferentes modelos y sus resultados obtenidos en cada error, se realizará un análisis detallado para escoger el mejor modelo. Este análisis se basará en las métricas mencionadas y en la capacidad del modelo para adaptarse a las características específicas del patrón de demanda del producto.

El análisis debe tener en cuenta:

- **Precisión**: Modelos con menor MAE, MAPE y RMSE son preferidos.
- **Sesgo**: Modelos con ME cercano a cero indican menos sesgo.
- **Consistencia**: Modelos con alta correlación (CORR) y bajo MinMax muestran una mejor relación con los datos reales y consistencia en las predicciones.

El uso correcto de los modelos en la práctica implica actualizar las variables y recalibrar los modelos de manera periódica, idealmente cada mes, para asegurar una mejora continua en la precisión de las previsiones. Esto permite:

- **Evaluar y ajustar**: Revisar y ajustar los modelos con nuevos datos para mantener su relevancia y precisión.
- **Detectar cambios en el patrón de demanda**: Identificar y responder a cambios en el patrón de demanda que podrían afectar la precisión del modelo.
- **Optimizar el desempeño**: Implementar ajustes y mejoras basadas en los errores observados y la evolución del comportamiento del mercado.

La correcta evaluación y uso de los modelos de previsión es esencial para lograr una previsión precisa de las ventas del producto en Supermaxi de Salinas. Mediante un análisis riguroso y una actualización continua, se puede asegurar que el modelo seleccionado no solo sea el más adecuado inicialmente, sino que se mantenga preciso y relevante a lo largo del tiempo.

6 DESARROLLO

El desarrollo de este trabajo se ha hecho en el entorno de Google Colab, dicho entorno está basado en Jupyter Notebook lo que permitirá descargar los archivos y usarlos en otro entorno si es lo que se prefiere por motivos académicos.

6.1. Herramientas e implementación

En este trabajo, se emplea el lenguaje de programación Python, conocido por su versatilidad y facilidad de uso en diversas aplicaciones informáticas. El archivo principal tiene nombre y extensión "*archivo.ipynb*", lo que indica que es un cuaderno de Jupyter, una plataforma interactiva que permite combinar código, visualizaciones y texto explicativo en un solo documento.

Las bibliotecas o librerías utilizadas en este proyecto son componentes fundamentales de Python que proporcionan funcionalidades específicas para el análisis de datos y la creación de modelos estadísticos. Entre las librerías más destacadas se encuentran:

- **Pandas**: Utilizada para la manipulación y análisis de datos estructurados, proporcionando estructuras de datos flexibles, herramientas para la limpieza y transformación de datos.
- **Matplotlib**: Una biblioteca de visualización que permite crear gráficos estáticos, animados e interactivos en Python, lo que facilita la representación visual de los datos y la comunicación de resultados.
- **Statsmodels**: Ofrece herramientas para el modelado estadístico y econométrico, incluyendo series temporales, modelos lineales, pruebas de hipótesis y así como otras herramientas.
- **Scikit-learn**: Una biblioteca de aprendizaje automático (machine learning) que proporciona algoritmos y herramientas para la clasificación, regresión, clustering y preprocesamiento de datos.

Además, se emplean otras librerías específicas para el análisis de series temporales, como **ExponentialSmoothing** para Holt-Winters, **Arima (AutoRegressive Integrated Moving Average)** y **pmdArima** para Arima, que son técnicas comúnmente utilizadas para modelar y predecir datos temporales.

Es importante destacar que estas librerías deben ser instaladas previamente para su uso, lo que se puede hacer en el entorno de Python mediante el gestor de paquetes **pip**.

Las librerías usadas para este trabajo son:

```
import warnings
import pandas as pd
from google.colab import drive
import copy
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
import numpy as np
from sklearn.metrics import mean_squared_error
from statsmodels.tsa.api import SimpleExpSmoothing
from scipy.optimize import minimize
import math
from statsmodels.tsa.api import Holt
from statsmodels.tsa.holtwinters import ExponentialSmoothing
pip install pmdArima
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
from pmdArima import Arima
from pmdArima import auto_Arima
import itertools
import statsmodels.api as sm
```

Es importante tener en cuenta que para usar las librerías anteriores previamente deben ser instaladas.

```
!pip install pmdArima

!pip install pandas

!pip install matplotlib

!pip install statsmodels

!pip install numpy

!pip install scikit-learn
```

6.2. Implementación de modelos de previsión

Previo al análisis de los datos contenidos en el archivo 'train.csv', se requiere montar la unidad de almacenamiento en la que se encuentra dicho archivo dentro del entorno de ejecución. Este procedimiento permite acceder a los datos almacenados en el archivo mencionado, facilitando así su carga y manipulación en el entorno de trabajo.

Es importante destacar que este paso preliminar garantiza la disponibilidad y accesibilidad de los datos requeridos para la implementación y evaluación de los modelos de previsión abordados en este apartado.

```
# Montaje de drive en el entorno
drive.mount('/content/drive')
# Filtrar los avisos relacionados con SettingWithCopyWarning
warnings.filterwarnings('ignore', category=pd.core.generic.SettingWithCopyWarning)
```

Se ha añadido también una línea de código para ignorar ciertos avisos que pueden dificultar la lectura.

A continuación se expone en Python como se han cargado los datos desde el archivo ‘train.csv’, en el código se escoge como parámetros la tienda y el número de elemento que se va estudiar para posteriormente hacer un filtrado y volcarlo todo en un datarfame.

```
# Definir el nombre del archivo y el número de tienda deseado
archivo = "/content/drive/MyDrive/pronostico/train.csv"  # Reemplaza "ruta_del_archivo.csv"
con la ruta real de tu archivo CSV

##########  Elegir datos de importación ###########
tienda_nbr = 25 # Representa el número de tienda, existen 54
# Item_nbr a filtrar
item_nbr_a_buscar = 129758 # Representa el número de producto de todos los existentes
#####################################

# Iterar sobre los elementos del archivo CSV y filtrar por tienda
chunks = pd.read_csv(archivo, chunksize=1000)  # Ajusta el tamaño de divisiones según sea
necesario
df_tienda = pd.concat(chunk[chunk['store_nbr'] == tienda_nbr] for chunk in chunks)

# Se representa las primeras filas del DataFrame de la tienda seleccionada
print(f"El dataframe del archivo csv cargado tiene los siguientes elementos (4 primeras
filas impresas)\n {df_tienda.head()}")
df=df_tienda.copy(deep=True) #mantener una copia de seguridad

def filtrar_por_item(item_nbr, dataframe):
```

```
    # Filtrar el DataFrame original por item_nbr
    df_filtrado = dataframe[dataframe['item_nbr'] == item_nbr]

    # Convertir la columna 'date' del DataFrame filtrado a tipo datetime
    df_filtrado['date'] = pd.to_datetime(df_filtrado['date'])

    # Crear un DataFrame con todas las fechas y unidades vendidas para el item_nbr
especificado
    todas_fechas = pd.date_range(start=df_filtrado['date'].min(),
end=df_filtrado['date'].max(), freq='D')
    nuevo_dataframe = pd.DataFrame({'date': todas_fechas})

    # Merge para unir las fechas y unidades vendidas
    nuevo_dataframe = pd.merge(nuevo_dataframe, df_filtrado[['date', 'unit_sales']],
on='date', how='left').fillna(0)

    # Calcular las unidades totales
    nuevo_dataframe['total_units'] = nuevo_dataframe['unit_sales'].cumsum()

    return nuevo_dataframe

# Obtener el nuevo DataFrame
nuevo_dataframe = filtrar_por_item(item_nbr_a_buscar, df)
nuevo_df=nuevo_dataframe.copy(deep=True) # Copia de seguridad

''' Solo se puede ejecutar una vez porque cuando 'date' es índice
    se vuelve inalterable'''
nuevo_df['date']=pd.to_datetime(nuevo_df['date'])

# Establece la columna 'date' como el índice del DataFrame
nuevo_df.set_index('date', inplace=True)

######### Se elige hasta que fecha se estudia #################

# Filtrar el DataFrame para mantener las filas anteriores a '2017-07-01'
fecha_limite = pd.to_datetime('2017-07-01')
df_filtrado = nuevo_df.loc[nuevo_df.index < fecha_limite]

############################################################

# Se reajusta por (trimestre, mes, semana), hace el dataframe más pequeño
'''
La funcion resample() usa :

W : frecuencia semanal
M : frecuencia de fin de mes
SM : frecuencia de fin de mes (15 y fin de mes)
Q : frecuencia de fin de trimestre

'''
resampled_data = df_filtrado.unit_sales.resample('M').sum()

 # Se crea un nuevo DataFrame con la serie resultante y su nombre
df_res = pd.DataFrame({'unit_sales': resampled_data})
df_res_copia = df_res.copy(deep=True)
```

El script expuesto en Python realiza lo siguiente:

1. Lee un archivo CSV que contiene datos de ventas, especialmente de diferentes tiendas.
2. Filtra los datos para seleccionar solo las filas correspondientes a una tienda específica (tienda_nbr).

3. Define una función llamada (filtrar_por_item()) que toma un número de artículo (item_nbr) y un DataFrame, y devuelve un nuevo DataFrame con las fechas y las unidades vendidas para ese artículo.
4. Invoca a la función (filtrar_por_item()) y filtra los datos para un artículo específico (item_nbr_a_buscar).
5. Utiliza la función resample de Pandas para reagrupar las ventas de unidades en intervalos mensuales ('M') y suma las unidades vendidas en cada intervalo.
6. Almacena los resultados en un nuevo DataFrame llamado df_res.

En resumen, este script carga datos de ventas, filtra y procesa los datos para un artículo específico y luego reagrupa las ventas mensuales para ese artículo en un nuevo DataFrame. Finalmente, los dataframe que nos servirán para estudiar serán df_res y df_res_copia que será una copia de seguridad del anterior.

La implementación se ha hecho teniendo en cuenta los diferentes patrones de demanda ya que existen diferentes modelos según la demanda sea constante, con tendencia o estacional.

En los métodos a continuación desarrollados se usa la demanda del artículo número 129758 de la tienda número 25. Dependiendo de qué método se use se modifican los valores para exponer mejor para qué tipo de demandas está hecho cada método dado que el objetivo del trabajo es que sea un caso de estudio para preveer la demanda con diferentes patrones.

Cada modelo de predicción en el entorno de programación estará formado por las siguientes partes:

- Librerías usadas
- Variables a escoger
- Funciones definidas
- Ejecución
- Errores
- Representación

Es necesario incluir aquí el código correspondiente a la función de errores denominada **metricas()**, dicha función toma como argumentos de entrada los valores reales de entrenamiento y las predicciones de entrenamiento, la función calcula los diferentes tipo de errores y los devuelve. Para ello son necesarias las librerías **numpy** que calcula los errores y **sklearn.metrics** para usar la función del error cuadrático medio. Posteriormente se usará en cada método dicha función para incluirla en un dataframe exclusivo para errores y poder visualizar los diferentes errores de cada método según el tipo de patrón de demanda.

```
def metricas(actual, forecast):
    """
    Función que analiza distintas métricas asociadas al error
    :param actual:
    :param forecast:
    :return:
    """
    # Identificar índices con valores NaN en actual y forecast
    nan_indices_actual = np.isnan(actual)
    nan_indices_forecast = np.isnan(forecast)

    # Combinar los índices de NaN de ambas series
    nan_indices = np.logical_or(nan_indices_actual, nan_indices_forecast)

    # Eliminar valores NaN de ambos arrays
    actual_clean = actual[~nan_indices]
    forecast_clean = forecast[~nan_indices]

    me = np.mean(forecast_clean - actual_clean)     # ME
    mae = np.mean(np.abs(forecast_clean - actual_clean))     # MAE
    mse = np.mean((forecast_clean - actual_clean)**2)
```

```
    rmse = np.sqrt(mean_squared_error(actual_clean, forecast_clean))  # RMSE

    mpe = np.mean((forecast_clean - actual_clean)/actual_clean)   # MPE
    mape = np.mean(np.abs(forecast_clean - actual_clean)/np.abs(actual_clean))  # MAPE
    corr = np.corrcoef(forecast_clean, actual_clean)[0,1]   # corr
    mins = np.amin(np.hstack([forecast_clean[:,None],
                              actual_clean[:,None]]), axis=1)
    maxs = np.amax(np.hstack([forecast_clean[:,None],
                              actual_clean[:,None]]), axis=1)
    minmax = 1 - np.mean(mins/maxs)             # minmax
    return (me, mae, mpe, mape, rmse, corr, minmax)
```

La función métricas se usará para recoger los errores en una tabla, será un dataframe denominado df_errores que estará mencionado al final de cada bloque de código donde se completan las filas de errores ya que Colab lo imprime en formato tabla, al ser impreso como tabla es más sencillo su manejo.

La descomposición de una serie temporal es un proceso mediante el cual se separan los diferentes componentes de la serie, permitiendo así analizar y modelar cada componente por separado. Una técnica comúnmente utilizada para la descomposición es el método seasonal_decompose en Python, que separa la serie en sus componentes de tendencia, estacionalidad y residuos, se puede consultar visualmente el resultado en Figura 7-2.

6.2.1 Métodos de previsión para patrones de demanda constante

En este apartado con una demanda constante el dataframe estará compuesto por dos columnas, una de ellas denominada “date” es la columna índice y la otra denominada “unit_sales” que contendrá los datos de ventas mensuales del producto.

6.2.1.1 Medias Móviles

Se distingue en este trabajo entre dos tipos de medias móviles, simple y ponderada, se detalla en cada apartado las funciones desarrolladas para cada una de ellas. La media móvil ponderada es una evolución de la simple, donde se le añade más peso a las últimas observaciones.

6.2.1.1.1 Media Móvil Simple

La librería usada fundamentalmente para este modelo es Pandas de (The pandas development team, 2020)

Para el cálculo de la media móvil simple se puede escoger el parámetro ‘**N**’ que es el número de elementos que conforman la media y ‘**valor**’ el tamaño o fracción de los elementos de entrenamiento del modelo donde podrá escoger un valor entre cero y uno.

```
######### Elegir tamaño de ventana y longitud de datos de entranamiento y testeo #######
# Se define el valor N
N=4
# Se define la longitud del tamaño de entrenamiento, su valor debe ser [0,1]
valor=0.8
train_size=int(len(df_res)*valor) # 80% para entrenamiento
train_data = copy.deepcopy(df_res_copia[:train_size])
forecast_data = copy.deepcopy(df_res_copia[train_size:])
#################################################################################
```

La función necesaria para el cálculo de la media es ‘**MMS**’:

```
def MMS(train,variable,size):
    # En este método media móvil:
    forecast_value = train[variable].rolling(size).mean().shift(1) # shift(1) permite
desplazar el resultado en 1
    forecast_value2 = train[variable].rolling(size).mean().iloc[-1]
    return forecast_value,forecast_value2
```

La función **MMS**, calcula la media móvil simple de una serie de tiempo dada utilizando la biblioteca **pandas**. El procedimiento paso a paso se describe a continuación:

1. Se introducen tres parámetros como entrada:
    - **train**: El conjunto de datos de entrenamiento que contiene la serie de tiempo.
    - **Variable**: El nombre de la columna que contiene la serie de tiempo dentro del conjunto de datos de entrenamiento.
    - **size**: El tamaño de la ventana para calcular la media móvil.
2. Calcula la media móvil simple:
    - **train[variable].rolling(size).mean()**: Utiliza el método **rolling()** de **pandas** para calcular la media móvil simple de la serie de tiempo especificada por **variable** con una ventana de tamaño **size**.
    - **shift(1)**: Desplaza el resultado un paso hacia adelante, de modo que el valor de la media móvil se ajuste para predecir el siguiente punto en la serie de tiempo.
    - **iloc[-1]**: Obtiene el último valor de la serie de tiempo resultante, que representa la predicción de la media móvil para el siguiente punto en el tiempo después del conjunto de datos de entrenamiento.
3. Retorna dos valores:
    - **forecast_value**: Una serie de tiempo que contiene la media móvil simple calculada, desplazada un paso hacia adelante.
    - **forecast_value2**: El valor de la media móvil simple predicho para el siguiente punto en el tiempo después del conjunto de datos de entrenamiento.

Donde devuelve dos valores, el primero es la lista de valores del entrenamiento y el segundo valor es la predicción a partir de la fecha impuesta.

La implementación del método se presenta a continuación:

```
entrenamiento,prediccion = MMS(train_data,'unit_sales',N)
train_data['MMS']=copy.deepcopy(entrenamiento)
print(f'la prediccion  : {prediccion}')
forecast_data['MMS'] = copy.deepcopy(prediccion)
```

Para analizar la calidad de los resultados obtenidos se llama a la función métricas que permite elegir los mejores parámetros que minimicen el error, se crea un dataframe específico donde se alojaran los errores de los métodos:

```
# Dataframe para errores
df_errores = pd.DataFrame()

unit_sales = train_data['unit_sales'].to_numpy()
MMS = train_data['MMS'].to_numpy()

# Usar la función métricas
errores = metricas(unit_sales, MMS)

# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[0, 'Modelo'] = 'Media Móvil Simple'
df_errores.loc[0, 'me'] = errores[0]
df_errores.loc[0, 'mae'] = errores[1]
df_errores.loc[0, 'mpe'] = errores[2]
df_errores.loc[0, 'mape'] = errores[3]
df_errores.loc[0, 'rmse'] = errores[4]
df_errores.loc[0, 'corr'] = errores[5]
df_errores.loc[0, 'minmax'] = errores[6]
df_errores
```

La representación donde permite visualizar las gráficas de los valores reales en naranja, el entrenamiento en negro y la previsión en verde.

```
# plot
plt.figure(figsize=(12,6))
plt.plot(train_data['unit_sales'],'-o',color='orange',label='Unit_sales')
plt.plot(train_data['MMS'],'-o',color='black',label='MMS')
plt.plot(forecast_data['MMS'],'-o',color='green',label='forecast')

plt.xlabel("PERIODOS",fontsize=14)
plt.ylabel("VENTAS",fontsize=14)
plt.title("SERIE",fontsize=18)
plt.legend()
plt.show()
```

6.2.1.1.2 Medias Móviles Ponderadas

En este modelo los parámetros de elección son el número de elementos a seleccionar y los pesos a aplicar a cada elemento:

```
######## Elegir 'N' y 'pesos' ##########################
# Calcular la media móvil con un valor N de 4
N=4
pesos=[0.2,0.3,1,2.5] # Añadir los pesos que se consideren oportunos, la suma debe vale  N
########################################################
```

La función necesaria es la que calcula la media móvil ponderada teniendo en cuenta a los pesos anteriormente definidos, se presenta a continuación la implementación de la función:

```
def media_movil_ponderada_N(datos, pesos, N_size):
    resultado = []

    # Rellenar con NaN para los primeros elementos
    resultado.extend([np.nan] * (N_size))

    for i in range(N_size - 1, len(datos)-1):
        N = datos[i - N_size + 1: i + 1] # Ventana deslizante
        media_ponderada = np.average(N, weights=pesos)
        resultado.append(media_ponderada)
    i+=1
    N = datos[i - N_size + 1: i + 1] # Ventana deslizante
    media_ponderada2 = np.average(N, weights=pesos)
    return resultado,media_ponderada2
```

La función **media_movil_ponderada_N** calcula la media móvil ponderada de una serie de tiempo utilizando los pesos especificados de la siguiente forma:

1. Se introducen tres parámetros como entrada:
    - **datos**: La serie de tiempo sobre la cual se calculará la media móvil ponderada.
    - **pesos**: Una lista de pesos a aplicar a cada elemento dentro de la ventana de la media móvil.
    - **N_size**: El tamaño de la ventana para calcular la media móvil.
2. Inicializa una lista llamada **resultado** para almacenar los resultados de la media móvil ponderada.
3. Rellena los primeros **N_size** elementos de la lista **resultado** con valores **NaN**, ya que no hay suficientes datos para calcular la media móvil ponderada al principio de la serie de tiempo.
4. Itera sobre los datos desde el índice **N_size - 1** hasta el final de la serie de tiempo:
    - Para cada iteración, selecciona una ventana de datos de tamaño **N_size**.
    - Calcula la media ponderada de la ventana de datos utilizando la función **np.average()** de NumPy, aplicando los pesos especificados.

- Agrega el resultado de la media ponderada a la lista **resultado**.

5. Calcula la media ponderada para el último conjunto de datos de la serie, similar al paso anterior, pero sin iterar.
6. Retorna dos valores:
   - **resultado**: Una lista que contiene los valores de la media móvil ponderada para cada punto en la serie de tiempo, incluidos los valores **NaN** al principio.
   - **media_ponderada2**: El valor de la media móvil ponderada para el siguiente punto en el tiempo después del conjunto de datos de entrenamiento.

En su ejecución permite obtener dos resultados, el valor de entrenamiento y la predicción:

```
media,prediccion=media_movil_ponderada_N(train_data['unit_sales'],pesos,N)

train_data['MMS_P']=copy.deepcopy(media)
forecast_data['MMS_P'] = copy.deepcopy(prediccion)
```

Se invoca a la función metricas() y se crea otra fila en el dataframe para alojar los errores:

```
MMS_P = train_data['MMS_P'].to_numpy()
# Se usa la función métricas
errores = metricas(unit_sales, MMS_P)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[1, 'Modelo'] = 'Media Móvil Simple Ponderada'
df_errores.loc[1, 'me'] = errores[0]
df_errores.loc[1, 'mae'] = errores[1]
df_errores.loc[1, 'mpe'] = errores[2]
df_errores.loc[1, 'mape'] = errores[3]
df_errores.loc[1, 'rmse'] = errores[4]
df_errores.loc[1, 'corr'] = errores[5]
df_errores.loc[1, 'minmax'] = errores[6]
df_errores
```

La representación:

```
# plot
plt.figure(figsize=(12,6))
plt.plot(df_res['unit_sales'],'-o',color='orange',label='Unit_sales')
plt.plot(train_data['MMS_P'],'-o',color='black',label='MMS_P')
plt.plot(forecast_data['MMS_P'],'-o',color='green',label='forecast')


plt.xlabel("PERIODOS",fontsize=14)
plt.ylabel("VENTAS",fontsize=14)
plt.title("SERIE",fontsize=18)
plt.legend()
plt.show()
```

6.2.1.2 Ajuste Exponencial Simple

Para el ajuste exponencial simple se ha implementado dos modos diferentes, uno denominado '**manual**' donde se puede experimentar con los valores de alfa y otro denominado '**automático**' que calcula el mejor valor de alfa que minimiza el error cuadrático medio.

La librería usada para el modo manual y modo automático será '**statsmodels**', de la que se importa '**SimpleExpSmoothing**' (Seabold et al., 2010).

6.2.1.2.1 Modo Manual

El parámetro de elección que se puede escoger en modo manual es '**alpha**':

```
#### Añadir el valor de alpha ##############
# Ajuste del modelo en datos de entrenamiento con alpha óptimo
alpha = 0.2
##########################################
```

La implementación permite invocar a la función que calcula el modelo, se define a continuación:

```
modelo = SimpleExpSmoothing(train_data['unit_sales']).fit(smoothing_level=alpha,
optimized=False)
train_data['AE_S[M]'] = modelo.fittedvalues

# Predicción en datos de prueba
forecast_data['AE_S[M]'] = modelo.forecast(int(len(forecast_data)))
```

Se ajusta el modelo de suavizado exponencial simple a los datos de entrenamiento utilizando el valor de **alpha** especificado. Esto se realiza utilizando la función **SimpleExpSmoothing** de la biblioteca **statsmodels**.

Se crea otra línea en el dataframe de errores para que se pueda comparar también junto al modo automático donde verá los mejores valores de alfa:

```
AES = train_data['AE_S[M]'].to_numpy()
unit_sales = train_data['unit_sales'].to_numpy()
# Se usa la función métricas
errores = metricas(unit_sales, AES)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[2, 'Modelo'] = 'Ajuste Exponencial Simple Modo Manual'
df_errores.loc[2, 'me'] = errores[0]
df_errores.loc[2, 'mae'] = errores[1]
df_errores.loc[2, 'mpe'] = errores[2]
df_errores.loc[2, 'mape'] = errores[3]
df_errores.loc[2, 'rmse'] = errores[4]
df_errores.loc[2, 'corr'] = errores[5]
df_errores.loc[2, 'minmax'] = errores[6]
df_errores
```

El código con la representación será el siguiente:

```
# Graficar resultados
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales'], '-o', color='orange', label='Training Data')
plt.plot(train_data['AE_S[M]'], '-o', color='black', label='AES')
plt.plot(forecast_data['AE_S[M]'], '-o', color='green', label='Forecasted Values (Test)')
plt.xlabel("PERIODOS", fontsize=14)
plt.ylabel("VENTAS", fontsize=14)
plt.title("SERIE", fontsize=18)
plt.legend()
plt.show()
```

6.2.1.2.2 Modo Automático

Al tratarse de un modelo que calcula automáticamente los mejores valores tan solo debemos marcar 'True' en los parámetros a elección:

```
############### Alfa óptimo ############
optimizado_automático = True # marcar True para saber mejor valor de alfa
########################################
```

Se realizan predicciones utilizando el modelo ajustado en los datos de prueba, de manera similar al modo manual, la ejecución alojará por pantalla el mejor valor de alfa para el modelo:

```
# Ajuste del modelo en datos de entrenamiento con alpha óptimo
modelo = SimpleExpSmoothing(train_data['unit_sales']).fit(optimized=optimizado_automático)
print(modelo.summary())
train_data['AE_S[A]'] = modelo.fittedvalues

# Predicción en datos de prueba
forecast_data['AE_S[A]'] = modelo.forecast(int(len(forecast_data)))  # Inicializar columna
con valores NaN
```

Se añadirá en este caso la última fila al dataframe de errores para que pueda ser comparado con los demás:

```
AES = train_data['AE_S[A]'].to_numpy()
unit_sales = train_data['unit_sales'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, AES)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[3, 'Modelo'] = 'Ajuste Exponencial Simple Alpha óptimo'
df_errores.loc[3, 'me'] = errores[0]
df_errores.loc[3, 'mae'] = errores[1]
df_errores.loc[3, 'mpe'] = errores[2]
df_errores.loc[3, 'mape'] = errores[3]
df_errores.loc[3, 'rmse'] = errores[4]
df_errores.loc[3, 'corr'] = errores[5]
df_errores.loc[3, 'minmax'] = errores[6]
df_errores
```

Y su respectiva representación:

```
# Graficar resultados
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales'], '-o', color='orange', label='Training Data')
plt.plot(train_data['AE_S[A]'], '-o', color='black', label='AES')
plt.plot(forecast_data['AE_S[A]'], '-o', color='green', label='Forecasted Values')
plt.xlabel("PERIODOS", fontsize=14)
plt.ylabel("VENTAS", fontsize=14)
plt.title("SERIE", fontsize=18)
plt.legend()
plt.show()
```

Para guardar los valores obtenidos de los dataframes principales se hace una copia, en este caso la terminación **DC** significa demanda constante.

```
train_DC = copy.deepcopy(train_data)
forecast_DC = copy.deepcopy(forecast_data)
df_errores_DC=copy.deepcopy(df_errores)
```

6.2.2 Métodos de previsión para patrones de demanda con tendencia

Se vuelve a usar la copia original del dataframe para usarlo limpio en los patrones de demanda con tendencia.

```
df_res = copy.deepcopy(df_res_copia)
```

Para exponer mejor los métodos que se usan para patrones de demanda con una tendencia ya sea positiva o negativa se han creado mediante código nuevos datos de ventas de forma que estos muestren para el caso expuesto una tendencia creciente con una pendiente de 10, de esta manera se podrá asimilar más fácilmente los conceptos para modelos de previsión con tendencia.

Además, se podrá escoger la pendiente que desea para los datos y el porcentaje de valores de entrenamiento que desea.

```
###### Valor de pendiente a elegir y porcentaje de valores de entrenamiento#######
pendiente = 10
valor = 0.8  # tamaño de la ventana de valores de entrenamiento (valor entre cero y uno)
########################################
```

La ejecución de código donde se realiza la adición de pendiente es el siguiente:

```
import matplotlib.pyplot as plt

# Convertir el índice a valores numéricos
df_res.reset_index(drop=False, inplace=True)

# Definir la pendiente positiva
slope = pendiente

# Aplicar la pendiente positiva a la serie de datos
df_res['unit_sales_with_slope'] = df_res['unit_sales'] + slope * df_res.index

# Convertir la columna 'date' para que solo contenga la fecha (sin horas, minutos y
segundos)
df_res['date'] = df_res['date'].dt.date
df_res.set_index('date', inplace=True)

train_size=int(len(df_res)*valor)
train_data = copy.deepcopy(df_res[:train_size])
forecast_data = copy.deepcopy(df_res[train_size:])

# Representar las gráficas
plt.figure(figsize=(10, 5))
plt.plot(df_res.index, df_res['unit_sales'], label='Datos originales')
plt.plot(df_res.index, df_res['unit_sales_with_slope'], label='Datos con pendiente
positiva', linestyle='--')
plt.xlabel('Índice')
plt.ylabel('Unit Sales')
plt.title('Datos originales vs Datos con pendiente positiva')
plt.legend()
plt.grid(True)
plt.show()
```

El fragmento de código proporcionado realiza varias operaciones:

1. Convertir el índice a valores numéricos:
    - **df_res.reset_index(drop=False, inplace=True)**: Esta línea de código resetea el índice del DataFrame **df_res** y lo convierte en una columna numérica. El parámetro **drop=False** evita que el índice anterior se agregue como una columna adicional y **inplace=True** modifica el DataFrame original en lugar de crear uno nuevo.
2. Definir la pendiente positiva:
    - **slope = pendiente**: Aquí se asigna un valor previamente definido a la variable **slope**, que representa la pendiente positiva deseada.
3. Aplicar la pendiente positiva a la serie de datos:
    - **df_res['unit_sales_with_slope'] = df_res['unit_sales'] + slope * df_res.index**: Esta línea calcula una nueva serie de datos llamada **'unit_sales_with_slope'**, que es la suma de la serie de datos original **'unit_sales'** y la pendiente multiplicada por el índice. Esto efectivamente agrega una pendiente positiva a los datos originales.
4. Volver a establecer el índice en la columna de fecha:
    - **df_res.set_index('date', inplace=True)**: Después de realizar las operaciones anteriores, el índice se vuelve a establecer en la columna **'date'**, que parece ser la columna que contiene las

fechas de los datos.

5. Dividir los datos en conjuntos de entrenamiento y prueba:
    - **train_size=int(len(df_res)*valor)**: Se calcula el tamaño del conjunto de entrenamiento como un porcentaje del tamaño total del DataFrame **df_res**.
    - **train_data = copy.deepcopy(df_res[:train_size])**: Se crea un DataFrame llamado **train_data** que contiene los primeros **train_size** registros de **df_res**, que se utilizarán como datos de entrenamiento.
    - **forecast_data = copy.deepcopy(df_res[train_size:])**: Se crea un DataFrame llamado **forecast_data** que contiene los registros restantes de **df_res**, que se utilizarán como datos de prueba para hacer predicciones.
6. Graficar los datos originales y los datos generados con pendiente positiva:
    - Se utiliza **matplotlib** para crear un gráfico que muestra los datos originales y los datos generados con la pendiente positiva. Esto permite visualizar cómo la pendiente afecta a la serie de datos original.

6.2.2.1 Medias Móviles Dobles

Las librerías usadas para el cálculo de la media móvil doble son **numpy** de (Harris et al., 2020) y **math** de (Van Rossum, 2020)**.** La librería **math** proporciona funciones matemáticas comunes, mientras que **numpy** es una librería fundamental para trabajar con matrices y realizar operaciones numéricas eficientes en Python

El único parámetro a definer por parte del usuario para este método es **N**

```
########### Definir 'N' #############
N = 7
##########################################
```

La función necesarias para el cálculo de la media móvil doble es la siguiente:

```
def calcular_M(lista, N):
    M_valores = []
    M_valores.extend([np.nan] * (N-1))

    for i in range(len(lista) - N + 1):
        ventana_actual = lista[i:i + N]
        M_valores.append(sum(ventana_actual)/N)

    return M_valores
```

La explicación de la función se detalla en los siguientes pasos:

- Definición de la función: La función **calcular_M** toma dos argumentos: **lista**, que representa la serie de datos, y **N**, que especifica el tamaño de la ventana para el cálculo de la MMS.
- Inicialización de la lista de valores de MMS: Se crea una lista vacía llamada **M_valores** que almacenará los valores de la MMS.
- Extensión de la lista con NaN: Se agregan **(N-1)** valores NaN al principio de la lista **M_valores**. Así se mantiene la longitud de la lista igual que la serie de datos original y para garantizar que la MMS comience correctamente.
- Cálculo de la MMS: Se itera sobre la serie de datos utilizando un bucle **for**. En cada iteración, se toma una ventana de tamaño **N** de la serie de datos, comenzando desde el índice **i** y terminando en **i + N**. Se calcula la media de los valores en esta ventana y se añade a la lista **M_valores**.
- Retorno de la lista de valores de MMS: Una vez que se han calculado todos los valores de la MMS, la función devuelve la lista **M_valores**.

La ejecución de código para la media móvil doble se detalla a continuación:

```
df_p = copy.deepcopy(train_data)

Mt= calcular_M(df_p['unit_sales_with_slope'],N)
Mt2= calcular_M(Mt,N)
df_p['Mt']=Mt
df_p['Mt2']=Mt2

j=0
df_res['MMD']=np.nan
train_data['MMD']=np.nan

for i in range((N-1)*2,len(Mt)-1): # i lleva el contador de los Mt y Mt2, siempre será
inferior en 1 respecto a donde se va calculando
    D=[]
    D.extend([np.nan] * (N+N-1+j))
    for tau in range(1,(len(Mt))-i ):
        d = 2 *df_p.iloc[i, df_p.columns.get_loc('Mt')] - (
        df_p.iloc[i, df_p.columns.get_loc('Mt2')] +
        (tau) * 2 * ((N-1) ** (-1)) *
        (df_p.iloc[i, df_p.columns.get_loc('Mt')] -
        df_p.iloc[i, df_p.columns.get_loc('Mt2')])
        )
        D.append(d)
        if (tau)==1:
          df_res.loc[df_res.index[i+1],'MMD']=d
          train_data.loc[train_data.index[i+1],'MMD']=d

    # Conocer el primer valor no NaN de D
    indice_primer_valor_no_nan = None

    for p in range(len(D)):
        if not math.isnan(D[p]):
            indice_primer_valor_no_nan = p
            break

    T= df_p.index[indice_primer_valor_no_nan-1].strftime('%Y-%m-%d')
    df_p[f'T={T}'] = D
    j+=1

mt=df_p['Mt'].iloc[-1]
mt2=df_p['Mt2'].iloc[-1]
forecast_data['MMD']=np.nan
tau=1
for i in range(0,len(forecast_data)):
  d=2*mt-mt2+(tau)*2 * ((N-1) ** (-1)) *(mt-mt2)
  forecast_data.iloc[i,forecast_data.columns.get_loc('MMD')]=d
  tau+=1
```

1. Creación de un DataFrame auxiliar (df_p):
    - Se realiza una copia profunda del DataFrame de datos de entrenamiento **train_data** y se almacena en **df_p**. Esta copia se hace para evitar modificar el DataFrame original durante el proceso de cálculo.
2. Cálculo de las MMD:
    - Se calcula el primer estimador, denotado como **Mt**, utilizando la función **calcular_M** en la columna **'unit_sales_with_slope'** del DataFrame **df_p**.
    - Se calcula el segundo estimador, denotado como **Mt2**, utilizando nuevamente la función **calcular_M** en la serie **Mt**.

- Estos valores se añaden como columnas adicionales **'Mt'** y **'Mt2'** en el DataFrame **df_p**.

3. Iteración para calcular la MMD y predecir valores:
   - Se inicializa un contador **j**.
   - Se inicializan las columnas **'MMD'** en NaN en los DataFrames **df_res** y **train_data**.
   - Se itera sobre un rango específico de valores para calcular las MMD adicionales y predecir valores futuros.
   - Dentro del bucle:
     - Se inicializa una lista vacía **D** para almacenar los valores de la MMD en cada iteración.
     - Se calcula el valor de **d** para la MMD actual utilizando la fórmula dada.
     - Se añade el valor **d** a la lista **D**.
     - Si **tau** es igual a 1, se asigna el valor de la MMD al siguiente índice en las columnas **'MMD'** de **df_res** y **train_data**.
     - Se calcula el valor de **T**, que representa la fecha en la que se produce la primera predicción basada en la MMD actual, y se añade como una nueva columna en el DataFrame **df_p**.
     - Se incrementa el contador **j**.
4. Cálculo de las MMD para datos de pronóstico:
   - Se calculan las últimas MMD utilizando la fórmula dada para los datos de pronóstico en el DataFrame **forecast_data**. Los valores resultantes se almacenan en la columna **'MMD'**

A continuación, se vuelve a crear un dataframe exclusivo para errores con su respective dataframe para la comparación de los modelos de prevision para patrones con tendencia:

```
# Dataframe para errores
df_errores = pd.DataFrame()
MMD = train_data['MMD'].to_numpy()
unit_sales = train_data['unit_sales_with_slope'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, MMD)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[0, 'Modelo'] = 'Media Móvil Doble'
df_errores.loc[0, 'me'] = errores[0]
df_errores.loc[0, 'mae'] = errores[1]
df_errores.loc[0, 'mpe'] = errores[2]
df_errores.loc[0, 'mape'] = errores[3]
df_errores.loc[0, 'rmse'] = errores[4]
df_errores.loc[0, 'corr'] = errores[5]
df_errores.loc[0, 'minmax'] = errores[6]
df_errores
```

Con su respectiva representación:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_with_slope'],marker='o',color='orange',label='valores con
pendiente')
plt.plot(train_data['MMD'],marker='o',color='black',label='entrenamiento')
plt.plot(forecast_data['MMD'],marker='o',color='green',label='forecast')
```

```
plt.xlabel('Fecha',fontsize=14)
plt.ylabel('Valor de Predicción',fontsize=14)
plt.title('Predicciones basadas en media móvil doble', fontsize=18)
plt.legend()
plt.show()
```

6.2.2.2 Ajuste Exponencial Doble

Las librerías importadas vuelven a ser como anteriormente en la media móvil doble **math** y **numpy**.

Los parámetros que el usuario puede elegir para ajustar el modelo de suavización exponencial doble:

```
########### Elegir hasta que periodo para recta de regresión #########
t = 13  # Por ejemplo, los primeros 13 periodos
###############################################################################

######### Elegir alpha #########
# Factor de suavización
alpha = 0.2
################################
```

Donde '**t**' y '**alpha**' representa lo siguiente:

1. Número de períodos para calcular la recta de regresión **(t)**:
    - Este parámetro determina cuántos períodos se utilizarán para calcular la tendencia de la serie temporal mediante una regresión lineal.
2. Factor de suavización **(alpha)**:
    - Este parámetro define el nivel de suavizado exponencial aplicado a la serie temporal. Un valor más bajo de **alpha** dará más peso a los datos históricos, mientras que un valor más alto dará más peso a los datos más recientes.

Las funciones necesarias para el cálculo del ajuste exponencial doble son las siguientes:

```
def calcular_st(alpha,demanda,s0,t):
  serie_st = []
  serie_st.extend([np.nan] * (t-1))
  serie_st.append(s0)
  serie_st.append(alpha * demanda[t] + (1 - alpha) * s0)

  for i in range(len(serie_st),len(demanda)):
    st = alpha * demanda[i] + (1 - alpha) * serie_st[i-1]
    serie_st.append(st)

  return serie_st
def regresion(df, t):
  # Seleccionar los primeros t periodos
  subset = df.iloc[:t]

  # Extraer los valores de las unidades de ventas con tendencia
  x = np.arange(t)
  y = subset['unit_sales_with_slope'].values

  # Calcular la regresión lineal
  pendiente, eje_ordenada = np.polyfit(x, y, 1) # Recta de regresión de orden 1

  # Calcular los valores de y para la recta de regresión
  recta = pendiente * x + eje_ordenada
  return pendiente, eje_ordenada , recta
```

Donde cada una de ellas realiza lo siguiente:

1. Función **calcular_st**:
    - Esta función calcula la serie de nivel suavizada **st** utilizando el método de suavización exponencial. Toma como entrada el coeficiente de suavización **alpha**, la serie de demanda **demanda**, el valor inicial **s0** y el índice de tiempo **t**.
    - La serie **serie_st** se inicializa con valores NaN para los primeros **t-1** periodos y luego se agregan los primeros dos valores: **s0** y el valor suavizado para el primer período utilizando la fórmula de suavización exponencial.
    - Para cada período subsiguiente, se calcula el valor suavizado **st** utilizando la fórmula de suavización exponencial y se agrega a la serie.
    - La función devuelve la serie suavizada **serie_st**.
2. Función **regresión**:
    - Esta función realiza una regresión lineal para encontrar la tendencia de los primeros **t** periodos de una serie temporal.
    - Toma como entrada un DataFrame **df** que contiene los datos de la serie temporal y el número de períodos **t** para los cuales se realizará la regresión.
    - Selecciona los primeros **t** periodos del DataFrame.
    - Extrae los valores de las unidades de ventas con tendencia para estos períodos.
    - Calcula la regresión lineal utilizando la función **np.polyfit()** de NumPy, que ajusta una línea recta a los datos. (Jorge A. Perez Prieto, 2019)
    - Devuelve la pendiente, la ordenada en el origen y los valores de la recta de regresión.

El código donde se ejecutan los cálculos para el ajuste exponencial doble es el siguiente:

```
pendiente, eje_ordenada, recta = regresion(train_data, t)
print("Pendiente:", pendiente)
print("Eje ordenada:", eje_ordenada)
df_p = copy.deepcopy(train_data)


# Inicialización  de regresión
p0=slope
D0=eje_ordenada + t * slope

S0 = D0 - p0 * (1-alpha)/alpha
S0_2= D0 - 2 * p0 * (1-alpha)/alpha

ST = calcular_st (alpha,df_p['unit_sales_with_slope'],S0,t)
df_p['ST']=ST
ST2 = calcular_st (alpha,df_p['ST'],S0_2,t)
df_p['ST2']=ST2

df_res['AED']=np.nan
train_data['AED']=np.nan
j=0
for i in range(t-1,len(df_p)-1):
    D=[]
    D.extend([np.nan] * (t+j))
    for tau in range(1,(len(df_p))-i):
        d = 2 * df_p.iloc[i, df_p.columns.get_loc('ST')] -(
        df_p.iloc[i, df_p.columns.get_loc('ST2')] +
        (tau) * alpha* ((1-alpha) ** (-1)) *
        (df_p.iloc[i, df_p.columns.get_loc('ST')] -
        df_p.iloc[i, df_p.columns.get_loc('ST2')])
```

```
        )
        D.append(d)


        if (tau)==1 :
          df_res.loc[df_res.index[i+1],'AED']=d
          train_data.loc[train_data.index[i+1],'AED']=d
    # Conocer el primer valor no NaN de D
    indice_primer_valor_no_nan = None
    for p in range(len(D)):
        if not math.isnan(D[p]):
            indice_primer_valor_no_nan = p
            break

    T= df_p.index[indice_primer_valor_no_nan-1].strftime('%Y-%m-%d')
    df_p[f'T={T}'] = D
    j+=1

st=df_p['ST'].iloc[-1]
st2=df_p['ST2'].iloc[-1]
forecast_data['AED']=np.nan
tau=1
for i in range(0,len(forecast_data)):
  d = 2*st - st2 + tau * (alpha/(1-alpha)) *(st-st2)
  forecast_data.iloc[i,forecast_data.columns.get_loc('AED')]=d
  tau+=1
```

Este fragmento de código ejecuta el ajuste exponencial doble y el cálculo de las predicciones:

1. Cálculo de la regresión lineal:
    - Se obtiene la pendiente y la ordenada al origen de una regresión lineal sobre los primeros **t** períodos de los datos de entrenamiento.
2. Inicialización de variables:
    - Se inicializan las variables necesarias para el cálculo del suavizado exponencial doble, utilizando los parámetros obtenidos de la regresión lineal.
3. Cálculo del suavizado exponencial doble:
    - Se calculan los estimadores (**ST** y **ST2**) utilizando la función **calcular_st**.
4. Cálculo de las predicciones:
    - Se calculan las predicciones utilizando el suavizado exponencial doble, basándose en las últimas observaciones y aplicando la fórmula correspondiente. Dichos valores para cada primer periodo se almacenan en **train_data** para poder ser luego argumentos en la función **métricas** y calcular y analizar los errores
5. Almacenamiento de resultados:
    - Los resultados se almacenan en el DataFrame **forecast_data** para su posterior análisis y visualización.

Como anteriores veces se añade a continuación una nueva fila en el dataframe de errores para que se pueda comparar según las variables elegidas:

```
AED = train_data['AED'].to_numpy()
unit_sales = train_data['unit_sales_with_slope'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, AED)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
```

```
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[1, 'Modelo'] = 'Ajuste Exponencial Doble'
df_errores.loc[1, 'me'] = errores[0]
df_errores.loc[1, 'mae'] = errores[1]
df_errores.loc[1, 'mpe'] = errores[2]
df_errores.loc[1, 'mape'] = errores[3]
df_errores.loc[1, 'rmse'] = errores[4]
df_errores.loc[1, 'corr'] = errores[5]
df_errores.loc[1, 'minmax'] = errores[6]
df_errores
```

Y su respectiva representación, en este caso se añade una nueva gráfica de color rojo perteneciente a la recta de regresión:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_with_slope'],marker='o',color='orange',label='valores con
pendiente')
plt.plot(train_data['AED'],marker='o',color='black',label='entrenamiento')
plt.plot(forecast_data['AED'],marker='o',color='green',label='test')
plt.plot(df_res.index[:t], recta, color='red', linestyle='-', label='Recta de Regresión')

plt.xlabel('Fecha',fontsize=14)
plt.ylabel('Valor de Predicción',fontsize=14)
plt.title('Predicciones basadas en ajuste exponencial doble', fontsize=18)
plt.legend()
```

6.2.2.3 Holt mediante 'statsmodels'

Como en el Ajuste Exponencial Simple para patrones de demanda constante se ha hecho una division entre un modo manual donde se pueden variar los parámetros o variables y un modo automático que calcula las constantes '**alfa**' y '**beta**' óptimas del modelo.

6.2.2.3.1 Modo Manual

El modo manual permite al usuario escoger diferentes parámetros para que experimente la manera más idónea en la minización del error.

En el modo manual se debe introducir los parámetros a elección que se muestra en el siguiente código:

```
########### Elegir hasta que periodo se calcula la recta de regresión #########
t = 13  # Por ejemplo, los primeros 13 periodos

########### alfa, beta ##############
alpha=0.6
beta=0.8
###################################
```

El usuario puede escoger los siguientes parámetros:

1. Número de períodos para calcular la recta de regresión **(t)**: Este parámetro determina cuántos períodos se utilizarán para calcular la recta de regresión. En el ejemplo, se establece en 13 períodos.
2. Parámetros de suavización **(alfa y beta)**: Son los parámetros de suavización para el nivel y la tendencia, respectivamente, en el modelo Holt. En el código, se establecen los valores de alfa y beta en 0.6 y 0.8, respectivamente.

La ejecución de código para obtener los valores del modelo de entrenamiento y predicción es el siguiente:

```
pendiente, eje_ordenada, recta = regresion(train_data, t)
print("Pendiente:", pendiente)
print("Eje ordenada:", eje_ordenada)
# Inicialización  REGRESIÓN
```

```
p0=slope
D0=eje_ordenada + t * slope

# Establecer la frecuencia de los datos
train_data.index.freq = 'M'
forecast_data.index.freq = 'M'

# Ajuste del modelo en datos de entrenamiento con alpha y beta óptimos
modelo = Holt(train_data['unit_sales_with_slope'].iloc[t:],initialization_method='known',
initial_level=p0, initial_trend=D0).fit(smoothing_level=alpha, smoothing_trend=beta,
optimized=False)
train_data['Holt'] = modelo.fittedvalues

# Predicción en datos de prueba
forecast_data['Holt'] = np.nan  # Inicializar columna con valores NaN

forecast_data['Holt'] = modelo.forecast(int(len(forecast_data)))  # Inicializar columna con
valores NaN
```

Los pasos que realiza el anterior código se detallan a continuación:

1. Regresión lineal: Utiliza la función **regresion** para calcular la pendiente (**pendiente**) y la intersección en el eje y (**eje_ordenada**) de una recta de regresión lineal utilizando los datos de entrenamiento (**train_data**) y el número de períodos especificado (**t**).
2. Inicialización de la regresión: Se inicializan los valores de la pendiente (**p0**) y la intersección en el eje y (**D0**) utilizando valores previamente calculados (**slope** y **eje_ordenada**).
3. Establecimiento de la frecuencia de los datos: Se establece la frecuencia de los datos en "M" (mes) para ambos conjuntos de datos.
4. Ajuste de Holt: El modelo de Holt se ajusta al entrenamiento (**train_data**) con sus parámetros de suavización escogidos (**alpha** y **beta**), los valores iniciales de la pendiente y el valor de la recta al final de esta (**p0** y **D0**) mediante el método de inicialización conocido.
5. Predicción en los datos de prueba: Se realizan predicciones para los datos de prueba utilizando el modelo Holt ajustado (**modelo**) y se almacenan en la columna "Holt" del conjunto de datos de predicción (**forecast_data**).

Como es habitual se añade una fila más en el dataframe de errores:

```
HOLT = train_data['Holt'].to_numpy()
unit_sales = train_data['unit_sales_with_slope'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, HOLT)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[2, 'Modelo'] = 'Holt Manual'
df_errores.loc[2, 'me'] = errores[0]
df_errores.loc[2, 'mae'] = errores[1]
df_errores.loc[2, 'mpe'] = errores[2]
df_errores.loc[2, 'mape'] = errores[3]
df_errores.loc[2, 'rmse'] = errores[4]
df_errores.loc[2, 'corr'] = errores[5]
df_errores.loc[2, 'minmax'] = errores[6]
df_errores
```

Con su respectiva representación, también con la recta de regresión:

```
# Representación
plt.figure(figsize=(12,6))
```

```
plt.plot(df_res['unit_sales_with_slope'],'-o',color='orange',label='Unit_sales')
plt.plot(train_data['Holt'],'-o',color='black',label='prediccion_Holt')
plt.plot(forecast_data['Holt'],'-o',color='green',label='forecast_Holt')
plt.plot(df_res.index[:t], recta, color='red', linestyle='-', label='Recta de Regresión')

plt.xlabel("PERIODOS",fontsize=14)
plt.ylabel("VENTAS",fontsize=14)
plt.title("SERIE",fontsize=18)
plt.legend()
plt.show()
```

6.2.2.3.2 Alpha y beta óptimo

Para calcular los valores alpha y beta óptimos son necesarias las librerías de **scipy.optimize** de (Virtanen et al., 2020) y **sklearn.metrics** de (Pedregosa et al., 2011). Estas importaciones son esenciales para el código, ya que se utilizan para optimizar los parámetros **alpha** y **beta** y para calcular la pérdida cuadrática media (MSE, por sus siglas en inglés). De ellas se importan **minimize** y **mean_squared_error** respectivamente.

Tras ello se escogen los parámetros a elección:

```
########## Elegir hasta donde se calcula la recta de regresión #########
t = 13  # Por ejemplo, los primeros 13 periodos
#########################################################################
```

En el código proporcionado, se establece el número de periodos para calcular la recta de regresión, siendo seleccionados 13 periodos. Esta elección determina la longitud de la ventana temporal utilizada para estimar la tendencia inicial del modelo Holt. Además, se activa la opción de optimización automática (**optimizado_automatico = True**), lo que implica que los parámetros alfa y beta del modelo Holt se ajustarán automáticamente durante el proceso de entrenamiento

Las funciones necesarias para encontrar los valores óptimos son las siguientes:

```
# Función para optimizar los parámetros alpha y beta
def mse_loss(params, data):
    alpha, beta = params
    model = Holt(data).fit(smoothing_level=alpha, smoothing_trend=beta, optimized=False)
    fitted_values = model.fittedvalues
    return mean_squared_error(data, fitted_values)

def optimize_alpha_beta(data, alpha_bounds=(0, 1), beta_bounds=(0, 1)):
    result = minimize(mse_loss, x0=[0.5, 0.5], args=(data,),
                      bounds=[alpha_bounds, beta_bounds])
    optimized_alpha, optimized_beta = result.x
    return optimized_alpha, optimized_beta
```

Este fragmento de código define dos funciones que se utilizan para optimizar los parámetros alfa y beta del modelo Holt.

- La función **mse_loss** calcula el error cuadrático medio (MSE) entre los datos observados y los valores ajustados por el modelo Holt. Tiene como argumentos los parámetros **params**, que son los valores de alfa y beta a optimizar, y los datos sobre los cuales se ajustará el modelo con los parámetros dados y calcula los valores ajustados. Finalmente, devuelve el MSE entre los datos observados y los valores ajustados.
- La función **optimize_alpha_beta** utiliza la función **minimize** de la librería **scipy.optimize** para encontrar los valores óptimos de alfa y beta que minimizan el MSE. Tiene de argumento la función objetivo a minimizar (error cuadrático medio) y los datos sobre los cuales se ajustará el modelo, así como los límites para los valores de alfa y beta. Por defecto, los límites se establecen en el rango [0, 1]. La función **minimize** busca los valores óptimos de alfa y beta que minimizan la función de pérdida **mse_loss**. Los valores iniciales de alfa y beta se establecen en 0.3, y se devuelve el valor óptimo de alfa y beta encontrados por el algoritmo de optimización (Jason Brownlee, 2021).

La siguiente fracción de código ejecuta y calcula la predicción del modelo:

```
pendiente, eje_ordenada, recta = regresion(train_data, t)
print("Pendiente:", pendiente)
print("Eje ordenada:", eje_ordenada)
# Inicialización  REGRESIÓN
p0=slope
D0=eje_ordenada + t * slope
# Establecer la frecuencia de los datos
train_data.index.freq = 'M'
forecast_data.index.freq = 'M'

# Optimizar alpha y beta con los datos de entrenamiento
alpha_optimo, beta_optimo = optimize_alpha_beta(train_data['unit_sales_with_slope'])

print("Alpha óptimo encontrado:", alpha_optimo)
print("Beta óptimo encontrado:", beta_optimo)

# Ajuste del modelo en datos de entrenamiento con alpha y beta óptimos
modelo =
Holt(train_data['unit_sales_with_slope'].iloc[t:]).fit(smoothing_level=alpha_optimo,
smoothing_trend=beta_optimo, optimized=False)
train_data['Holt_Optimo'] = modelo.fittedvalues

# Predicción en datos de prueba
forecast_data['Holt_Optimo'] = np.nan  # Inicializar columna con valores NaN

forecast_data['Holt_Optimo'] = modelo.forecast(int(len(forecast_data)))  # Inicializar
columna con valores NaN
modelo.summary()
```

1. Regresión lineal: Esta parte del código realiza una regresión lineal sobre los datos de entrenamiento (**train_data**) usando un método **regresion**, el cual devuelve la pendiente (**pendiente**), el **eje_ordenada** y la recta de ajuste.
2. Impresión de resultados: Se imprimirá la pendiente y el eje ordenada de la línea de regresión. La pendiente representa el cambio en la variable dependiente por unidad de cambio en la variable independiente, y el eje ordenada es el punto donde la línea cruza el eje Y.
3. Inicialización: Se inicializan las variables **p0** y **D0** usando la pendiente (**slope**) y el eje ordenada (**eje_ordenada**), **p0** es igual a la pendiente, y **D0** un valor inicial calculado usando el eje ordenada y la pendiente multiplicada por **t**.
4. Frecuencia mensual: Este código establece la frecuencia de los índices de los conjuntos de datos de entrenamiento (**train_data**) y de prueba (**forecast_data**) a mensual ('M'), lo cual es importante para los modelos de series temporales que utilizan esta información para sus cálculos.
5. Optimización de parámetros: Se optimizan los parámetros de suavización (**alpha** y **beta**) usando los datos de ventas unitarias ajustadas con la pendiente. Estos parámetros controlan el nivel de suavización y la tendencia en el modelo de Holt.
6. Resultados de optimización: Se imprimen los valores óptimos de **alpha** y **beta** encontrados para conocer los mejores parámetros para el modelo.
7. Modelo de Holt: Se ajusta un modelo de suavización exponencial de Holt a los datos de entrenamiento utilizando los parámetros óptimos.
8. Valores ajustados: Los valores ajustados del modelo se almacenan en una nueva columna en el conjunto de datos de entrenamiento (**train_data**), permitiendo evaluar cómo se ajusta el modelo a los datos históricos.
9. Inicialización de la columna de predicción: Se inicializa una columna (**forecast_data**) en el conjunto de datos de prueba para las predicciones, asegurándose de que la columna exista antes de rellenarla con valores predichos.

Se añade una nueva fila como en anteriores casos al dataframe de errores:

```
HOLT = train_data['Holt_Optimo'].to_numpy()
unit_sales = train_data['unit_sales_with_slope'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, HOLT)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[3, 'Modelo'] = 'Holt Automatico'
df_errores.loc[3, 'me'] = errores[0]
df_errores.loc[3, 'mae'] = errores[1]
df_errores.loc[3, 'mpe'] = errores[2]
df_errores.loc[3, 'mape'] = errores[3]
df_errores.loc[3, 'rmse'] = errores[4]
df_errores.loc[3, 'corr'] = errores[5]
df_errores.loc[3, 'minmax'] = errores[6]
df_errores
```

Y la respectiva representación del conjunto de datos:

```
# Representación
plt.figure(figsize=(12,6))
plt.plot(df_res['unit_sales_with_slope'],'-o',color='orange',label='Unit_sales')
plt.plot(train_data['Holt_Optimo'],'-o',color='black',label='prediccion_Holt')
plt.plot(forecast_data['Holt_Optimo'],'-o',color='green',label='forecast_Holt')
plt.plot(df_res.index[:t], recta, color='red', linestyle='-', label='Recta de Regresión')

plt.xlabel("PERIODOS",fontsize=14)
plt.ylabel("VENTAS",fontsize=14)
plt.title("SERIE",fontsize=18)
plt.legend()
plt.show()
```

6.2.2.4 Modelo Arima No-Estacional

El procedimiento seguido para la obtención de las predicciones del modelo Arima en Python está basado en (Joaquín Amat Rodrigo & Javier Escobar Ortiz, 2023), aunque antes debe ser instalada la biblioteca **pmdArima** (Taylor G. Smith and others, 2017) y después importadas las librerías que a continuación se exponen:

```
pip install pmdArima
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdArima import Arima
from pmdArima import auto_Arima
import itertools
import statsmodels.api as sm
```

Como se expone en el capítulo 9 de (Hyndman, 2021), una serie con tendencia no puede ser estacionaria y es por ello que debe ser diferenciada, la diferenciación permite la eliminación de la tendencia.

Para reconocer la estacionariedad de una serie se usa la prueba de Dickey-Fuller aumentada, con la función **adfuller()** de la biblioteca de Statsmodels, (Seabold et al., 2010).

A continuación se usa dicha función desde la serie normal hasta una serie diferenciada de segundo orden, para diferenciar se usa el método '**.diff()**' de Pandas (The pandas development team, 2020) junto a '**dropna()**' para ignorar los valores NaN :

```
# Test estacionariedad
# ================================================================================
```

```
warnings.filterwarnings("ignore")

train=train_data['unit_sales_with_slope'].copy(deep=True)
datos_r=df_res['unit_sales_with_slope'].copy(deep=True)

dif1 = train.diff().dropna()
dif2 = dif1.diff().dropna()

print('Test estacionariedad serie original')
print('-----------------------------------')
adfuller_result = adfuller(datos_r)
print(f'ADF Statistic: {adfuller_result[0]}, p-value: {adfuller_result[1]}')


print('\nTest estacionariedad de serie diferenciada orden 1')
print('--------------------------------------------------')
adfuller_result = adfuller(dif1)
print(f'ADF Statistic: {adfuller_result[0]}, p-value: {adfuller_result[1]}')


print('\nTest estacionariedad de serie diferenciada orden 2')
print('--------------------------------------------------')
adfuller_result = adfuller(dif2)
print(f'ADF Statistic: {adfuller_result[0]}, p-value: {adfuller_result[1]}')


warnings.filterwarnings("default")

# Gráfico series
# ==============================================================================
fig, axs = plt.subplots(nrows=3, ncols=1, figsize=(7, 7), sharex=True)
datos_r.plot(ax=axs[0], title='Serie original')
dif1.plot(ax=axs[1], title='Diferenciación orden 1')
dif2.plot(ax=axs[2], title='Diferenciación orden 2');
```

Una vez se obtienen los resultados impresos por pantalla es necesario saber interpretarlo, según (Burcu Özcan & Ilhan Öztürk, 2019) cuanto más negativo es el valor **ADF** más estacionaria es la serie, además teniendo de referencia el **p-value,** debe ser inferior a 0.05 para rechazar la hipótesis nula (no es estacionaria), es decir, que si alcanza un valor inferior a 0.05 se debe tomar la hipótesis alternativa que sería que es estacionaria.

Para determinar los valores de p y q se sigue la explicación del apartado 9.5 de (Hyndman, 2021) usando las gráficas **ACF** y **PACF**.

A continuación, se imprimen las gráficas **ACF**:

```
# Gráfico de autocorrelación para la serie original y la serie diferenciada
# ==============================================================================
fig, axs = plt.subplots(nrows=3, ncols=1, figsize=(7, 7), sharex=True)
plot_acf(datos_r, ax=axs[0], lags=42, alpha=0.05)
axs[0].set_title('Autocorrelación serie original')
plot_acf(dif1, ax=axs[1], lags=41, alpha=0.05)
axs[1].set_title('Autocorrelación serie diferenciada (order=1)')
plot_acf(dif2, ax=axs[2], lags=40, alpha=0.05)
axs[2].set_title('Autocorrelación serie diferenciada (order=2)')
```

A continuación, se presenta el código qque imprimirán las gráficas **PACF**:

```
# Autocorrelación parcial para la serie original y la serie diferenciada
# ==============================================================================
fig, axs = plt.subplots(nrows=3, ncols=1, figsize=(7, 7), sharex=True)
plot_pacf(datos_r, ax=axs[0], lags=26, alpha=0.05)
axs[0].set_title('Autocorrelación parcial serie original')
```

```
plot_pacf(dif1, ax=axs[1], lags=20, alpha=0.05)
axs[1].set_title('Autocorrelación parcial serie diferenciada (order=1)')
plot_pacf(dif2, ax=axs[2], lags=20, alpha=0.05)
axs[2].set_title('Autocorrelación parcial serie diferenciada (order=1)')
plt.tight_layout()
```

A continuación, según el análisis anterior se eligirán los parámetros:

```
########### Elección de parámetros #########
p=1
d=1
q=1
#############################################
```

Una vez elegidos los valores **p**, **d** y **q** se ejecuta el modelo:

```
# pmdArima
# Modelo
# ============================================================================
modelo = Arima(order=(p, d, q))
modelo.fit(y=train_data['unit_sales_with_slope'])
modelo.summary()

# Predicciones en el conjunto de entrenamiento
predicciones_entrenamiento_pdmArima = modelo.predict_in_sample()
train_data['ANE']=copy.deepcopy(predicciones_entrenamiento_pdmArima)
# Predicción
# ============================================================================
predicciones_pdmArima = modelo.predict(len(forecast_data['unit_sales_with_slope']))
forecast_data['ANE']=copy.deepcopy(predicciones_pdmArima)
```

Se ejecuta el código para conocer errores:

```
Arima = train_data['ANE'].to_numpy()
unit_sales = train_data['unit_sales_with_slope'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, Arima)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[4, 'Modelo'] = 'Arima No-Estacional pdq elegido'
df_errores.loc[4, 'me'] = errores[0]
df_errores.loc[4, 'mae'] = errores[1]
df_errores.loc[4, 'mpe'] = errores[2]
df_errores.loc[4, 'mape'] = errores[3]
df_errores.loc[4, 'rmse'] = errores[4]
df_errores.loc[4, 'corr'] = errores[5]
df_errores.loc[4, 'minmax'] = errores[6]
df_errores
```

Posteriormente se imprimirá su gráfica como en anteriores ocasiones:

```
# Representación
# ============================================================================
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_with_slope'], '-o', color='orange', label='train')
plt.plot(train_data['ANE'], '-o', color='black', label='predicciones_entrenamiento')
plt.plot(forecast_data['ANE'], '-o', color='green', label='pmdArima')
plt.xlabel("PERIODOS", fontsize=14)
plt.ylabel("VENTAS", fontsize=14)
plt.title("Predictions with Arima models", fontsize=18)
plt.legend()
```

```
plt.show()
```

Tras exponer el código para que el usuario experimente con diferentes valores de **p**, **d** y **q**, se expone a continuación el código con la función **auto_Arima** de **pmdArima**, dicha función permite obtener los mejores valores (**p**, **d**, **q**) para el conjunto de datos estudiados.

```
# Auto Arima: seleccion basada en AIC
# ======================================================================
modelo = auto_Arima(
            y                 = train_data['unit_sales_with_slope'],
            start_p           = 0,
            start_q           = 0,
            max_p             = 3,
            max_q             = 3,
            seasonal          = False,
            test              = 'adf',
            m                 = 0, # periodicidad de la estacionalidad
            d                 = 1,
            D                 = None, # El algoritmo determina 'D'
            trace             = True,
            error_action      = 'ignore',
            suppress_warnings = True,
            stepwise          = True
        )
```

Tras esto el usuario podrá volver a escoger variables como anteriormente se ha expuesto y ejecutar de nuevo el código para ver los resultados, esta vez con los mejores resultados que la función haya arrojado.

Finalmente se hace una copia de los tres principales dataframes, donde DT representa demanda con tendencia.

```
train_data_DT=copy.deepcopy(train_data)
forecast_data_DT=copy.deepcopy(forecast_data)
df_errores_DT=copy.deepcopy(df_errores)
```

6.2.3 Métodos de previsión para patrones de demanda con estacionalidad

Al igual que se hizo con los métodos de prevision para demanda con tendencia se vuelve a usar el dataframe original como al principio. Se añaden ciertos métodos extras para facilitar lo que posteriormente se hace para adaptar los valores a nuevos datos con estacionalidad y tendencia.

```
df_res = copy.deepcopy(df_res_copia)
# Convertir el índice a valores numéricos
df_res.reset_index(drop=False, inplace=True)
# Convertir la columna 'date' para que solo contenga la fecha (sin horas, minutos y
segundos)
df_res['date'] = df_res['date'].dt.date
df_res.set_index('date', inplace=True)
del(forecast_data)
del(train_data)
```

Para comprender mejor los modelos de previsión para componentes estacionales se han modificado los datos originales de ventas y se han creado nuevos datos con la componente estacional de los mismos.

El código para elegir parámetros se detalla a continuación:

```
########## Variables a elegir #######

m = 12 # Estacionalidad mensual

# Modificar la amplitud de la estacionalidad para que sea cada vez más prominente
amplitud_base = 1.5 # Amplitud base
incremento_amplitud = 0.1 # Incremento en la amplitud en cada mes
```

```
# Tendencia lineal creciente
pendiente = 10 # Pendiente positiva

# Dividir los datos en entrenamiento y prueba
valor=0.8
##################################################
```

Donde cada parámetro representa:

1. Estacionalidad mensual (m): Se define un valor de 12, lo que indica que el patrón estacional se repite cada 12 meses.
2. Amplitud base (amplitud_base) y su incremento (incremento_amplitud): Se establece una amplitud base de 1.5 para la estacionalidad y un incremento de 0.1. Esto significa que la estacionalidad será más prominente cada mes, aumentando su influencia de manera gradual.
3. Pendiente de la tendencia (pendiente): Se elige una pendiente de 10, indicando que la tendencia de la serie temporal será creciente.
4. Elegir el tamaño de los datos de entrenamiento y predicción con el párametro (valor).

El código que crea los nuevos valores es el siguiente:

```
# Obtener datos mensuales de tu DataFrame df_res
mensual_sales = df_res['unit_sales'].values

# Generar datos para la componente estacional
np.random.seed(0)

# Componente estacional mensual (simulada a partir de los datos mensuales existentes)
num_meses = len(mensual_sales)
estacionalidad_base = np.tile(mensual_sales[:m], num_meses // m + 1)[:num_meses]  # toma la
estacionalidad de los primeros 'm' meses

amplitud = amplitud_base + np.arange(num_meses) * incremento_amplitud
estacionalidad = estacionalidad_base * amplitud

# Ruido aleatorio
ruido = np.random.normal(0, 1, num_meses)

tendencia = np.arange(num_meses) * pendiente

# Combinar los componentes para formar la serie temporal
data = estacionalidad + ruido + tendencia

# Crear un DataFrame de Pandas
fechas = df_res.index  # Mantenemos las mismas fechas que en df_res
df_estacional_tendencia = pd.DataFrame({'date': fechas, 'estacional_trend_sales': data})

# Visualizar la serie temporal generada
plt.figure(figsize=(10, 6))
plt.plot(df_estacional_tendencia['date'],
df_estacional_tendencia['estacional_trend_sales'])
plt.title('Serie Temporal Estacional con Tendencia Creciente y Estacionalidad Prominente
(Datos Mensuales)')
plt.xlabel('Fecha')
plt.ylabel('Unit Sales')
plt.grid(True)
plt.show()
```

1. Obtener datos mensuales del DataFrame:
    - Se extraen los datos de ventas mensuales (**unit_sales**) del DataFrame **df_res** utilizando el

método **values**. Esto convierte la columna de pandas en un array de NumPy.

2. Generar datos para la componente estacional:
    - Fijar la semilla del generador aleatorio**: np.random.seed(0)** se utiliza para asegurar que los números aleatorios generados sean reproducibles.
    - Determinar el número de meses**: len(mensual_sales)** obtiene el número total de meses de los datos.
    - Crear el patrón estacional: **np.tile()** se utiliza para repetir los valores de ventas de los primeros **m** meses, y luego se corta (**[:num_meses]**) para ajustar el tamaño al número total de meses. Esto crea una base estacional repetitiva.
3. Calcular la amplitud y la estacionalidad**:**
    - Amplitud creciente: **np.arange(num_meses)** genera un array de números enteros desde 0 hasta **num_meses-1**. Multiplicado por **incremento_amplitud** y sumado a **amplitud_base**, produce un array donde la amplitud aumenta linealmente.
    - Aplicar la amplitud: La estacionalidad base se multiplica por este array de amplitud para obtener una estacionalidad cuya influencia crece con el tiempo.
4. Añadir ruido aleatorio:
    - **np.random.normal(0, 1, num_meses)** genera un array de ruido aleatorio con una distribución normal de media 0 y desviación estándar 1, del mismo tamaño que el número de meses. Esto añade variabilidad a los datos.
5. Generar la tendencia:
    - **np.arange(num_meses) * pendiente** genera una tendencia lineal creciente. Aquí, **np.arange(num_meses)** produce un array de números enteros, que se multiplica por la pendiente para crear una tendencia que aumenta constantemente.
6. Combinar los componentes para formar la serie temporal:
    - La estacionalidad ajustada, el ruido aleatorio y la tendencia lineal se suman para formar la serie temporal completa.
7. Crear un DataFrame con los datos generados:
    - Se crea un nuevo DataFrame de Pandas, **df_estacional_tendencia**, utilizando las fechas originales del DataFrame **df_res** y los datos generados de la serie temporal. Esto se logra mediante **pd.DataFrame({'date': fechas, 'estacional_trend_sales': data})**.
8. Visualizar la serie temporal:
    - Crear la figura: **plt.figure(figsize=(10, 6))** establece el tamaño del gráfico.
    - Graficar los datos: **plt.plot()** dibuja la serie temporal.
    - Configurar el gráfico: **plt.title()**, **plt.xlabel()**, **plt.ylabel()** y **plt.grid(True)** añaden el título, etiquetas de los ejes y la cuadrícula al gráfico.
    - Mostrar el gráfico: **plt.show()** muestra el gráfico en pantalla.

```
# Convertir el índice de df_estacional_tendencia a DatetimeIndex
df_estacional_tendencia.set_index('date', inplace=True)

# Asignar la columna
df_res['unit_sales_season_trend'] =
copy.deepcopy(df_estacional_tendencia['estacional_trend_sales'])


train_size = int(len(df_res) * valor)
train_data, forecast_data = copy.deepcopy(df_res[:train_size]),
copy.deepcopy(df_res[train_size:])
```

```
# Establecer la frecuencia de los datos
train_data.index.freq = 'M'
forecast_data.index.freq = 'M'
```

6.2.3.1 Holt-Winters

Para recrear el modelo de Holt-Winters en Python es necesario importar la función **ExponentialSmoothing** de la librería **statsmodels.tsa.holtwinters**.

Una vez importada la función, este apartado se va a dividir en una parte manual donde el usuario podrá escoger los valores de los parámetros alfa, beta y gamma, una con el modelo aditivo y otra con el modelo multiplicativo.

6.2.3.1.1 Modo Manual

En el modo manual se define la función **HWintersManual**:

```
def HWintersManual(train, len_test, variable, tendencia, estacionalidad, alpha=None,
beta=None, gamma=None):

    # En este método de holt-winters:
    modelo = ExponentialSmoothing(np.asarray(train[variable]), seasonal_periods=12,
trend=tendencia, seasonal=estacionalidad)
    modelo = modelo.fit(smoothing_level=alpha, smoothing_slope=beta,
smoothing_seasonal=gamma,)
    training=modelo.fittedvalues
    print("Datos del entrenamiento", modelo.summary())
    forecast = modelo.forecast(len_test)
    return forecast,training
```

1. Definición de la función: La función toma varios parámetros:
    - **train**: Un DataFrame que contiene los datos de entrenamiento.
    - **len_test**: Un entero que indica la longitud de la serie de prueba (o el horizonte de predicción).
    - **variable**: El nombre de la columna del DataFrame que contiene la serie temporal a modelar.
    - **tendencia**: Un string que especifica el tipo de tendencia a modelar (puede ser 'add', 'mul' o None).
    - **estacionalidad**: Un string que especifica el tipo de estacionalidad a modelar (puede ser 'add', 'mul' o None).
    - **alpha**, **beta**, **gamma**: Parámetros opcionales de suavizado que corresponden al nivel, tendencia y estacionalidad, respectivamente. Si no se proporcionan, se ajustarán automáticamente.
2. Creación del modelo: Se crea un modelo de suavizado exponencial utilizando la clase **ExponentialSmoothing** de la librería **statsmodels**. Este modelo se ajusta a los datos de entrenamiento (**train**) de la columna especificada por **variable**.
    - **seasonal_periods=12** indica que se espera una estacionalidad anual en los datos (por ejemplo, datos mensuales con un ciclo de 12 meses).
    - **trend** y **seasonal** se configuran según los parámetros proporcionados (**tendencia** y **estacionalidad**).
3. Ajuste del modelo: Se ajusta el modelo a los datos de entrenamiento utilizando el método **fit()**, que también toma los parámetros de suavizado (**alpha**, **beta**, **gamma**) si se han proporcionado. Estos parámetros controlan la velocidad con la que se ajusta el modelo a los nuevos datos.
4. Valores ajustados del entrenamiento: Se obtienen los valores ajustados del modelo para los datos de entrenamiento a través de **modelo.fittedvalues**.
5. Resumen del modelo: Se imprime un resumen del modelo ajustado utilizando **modelo.summary()**, lo

cual proporciona una visión detallada de los parámetros del modelo y su desempeño en los datos de entrenamiento.

6. Predicción: Se realizan predicciones para los próximos **len_test** períodos utilizando el método **forecast()** del modelo ajustado.
7. Retorno de Resultados: La función devuelve dos elementos:
    - **forecast**: Las predicciones para los próximos **len_test** períodos.
    - **training**: Los valores ajustados del modelo para los datos de entrenamiento.

En parámetros a elegir se encuentran alfa, beta y gamma:

```
############## Valores a elegir ############
alpha = 0.13
beta = 0.01
gamma = 0.02
tendencia='additive'
estacionalidad='additive'
###################################
```

La ejecución en este caso es simplemente la llamada a la función en el orden anteriormente mostrado en su definición:

```
# Aplicar el modelo de Holt-Winters usando la función HWintersManual
forecast_data['HW_MANUAL'],train_data['HW_MANUAL'] = (
    HWintersManual(train_data, len(forecast_data),
                   'unit_sales_season_trend',
                   tendencia,estacionalidad,alpha=alpha,
                   beta=beta, gamma=gamma))
```

Tras ello le sigue el análisis de errores:

```
# Dataframe para errores
df_errores = pd.DataFrame()

Hwinters = train_data['HW_MANUAL'].to_numpy()
unit_sales = train_data['unit_sales_season_trend'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, Hwinters)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[0, 'Modelo'] = 'Holt-Winters Manual'
df_errores.loc[0, 'me'] = errores[0]
df_errores.loc[0, 'mae'] = errores[1]
df_errores.loc[0, 'mpe'] = errores[2]
df_errores.loc[0, 'mape'] = errores[3]
df_errores.loc[0, 'rmse'] = errores[4]
df_errores.loc[0, 'corr'] = errores[5]
df_errores.loc[0, 'minmax'] = errores[6]
df_errores
```

Y su respectiva representación:

```
# Visualizar resultados
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_season_trend'],'-o',  label='Training data',color='orange')
plt.plot(train_data.index,train_data['HW_MANUAL'],'-o' ,
label='Entrenamiento',color='black')
plt.plot(forecast_data.index, forecast_data['HW_MANUAL'],'-o', label='Predictions',
color='green')
plt.legend()
```

```
plt.title('Holt-Winters Forecasting')
plt.xlabel('Date')
plt.ylabel('Unit Sales')
plt.show()
```

6.2.3.1.2 Modelo Aditivo

La función necesaria para el modelo aditivo es **HWinters**:

```
def HWinters(train, len_test, variable,tendencia,estacionalidad):
    # En este método de holt-winters:
    modelo = ExponentialSmoothing(np.asarray(train[variable]), seasonal_periods=12,
                                  trend=tendencia, seasonal=estacionalidad).fit()
    training=modelo.fittedvalues
    print("Datos del entrenamiento", modelo.summary())
    forecast = modelo.forecast(len_test)
    return forecast,training
```

El principio es el mismo que el modo manual salvo que **ExponentialSmoothing** calcula los mejores valores para los parámetros.

La llamada a la función es:

```
# Aplicar el modelo de Holt-Winters usando la función HWinters
forecast_data['HW_AD'],train_data['HW_AD'] = (
    HWinters(train_data, len(forecast_data),
             'unit_sales_season_trend',
             'additive','additive'))
```

El código para añadir una fila más a los errores:

```
# Dataframe para errores
Hwinters = train_data['HW_AD'].to_numpy()
unit_sales = train_data['unit_sales_season_trend'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, Hwinters)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[1, 'Modelo'] = 'Holt-Winters aditivo'
df_errores.loc[1, 'me'] = errores[0]
df_errores.loc[1, 'mae'] = errores[1]
df_errores.loc[1, 'mpe'] = errores[2]
df_errores.loc[1, 'mape'] = errores[3]
df_errores.loc[1, 'rmse'] = errores[4]
df_errores.loc[1, 'corr'] = errores[5]
df_errores.loc[1, 'minmax'] = errores[6]
df_errores
```

Y su representación gráfica es la siguiente:

```
# Visualizar resultados
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_season_trend'],'-o',  label='Training data',color='orange')
plt.plot(train_data.index,train_data['HW_AD'],'-o' , label='Entrenamiento',color='black')
plt.plot(forecast_data.index, forecast_data['HW_AD'],'-o', label='Predictions',
color='green')
plt.legend()
plt.title('Holt-Winters Forecasting')
plt.xlabel('Date')
plt.ylabel('Unit Sales')
plt.show()
```

6.2.3.1.3 Modelo Multiplicativo

La función en el modelo multiplicativo es la misma que en el modelo aditivo, la única diferencia recae en su llamada, en los parámetros de tendencia y estacionalidad se especifica que es multiplicativo.

```
# Aplicar el modelo de Holt-Winters usando la función HWinters
forecast_data['HW_MUL'],train_data['HW_MUL'] = HWinters(
  train_data, len(forecast_data),
 'unit_sales_season_trend','multiplicative','multiplicative')
```

El código de errores:

```
# Dataframe para errores
Hwinters = train_data['HW_MUL'].to_numpy()
unit_sales = train_data['unit_sales_season_trend'].to_numpy()

# Se usa la función métricas
errores = metricas(unit_sales, Hwinters)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[2, 'Modelo'] = 'Holt-Winters multiplicativo'
df_errores.loc[2, 'me'] = errores[0]
df_errores.loc[2, 'mae'] = errores[1]
df_errores.loc[2, 'mpe'] = errores[2]
df_errores.loc[2, 'mape'] = errores[3]
df_errores.loc[2, 'rmse'] = errores[4]
df_errores.loc[2, 'corr'] = errores[5]
df_errores.loc[2, 'minmax'] = errores[6]
df_errores
```

Y su respectiva representación gráfica como en anteriores ocasiones:

```
# Visualizar resultados
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_season_trend'],'-o',  label='Training data',color='orange')
plt.plot(train_data.index,train_data['HW_MUL'],'-o' , label='Entrenamiento',color='black')
plt.plot(forecast_data.index, forecast_data['HW_MUL'],'-o', label='Predictions',
color='green')
plt.legend()
plt.title('Holt-Winters Forecasting')
plt.xlabel('Date')
plt.ylabel('Unit Sales')
plt.show()
```

6.2.3.2 Modelo Arima Estacional

Como anteriormente en el modelo No-Estacional, el procedimiento seguido es el expuesto en (Joaquín Amat Rodrigo & Javier Escobar Ortiz, 2023). A continuación se expone un caso particular donde la demanda es estacional y por tanto el modelo tiene 4 parámetros más que son **P**, **D**, **Q** y **m**.

Se vuelve hacer el estudio de la estacionariedad de la misma manera que en el modelo no-estacional, aunque esta vez, dado que existe un parámetro de diferenciación más a determinar llamado **D**, es necesario diferenciar la serie con los parámetros estacionales usando la función de Pandas **diff(m)**, se le aplica el mismo razonamiento que a **d**.

```
import warnings
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

# Test estacionariedad
# ==============================================================================
warnings.filterwarnings("ignore")
```

```
train = train_data['unit_sales_season_trend'].copy(deep=True)
datos_r = df_res['unit_sales_season_trend'].copy(deep=True)

# Aplicar diferencias
dif1 = train.diff().dropna()
dif2 = dif1.diff().dropna()

# Aplicar diferencia estacional (suponiendo m=12 para datos mensuales)
m = 12
estacional_dif = train.diff(m).dropna()
estacional_dif1 = estacional_dif.diff().dropna()
estacional_dif2 = estacional_dif1.diff().dropna()

# Prueba de DF-Aumentada
print('Test estacionariedad serie completa original')
print('-------------------------------------')
adfuller_result = adfuller(datos_r)
print(f'Estadístico ADF: {adfuller_result[0]}, p-value: {adfuller_result[1]}')

print('\nTest estacionariedad de serie diferenciada primer orden')
print('-------------------------------------------------')
adfuller_result = adfuller(dif1)
print(f'Estadístico ADF: {adfuller_result[0]}, p-value: {adfuller_result[1]}')

print('\nTest estacionariedad de serie diferenciada orden 2')
print('-------------------------------------------------')
adfuller_result = adfuller(dif2)
print(f'Estadístico ADF: {adfuller_result[0]}, p-value: {adfuller_result[1]}')

print('\nTest estacionariedad para serie diferenciada estacional m=12')
print('-------------------------------------------------')
adfuller_result = adfuller(estacional_dif)
print(f'Estadístico ADF: {adfuller_result[0]}, p-value: {adfuller_result[1]}')

print('\nTest estacionariedad para serie con diferenciada estacional y diferenciada orden
1')
print('-------------------------------------------------')
adfuller_result = adfuller(estacional_dif1)
print(f'Estadístico ADF: {adfuller_result[0]}, p-value: {adfuller_result[1]}')

print('\nTest estacionariedad para serie con diferencia estacional y diferenciada orden 2')
print('-------------------------------------------------')
adfuller_result = adfuller(estacional_dif2)
print(f'Estadístico ADF: {adfuller_result[0]}, p-value: {adfuller_result[1]}')

warnings.filterwarnings("default")

# Gráfico series
# ==============================================================================
fig, axs = plt.subplots(nrows=6, ncols=1, figsize=(10, 12), sharex=True)
datos_r.plot(ax=axs[0], title='Serie original')
dif1.plot(ax=axs[1], title='Diferenciación orden 1')
dif2.plot(ax=axs[2], title='Diferenciación orden 2')
estacional_dif.plot(ax=axs[3], title='Diferenciación estacional (m=12)')
estacional_dif1.plot(ax=axs[4], title='Diferenciación estacional y orden 1')
estacional_dif2.plot(ax=axs[5], title='Diferenciación estacional y orden 2')

plt.tight_layout()
```

```
plt.show()
```

Se debe hacer el análisis con los datos diferenciados que tengan más negativo el valor **ADF** y menor **p-value** tal y como se expone en la teoría.

El código para imprimir la gráfica **ACF**:

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# Gráfico de autocorrelación para la serie original y la serie diferenciada
# ================================================================================
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(7, 7), sharex=True)
plot_acf(datos_r, ax=axs[0], lags=42, alpha=0.05)
axs[0].set_title('Autocorrelación serie original')
plot_acf(dif1, ax=axs[1], lags=41, alpha=0.05)
axs[1].set_title('Autocorrelación serie diferenciada (order=1)')
```

El código para imprimir la gráfica **PACF**:

```
# Autocorrelación parcial para la serie original y la serie diferenciada
# ================================================================================
fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(7, 7), sharex=True)
plot_pacf(datos_r, ax=axs[0], lags=26, alpha=0.05)
axs[0].set_title('Autocorrelación parcial serie original')
plot_pacf(dif1, ax=axs[1], lags=20, alpha=0.05)
axs[1].set_title('Autocorrelación parcial serie diferenciada orden 1')
plt.tight_layout()
```

A diferencia con el método no-estacional se añade la elección de los parámetros **P**, **D**, **Q** y **m**.

```
########### Elección de parámetros #########
p=2
d=1
q=2
P=1
D=2
Q=1
m=12
#############################################
```

Se ejecuta el modelo:

```
# pmdArima
from pmdArima import Arima
from pmdArima import auto_Arima

# Modelo
modelo = Arima(order=(p,d,q), seasonal_order=(P, D, Q, m))
modelo.fit(y=train_data['unit_sales_season_trend'])
modelo.summary()

# Predicciones en el conjunto de entrenamiento
predicciones_entrenamiento_pdmArima = modelo.predict_in_sample()
train_data['Arima_ESTACIONAL']=predicciones_entrenamiento_pdmArima.copy(deep=True)
# Predicción
# ================================================================================
predicciones_pdmArima = modelo.predict(len(forecast_data['unit_sales_season_trend']))
forecast_data['Arima_ESTACIONAL']=copy.deepcopy(predicciones_pdmArima)
```

Se añade la última fila de errores:

```
# Dataframe errores
Arima = train_data['Arima_ESTACIONAL'].to_numpy()
unit_sales = train_data['unit_sales_season_trend'].to_numpy()
```

```
# Se usa la función métricas
errores = metricas(unit_sales, Arima)
# Redondear los valores de la tupla a dos decimales
errores = tuple(round(e, 3) for e in errores)
# Asignación de los valores de la tupla a las columnas del DataFrame
df_errores.loc[3, 'Modelo'] = 'Arima'
df_errores.loc[3, 'me'] = errores[0]
df_errores.loc[3, 'mae'] = errores[1]
df_errores.loc[3, 'mpe'] = errores[2]
df_errores.loc[3, 'mape'] = errores[3]
df_errores.loc[3, 'rmse'] = errores[4]
df_errores.loc[3, 'corr'] = errores[5]
df_errores.loc[3, 'minmax'] = errores[6]
df_errores
```

Se representa la gráfica con sus respectivos valores de entrenamiento y valores predichos:

```
# Representación
# ============================================================================
plt.figure(figsize=(12, 6))
plt.plot(df_res['unit_sales_season_trend'], '-o', color='orange', label='train')
plt.plot(train_data['Arima_ESTACIONAL'], '-o', color='black',
label='predicciones_entrenamiento')
plt.plot(forecast_data['Arima_ESTACIONAL'], '-o', color='green', label='pmdArima')
plt.xlabel("PERIODOS", fontsize=14)
plt.ylabel("VENTAS", fontsize=14)
plt.title("Predictions with Arima models", fontsize=18)
plt.legend()
plt.show()
```

Se expone a continuación el código con la función auto_Arima de pmdArima, dicha función permite obtener los mejores valores (**p**, **d**, **q**) y también de (**P**, **D**, **Q**) para el conjunto de datos estudiados.

```
# Auto Arima: seleccion basada en AIC
# ============================================================================
modelo = auto_Arima(
            y                 = df_res['unit_sales_season_trend'],
            start_p           = 0,
            start_q           = 0,
            max_p             = 3,
            max_q             = 3,
            seasonal          = True,
            test              = 'adf',
            m                 = 12, # periodicidad de la estacionalidad
            d                 = 1,
            D                 = None, # El algoritmo determina 'D'
            trace             = True,
            error_action      = 'ignore',
            suppress_warnings = True,
            stepwise          = True
        )
```

La estructura a seguir a la hora del desarrollo ha sido igual en cada método aunque particularizando las funciones para cada modelo de prevision, en cada método de previsión se ha incorporado gráficas y tabla de errores para facilitar la comprensión y análisis de resultados. Salvo para los métodos Arima donde la cantidad de parámetros a elegir aumenta la dificultad y por tanto deben ser analizadas más variables para elegir los valores adecuados de cada método particular.El principal inconveniente recae en la cantidad de potencia computacional que se requiere en los primeros pasos para obtener el dataframe con los valores de ventas de cada día, demorándose más tiempo a la hora de ejecutar los primeros bloques de código, es por ello que agragar mensualmente la demanda ayuda posteriormente en los tiempos de análisis.

7 RESULTADOS

Los gráficos y tablas que son resultados de las ejecuciones de código serán expuestos en este apartado con sus respectivas interpretaciones y explicaciones.

7.1. Descripción del caso

Como se ha comentado en el caso de estudio los datos descargados tal y como se expone en las tablas y figuras de abajo, es una demanda de tipo nivelada sin ningún patron de tendencia o estacionalidad, es por ello que tal y como se extraen los datos solo será útil para el caso de la demanda nivelada. La Tabla 7-1 muestra los valores del dataframe denominado df_res que contiene tanto fechas como unidades vendidas.

date	unit_sales
2013-01-31 00:00:00	84.0
2013-02-28 00:00:00	180.0
2013-03-31 00:00:00	155.0
2013-04-30 00:00:00	110.0
2013-05-31 00:00:00	89.0
2013-06-30 00:00:00	94.0
2013-07-31 00:00:00	61.0
2013-08-31 00:00:00	114.0
2013-09-30 00:00:00	92.0
2013-10-31 00:00:00	109.0
2013-11-30 00:00:00	82.0
2013-12-31 00:00:00	202.0
2014-01-31 00:00:00	136.0
2014-02-28 00:00:00	149.0
2014-03-31 00:00:00	160.0
2014-04-30 00:00:00	132.0
2014-05-31 00:00:00	82.0
2014-06-30 00:00:00	84.0
2014-07-31 00:00:00	112.0
2014-08-31 00:00:00	120.0
2014-09-30 00:00:00	116.0
2014-10-31 00:00:00	140.0
2014-11-30 00:00:00	146.0
2014-12-31 00:00:00	358.0
2015-01-31 00:00:00	223.0
2015-02-28 00:00:00	218.0
2015-03-31 00:00:00	171.0
2015-04-30 00:00:00	217.0
2015-05-31 00:00:00	180.0
2015-06-30 00:00:00	120.0
2015-07-31 00:00:00	121.0
2015-08-31 00:00:00	146.0
2015-09-30 00:00:00	105.0
2015-10-31 00:00:00	167.0
2015-11-30 00:00:00	128.0
2015-12-31 00:00:00	201.0
2016-01-31 00:00:00	280.0
2016-02-29 00:00:00	263.0
2016-03-31 00:00:00	214.0
2016-04-30 00:00:00	19.0
2016-05-31 00:00:00	114.0
2016-06-30 00:00:00	79.0
2016-07-31 00:00:00	59.0
2016-08-31 00:00:00	13.0
2016-09-30 00:00:00	0.0
2016-10-31 00:00:00	32.0
2016-11-30 00:00:00	115.0
2016-12-31 00:00:00	184.0
2017-01-31 00:00:00	179.0
2017-02-28 00:00:00	135.0
2017-03-31 00:00:00	151.0
2017-04-30 00:00:00	187.0
2017-05-31 00:00:00	100.0
2017-06-30 00:00:00	83.0

Tabla 7-1 Dataframe con los valores extraídos.

En la Figura 7-1 se puede observar que dicha demanda posee un componente nivelado bastante alto sin tendencia ni estacionalidad aparentes.

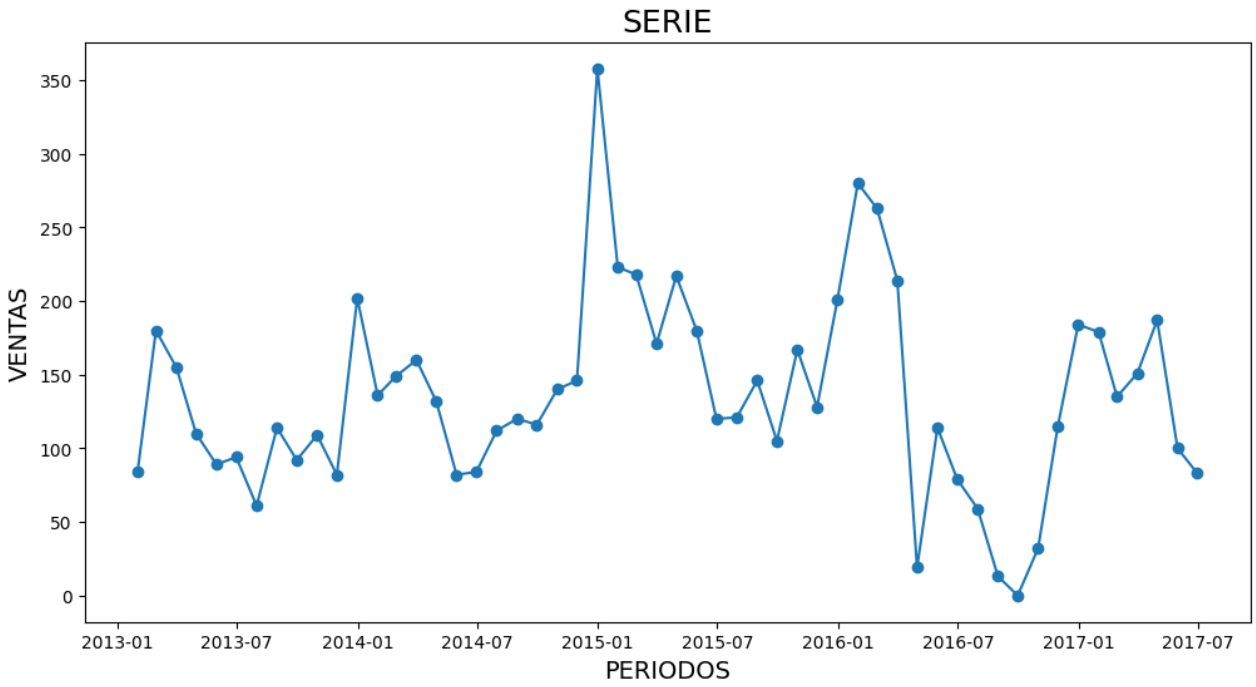

Figura 7-1 Representación de la demanda de los datos extraídos.

Para obtener una visión más global de los datos se han añadido unos gráficos donde se puede ver tanto la tendencia como la estacionalidad aproximada que capta la función **seasonal_decompose** de **statsmodel**.

La Figura 7-2 muestra la descomposición de una serie temporal de ventas unitarias (unit_sales) en sus componentes: tendencia (Trend), estacionalidad (Seasonal) y residuales (Resid). A continuación, se describe y comenta cada uno de estos componentes:

1. **Serie original (unit_sales)**:
    - La serie original muestra las ventas unitarias a lo largo del tiempo, desde mediados de 2013 hasta principios de 2017. Se observan fluctuaciones regulares con algunos picos notables, especialmente hacia finales de 2014 y mediados de 2015.
2. **Tendencia (Trend)**:
    - La componente de tendencia revela la dirección general de la serie a lo largo del tiempo, eliminando las variaciones estacionales y residuales. En este caso, se aprecia un incremento gradual en las ventas unitarias hasta aproximadamente 2016, seguido de una disminución hasta principios de 2017. Este patrón indica un crecimiento inicial en las ventas que luego empieza a declinar.
3. **Estacionalidad (Seasonal)**:
    - La componente estacional muestra las fluctuaciones periódicas y recurrentes en la serie temporal. Aquí, se intuye un patrón estacional claro con picos y valles que se repiten aproximadamente cada año. Este comportamiento sugiere que hay factores estacionales que afectan las ventas, como pueden ser promociones anuales, cambios estacionales en la demanda, o eventos específicos del mercado.
4. **Residuales (Resid)**:
    - La componente residual captura las variaciones que no pueden ser explicadas por la tendencia ni por la estacionalidad. Los residuales parecen estar centrados alrededor de cero, con dispersiones moderadas y sin un patrón claro. Esto indica que, después de eliminar la tendencia y la estacionalidad, las fluctuaciones restantes son esencialmente aleatorias y no siguen un patrón discernible.

La descomposición de la serie temporal permite entender mejor los factores que influyen en las ventas

unitarias. La tendencia muestra un aumento seguido de una disminución, reflejando posibles cambios en el mercado o en la empresa. La estacionalidad indica patrones repetitivos anuales que deben ser considerados en la planificación y previsión de ventas. Los residuales sugieren que, aunque hay variaciones no explicadas, éstas no siguen un patrón específico, lo que es esperable en datos reales debido a la presencia de ruido o factores impredecibles.

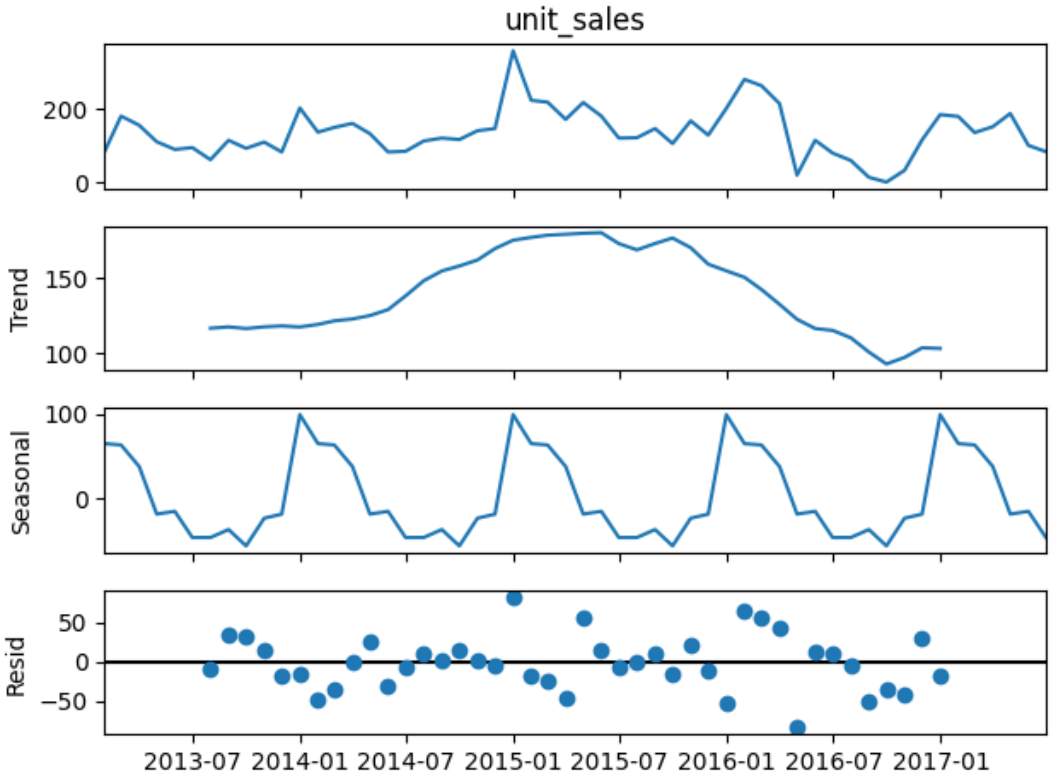

Figura 7-2 Descomposición en tendencia y estacionalidad.

Estos son los datos y las gráficas del caso descargado y de los valores obtenidos del archivo 'train.csv'. Para cada patron de demanda con tendencia o estacionalidad se modificarán y adaptarán al patron de demanda estudiado.

7.1.1 Implementación de la resolución y análisis del resultado

Los resultados expuestos en cada apartado serán los valores de los errores de prevision obtenidos en el entrenamiento con sus respectivas gráficas modeladas en el tiempo.

7.1.1.1 Métodos de prevision para patrones de demanda constante

Para patrones de demanda constante se han usado los valores tal y como fueron extraídos del archivo ya que en el tramo estudiado se puede observar que la tendencia es prácticamente nula.

Aquí se exponen las tablas con los valores de los errores que se han obtenido mediante la función de errores **metricas()**:

index	Modelo	me	mae	mpe	mape	rmse	corr	minmax
0	Media Móvil Simple	4.032	48.019	0.402	0.609	69.171	0.273	0.259
1	Media Móvil Simple Ponderada	2.333	41.685	0.340	0.533	62.563	0.468	0.230

2	Ajuste Exponencial Simple Modo Manual	-4.618	44.308	0.283	0.512	64.391	0.267	0.243
3	Ajuste Exponencial Simple Alpha óptimo	0.483	41.381	0.304	0.507	61.729	0.446	0.230

Tabla 7-2 Dataframe de errores de todos los modelos de previsión df_errores_DC.

7.1.1.1.1 Medias Móviles

En medias móviles se diferencian dos tipos diferentes, la media móvil simple y la media móvil ponderada.

7.1.1.1.1.1 Media Móvil Simple

En el código desarrollado en Python se puede escoger los diferentes valores de **N** según se considere oportunos e ir comparando con los errores, el valor escogido para este caso se trata de **N=4**, dado que los últimos valores toman más fuerza, tiene más sentido que tome valores de los últimos 4 meses. También es aquí donde se puede escoger el porcentaje de valores de entrenamiento y predicción que tendrá para el resto de modelos, en este caso se ha escogido el 80% para entrenamiento.

La Tabla 7-2 es la que muestra en la fila cero los valores de los errores para la media móvil simple. Se puede interpretar lo siguiente de los resultados:

1. **ME (Modelo Error) = 4.032**

Representa el error en el modelo, específicamente el valor ME.

2. **MAE (Error Absoluto Medio) = 48.019**

Indica el error promedio absoluto entre las predicciones del modelo y los valores observados. Un MAE de 48.019 sugiere que, en promedio, las predicciones del modelo se desvían en 48.019 unidades de las observaciones reales.

3. **MPE (Error Porcentual Medio) = 0.402**

Mide el error promedio en términos porcentuales. Un MPE de 0.402 (o 40.2%) indica que, en promedio, el modelo tiene un sesgo del 40.2%, es decir, las predicciones tienden a sobrestimar o subestimar en un 40.2%.

4. **MAPE (Error Porcentual Absoluto Medio) = 0.609**

Es el error absoluto promedio en términos porcentuales. Un MAPE de 0.609 (o 60.9%) indica que, en promedio, las predicciones del modelo están desviadas en un 60.9% de las observaciones reales.

5. **RMSE (Error Cuadrático Medio) = 69.171**

Mide la raíz cuadrada del error cuadrático medio, proporcionando una medida de la dispersión de los errores. Un RMSE de 69.171 indica una variabilidad significativa en las predicciones del modelo.

6. **Correlación (corr) = 0.273**

Mide la correlación entre los valores observados y los predichos por el modelo. Una correlación de 0.273 sugiere una relación moderada entre las predicciones del modelo y los valores reales.

7. **MinMax = 0.259**

El error MinMax compara el error de predicción con el rango de los datos observados. Un valor de 0.259 indica que el error de predicción es aproximadamente el 25.9% del rango total de los datos, proporcionando una medida del tamaño relativo del error.

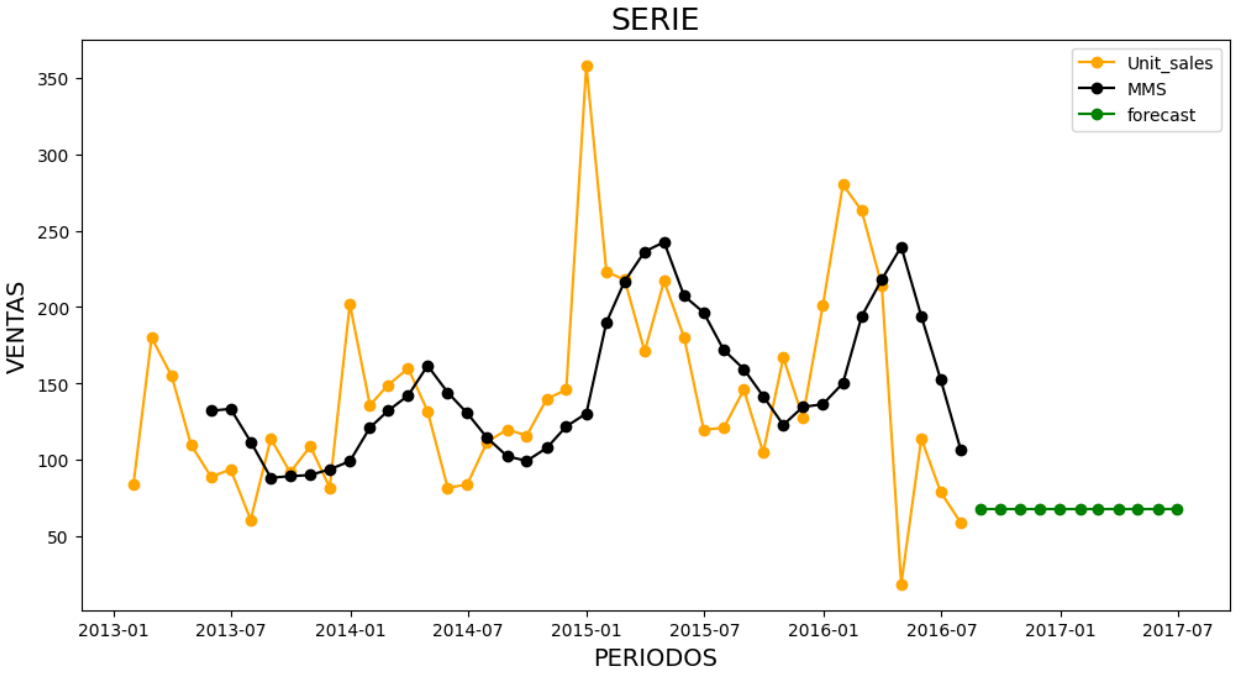

Figura 7-3 Representación Media Móvil Simple.

7.1.1.1.1.2 Media Móvil Ponderada

Además de escoger el tamaño de la ventana **N** se podrán añadir los pesos que se consideren oportunos a cada posición de la demanda, pudiendo así comprobar las diferencias entre dar más valor a los últimos valores de demanda o darle menos a estos. En el caso expuesto se ha vuelto a dar a **N** el valor 4 y los pesos han sido pesos= [0.2,0.3,1,2.5], de tal forma que se ha dado más pesos a los últimos valores registrados.

La Tabla 7-2 es la que muestra en la fila uno los valores de los errores para la media móvil ponderada. Se puede interpretar lo siguiente de los resultados:

1. **ME (Mean Error) = 2.33333333**

El Mean Error (ME) es la diferencia promedio entre los valores observados y los predichos. En este caso, un ME de 2.33 indica que, en promedio, las predicciones del modelo están por encima de las observaciones reales en 2.33 unidades. Este valor positivo sugiere un pequeño sesgo de sobreestimación en el modelo.

2. **MAE (Mean Absolute Error) = 41.6846154**

El MAE mide el error promedio absoluto entre las observaciones y las predicciones. Un MAE de 41.68 indica que, en promedio, las predicciones se desvían 41.68 unidades de los valores reales. Este error es considerablemente alto, lo que sugiere que las predicciones no son muy precisas.

3. **MPE (Mean Percentage Error) = 0.34008581**

El MPE proporciona una medida del error promedio en términos porcentuales. Un MPE de 0.34 (34%) sugiere que el modelo, en promedio, sobrestima las predicciones en un 34%. Este valor es indicativo de un sesgo hacia la sobreestimación, aunque es menor en comparación con el modelo de media móvil simple.

4. **MAPE (Mean Absolute Percentage Error) = 0.53295578**

El MAPE mide el error absoluto promedio en términos porcentuales. Un MAPE de 53.30% es bastante elevado, indicando que las predicciones del modelo, en promedio, se desvían en un 53.30% de las observaciones reales. Esto muestra que el modelo tiene una precisión moderadamente baja.

5. **RMSE (Root Mean Squared Error) = 62.5632178**

El RMSE proporciona una medida de la dispersión de los errores. Un RMSE de 62.56 sugiere una variabilidad considerable en las predicciones. Esta métrica es sensible a los errores grandes, por lo que un valor alto indica que hay predicciones con errores significativos.

6. **Correlación (corr) = 0.46799482**

La correlación mide la relación lineal entre las predicciones y los valores observados. Una correlación de 0.47 indica una relación moderada entre las predicciones y las observaciones reales. Este valor sugiere que el modelo captura parte de la estructura de los datos, pero aún hay margen para mejoras.

7. **Error MinMax = 0.22983185**

El Error MinMax compara el error de predicción con el rango de los datos observados. Un valor de 0.23 sugiere que el error de predicción es aproximadamente el 23% del rango total de los datos. Esto indica que el error es relativamente moderado en comparación con la variabilidad de los datos.

Comparación con el Modelo de Media Móvil Simple

- **Precisión mejorada**: La media móvil simple ponderada tiene un ME (2.33) y un MPE (0.34) menores que la media móvil simple (ME: 4.03, MPE: 48.02), lo que sugiere que la ponderación ha reducido el sesgo de sobreestimación.
- **MAE y MAPE más bajos**: Los valores de MAE (41.68) y MAPE (53.30%) son inferiores a los del modelo simple, indicando una mejora en la precisión.
- **RMSE menor**: Un RMSE de 62.56 es menor que el del modelo simple (69.17), lo que sugiere que la variabilidad de los errores ha disminuido.
- **Mayor correlación**: La correlación de 0.47 en el modelo ponderado es más alta que la del modelo simple (0.27), lo que indica una mejor relación lineal entre las predicciones y los valores observados.
- **Mejor error MinMax**: Un Error MinMax de 0.23 es más bajo que el del modelo simple (0.26), indicando que el error relativo al rango de los datos ha mejorado.

Conclusión

En particular, el modelo de media móvil ponderada redujo el error absoluto medio (MAE) de 48.019 a 41.685, lo que representa una disminución del 13.18%. El error cuadrático medio (RMSE) también disminuyó de 69.171 a 62.563, una reducción del 9.56%. La correlación entre los valores predichos y los valores reales aumentó significativamente de 0.273 a 0.468, lo que indica una mayor precisión en las predicciones. Además, el error porcentual medio absoluto (MAPE) disminuyó de 0.609 a 0.533, lo que sugiere que las predicciones del modelo ponderado son más precisas en términos relativos.

Estas mejoras sugieren que el modelo de media móvil ponderada es más eficaz para capturar las tendencias y patrones en los datos analizados, lo que lo hace más adecuado para aplicaciones en los datos estudiados. Sin embargo, es importante tener en cuenta que los pesos utilizados en este modelo fueron seleccionados empíricamente, por lo que futuras investigaciones podrían explorar métodos más sistemáticos para la determinación de pesos óptimos y evaluar la robustez del modelo en diferentes conjuntos de datos.

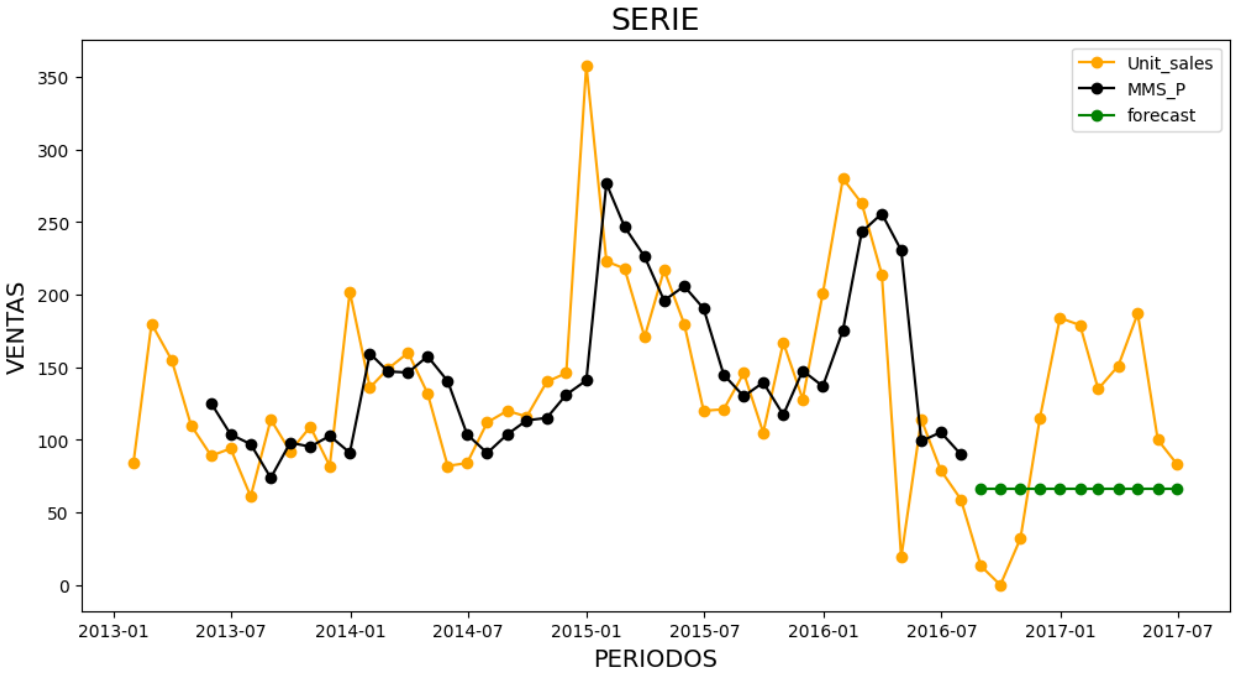

Figura 7-4 Representación de Media Móvil Simple Ponderada.

7.1.1.1.2 Ajuste Exponencial Simple

Para el Ajuste Exponencial Simple se han creado dos opciones en Google Colab, un modo manual donde se puede experimentar con el valor de alfa y un modo automático para conocer el valor óptimo de alfa, se calcula mediante la función **SimpleExpSmoothing.**

7.1.1.1.2.1 Modo Manual

En la Figura 7-5 se muestra la gráfica con los valores de entrenamiento y predicción del modo manual junto a los valores reales superpuestos.

En este modelo se puede variar y experimentar con el valor de alfa viendo así como varían los errores de los mismos. Para este caso expuesto se ha tomado alfa = 0.2, a diferencia de la media móvil ponderada se le ha dado más peso a los valores antiguos para compararla con la media móvil simple.

La Tabla 7-2 es la que muestra en la fila dos los valores de los errores para el Ajuste Exponencial Simple en modo manual. Se puede interpretar lo siguiente de los resultados:

1. **ME (Error Medio) = - 4.61828616**

El Error Medio (ME) es la diferencia promedio entre los valores observados y los predichos. En este caso, un ME de -4.62 indica que, en promedio, las predicciones del modelo están por debajo de las observaciones reales en 4.62 unidades. Este valor negativo sugiere un sesgo de subestimación en el modelo.

2. **MAE (Mean Absolute Error) = 44.307906**

El MAE mide el error promedio absoluto entre las observaciones y las predicciones. Un MAE de 44.31 indica que, en promedio, las predicciones se desvían 44.31 unidades de los valores reales.

3. **MPE (Mean Percentage Error) = 0.28326918**

El MPE proporciona una medida del error promedio en términos porcentuales. Un MPE de 0.28 (28.33%) sugiere que el modelo, en promedio, subestima las predicciones en un 28.33%. Este valor es indicativo de un sesgo hacia la subestimación.

4. **MAPE (Mean Absolute Percentage Error) = 0.51206788**

El MAPE mide el error absoluto promedio en términos porcentuales. Un MAPE de 51.21% es bastante elevado, indicando que las predicciones del modelo, en promedio, se desvían en un 51.21% de las observaciones reales. Esto muestra que el modelo tiene una precisión moderadamente baja, aunque es ligeramente mejor que el modelo ponderado.

5. **RMSE (Root Mean Squared Error) = 64.3910346**

El RMSE proporciona una medida de la dispersión de los errores. Un RMSE de 64.39 sugiere una variabilidad considerable en las predicciones.

6. **Correlación (corr) = 0.26749812**

La correlación mide la relación lineal entre las predicciones y los valores observados. Una correlación de 0.27 es bastante baja, indicando que el modelo no captura bien la relación lineal entre las predicciones y las observaciones reales.

7. **Error MinMax = 0.24342776**

El Error MinMax compara el error de predicción con el rango de los datos observados. Un valor de 0.24 sugiere que el error de predicción es aproximadamente el 24% del rango total de los datos. Esto indica que el error es moderadamente alto en comparación con la variabilidad de los datos.

Comparación entre modelos

- **Precisión y sesgo**: El ajuste exponencial simple muestra un sesgo de subestimación (ME: -4.62) en comparación con el modelo de media móvil ponderada (ME: 2.33) y el modelo de media móvil simple (ME: 4.03).
- **Errores absolutos y porcentuales**: Los valores de MAE (44.31) y MAPE (51.21%) son comparables con los del modelo ponderado, indicando que ambos modelos tienen una precisión moderada. Sin embargo, el ajuste exponencial tiene un MPE más bajo (28.33%) que ambos modelos de media móvil.
- **RMSE**: El RMSE de 64.39 es menor que el del modelo simple (69.17), pero mayor que el del modelo ponderado (62.56), sugiriendo que la variabilidad de los errores en el ajuste exponencial es menor que en el modelo simple pero aún significativa.
- **Correlación**: La correlación de 0.27 es más baja que la del modelo ponderado (0.47) y similar a la del modelo simple, indicando que el ajuste exponencial simple no captura bien la relación entre las observaciones y las predicciones.
- **Error MinMax**: Un Error MinMax de 0.24 es ligeramente mejor que el del modelo simple (0.26) y peor que el del modelo ponderado (0.23), indicando una mejora moderada en el error relativo al rango de los datos.

Dado que la media móvil ponderada es una mejora de la media móvil simple se comparará el ajuste exponencial simple junto a la media móvil simple. En general, el Ajuste Exponencial Simple Modo Manual muestra un mejor desempeño en la mayoría de las métricas de error (mae, mpe, mape, rmse, minmax) en comparación con la Media Móvil Simple. Aunque la correlación es ligeramente menor, las otras métricas indican que el ajuste exponencial simple manual tiene una mayor precisión y un mejor ajuste a los datos observados.

Este análisis sugiere que el Ajuste Exponencial Simple Modo Manual es una mejor opción que la Media Móvil Simple para este conjunto de datos específicos.

Conforme el valor de alfa aumenta y les da más peso a los valores más recientes, el desempeño mejora en este conjunto de datos, por eso se expone en el apartado de a continuación el mejor valor de alfa.

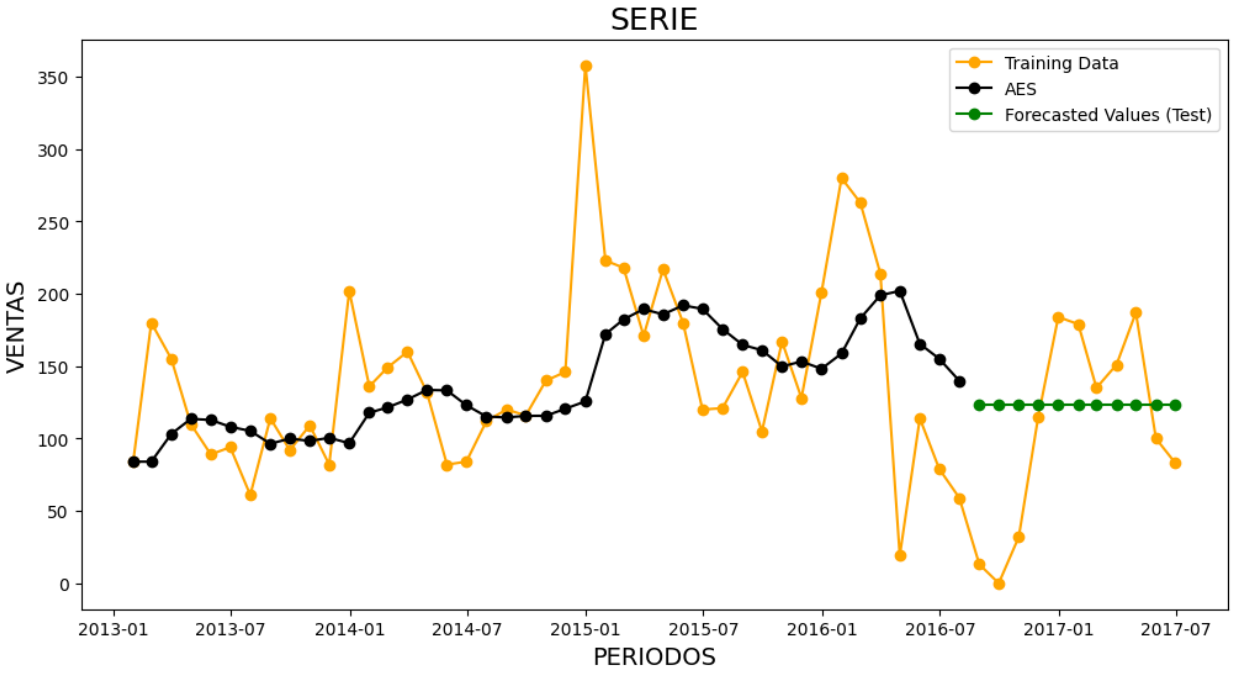

Figura 7-5 Representación del Ajuste Exponencial Simple con alfa = 0.2.

7.1.1.1.2.2 Modo Automático

En la Figura 7-6 se muestra la gráfica con los valores de entrenamiento y predicción del modo automático junto a los valores reales superpuestos, los valores son creados gracias a que la función **SimpleExpSmoothing** toma el mejor valor de alfa.

En este modelo se puede seleccionar con True o False si desea el optimizado automático o no, dado que el objetivo es mostrar el valor más óptimo está marcado True, dando un valor de alfa = 0.6106633.

La Tabla 7-2 es la que muestra en la fila tres los valores de los errores para el Ajuste Exponencial Simple en modo automático. Se puede interpretar lo siguiente de los resultados:

1. **ME (Mean Error) = 0.48280755**

El Error Medio (ME) es la diferencia promedio entre los valores observados y los predichos. En este caso, un ME de 0.48 indica que, en promedio, las predicciones del modelo están ligeramente por encima de las observaciones reales en 0.48 unidades. Este valor cercano a cero sugiere un sesgo mínimo en el modelo.

2. **MAE (Mean Absolute Error) = 41.3810695**

El MAE mide el error promedio absoluto entre las observaciones y las predicciones. Un MAE de 41.38 indica que, en promedio, las predicciones se desvían 41.38 unidades de los valores reales. Este error es comparable al del modelo de media móvil ponderada y sugiere una precisión moderada.

3. **MPE (Mean Percentage Error) = 0.3037248**

El MPE proporciona una medida del error promedio en términos porcentuales. Un MPE de 30.37% sugiere que el modelo, en promedio, sobrestima las predicciones en un 30.37%. Este valor es indicativo de un sesgo hacia la sobreestimación, pero es menor que el MPE del modelo de media móvil ponderada y similar al ajuste exponencial manual.

4. **MAPE (Mean Absolute Percentage Error) = 0.5067843**

El MAPE mide el error absoluto promedio en términos porcentuales. Un MAPE de 50.68% es bastante elevado, indicando que las predicciones del modelo, en promedio, se desvían en un 50.68% de las observaciones reales. Esto muestra que el modelo tiene una precisión moderada.

5. **RMSE (Root Mean Squared Error) = 61.7292612**

El RMSE proporciona una medida de la dispersión de los errores. Un RMSE de 61.73 sugiere una variabilidad considerable en las predicciones. Este valor es menor que los RMSE de los modelos anteriores, lo que indica

una mejora en la precisión y una reducción en la variabilidad de los errores.

6. **Correlación (corr) = 0.44574908**

La correlación mide la relación lineal entre las predicciones y los valores observados. Una correlación de 0.45 indica una relación moderada entre las predicciones y las observaciones reales. Este valor es similar al del modelo de media móvil ponderada y sugiere que el modelo captura bien parte de la estructura de los datos.

7. **Error MinMax = 0.22973757**

El Error MinMax compara el error de predicción con el rango de los datos observados. Un valor de 0.23 sugiere que el error de predicción es aproximadamente el 23% del rango total de los datos. Este valor es igual al del modelo de media móvil ponderada, indicando una mejora significativa en comparación con los otros modelos.

En el código **modelo.summary()** aporta los resultados y ciertos indicadores que muestran lo bien ajustado que está el modelo (AIC y BIC).

```
                       SimpleExpSmoothing Model Results
==============================================================================
Dep. Variable:              unit_sales   No. Observations:                   43
Model:             SimpleExpSmoothing   SSE                        163851.573
Optimized:                       True   AIC                           358.557
Trend:                           None   BIC                           362.080
Seasonal:                        None   AICC                          359.610
Seasonal Periods:                None   Date:                 Sun, 02 Jun 2024
Box-Cox:                        False   Time:                         09:13:50
Box-Cox Coeff.:                  None
==============================================================================
                         coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level            0.6106633                 alpha                 True
initial_level              84.000000                   1.0                False
------------------------------------------------------------------------------
```

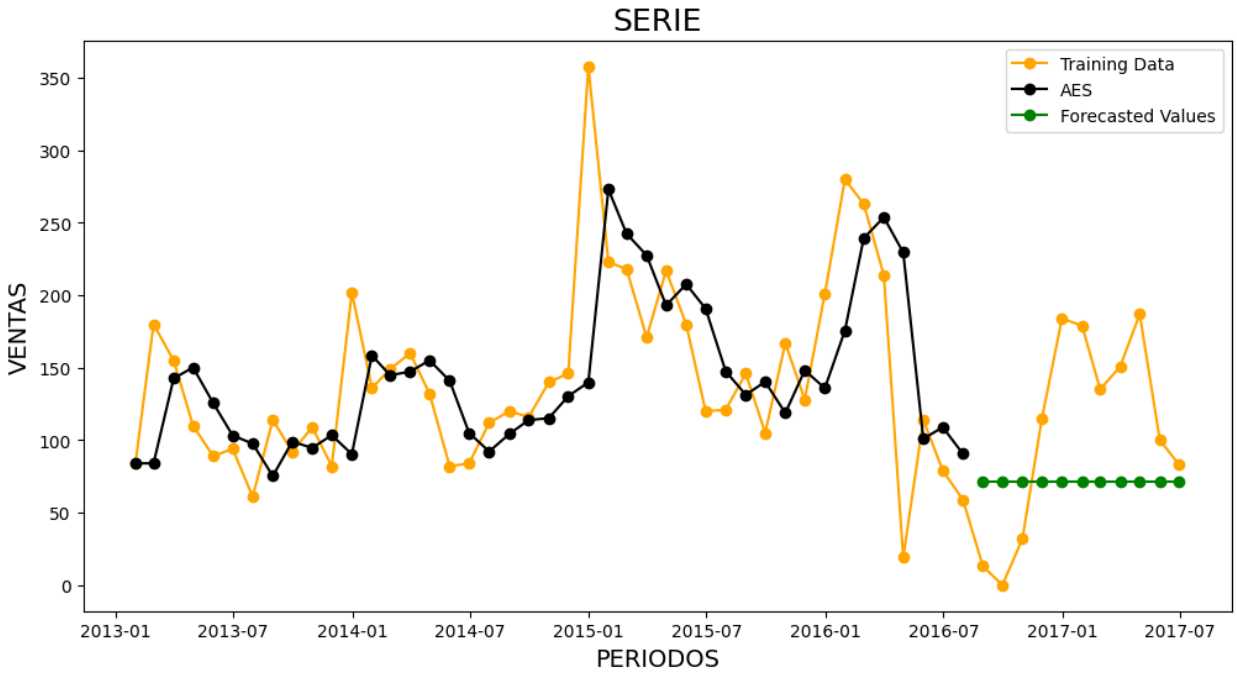

Figura 7-6 Representación del Ajuste Exponencial Simple para alfa óptimo.

En términos generales el Ajuste Exponencial con alfa óptimo muestra mejores resultados en cuanto a errores se refiere si se compara con los modelos expuestos en la demanda nivelada.

7.1.1.2 Métodos de previsión para patrones de demanda con tendencia

Para patrones de demanda con tendencia se usaron los datos extraídos del archivo train.csv y se le aplicó una pendiente con un valor de 10 respecto al índice en el que se encuentra. En la Figura 7-7 están los nuevos valores que se usarán para exponer los modelos de demanda con tendencia.

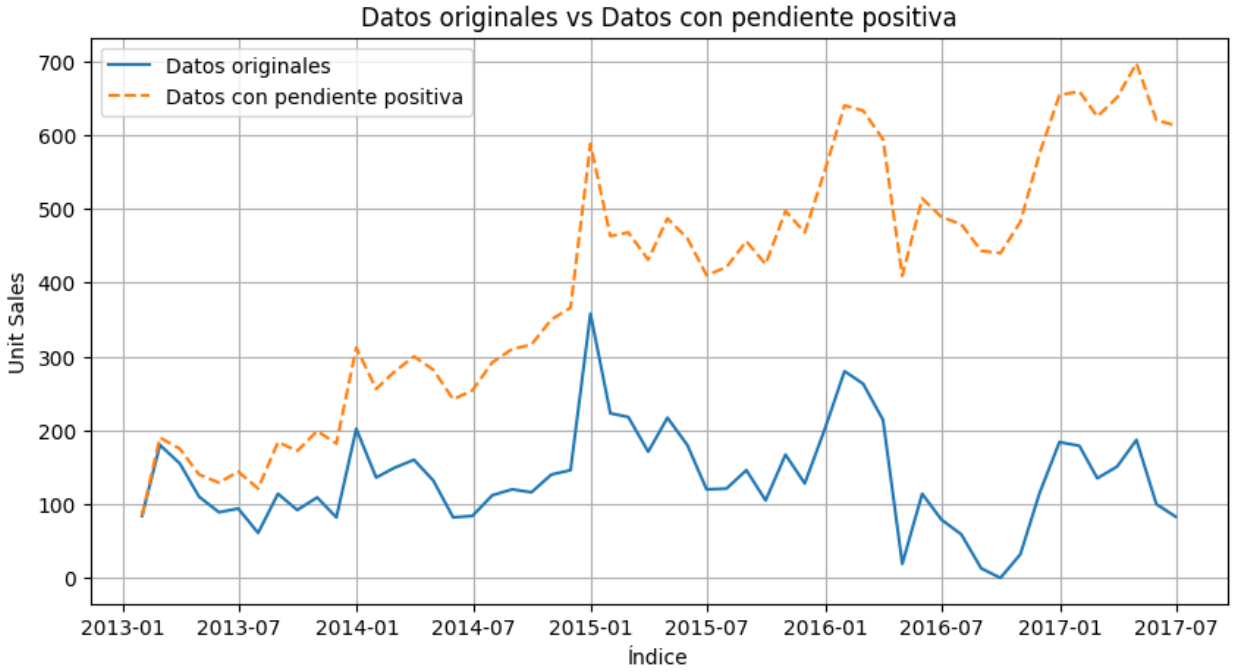

Figura 7-7 Representación de los valores con pendiente.

Para mejorar la facilidad en las explicaciones se crearán dos tablas diferentes una agrupando a los modelos de Medias Dobles, Ajuste Exponencial Doble y Holt y otra diferente para el método Arima.

index	Modelo	me	mae	mpe	mape	rmse	corr	minmax
0	Media Móvil Doble	-12.148	64.278	-0.014	0.144	86.246	0.697	0.132
1	Ajuste Exponencial Doble	-16.088	50.682	-0.029	0.115	72.857	0.788	0.108
2	Holt Manual	16.557	81.845	0.066	0.21	107.603	0.599	0.163
3	Holt Automatico	5.131	50.888	0.023	0.114	73.925	0.788	0.101
4	Arima No-Estacional pdq elegido	-4.786	40.34	-0.01	0.138	59.193	0.924	0.127
5	Arima No-Estacional	-2.753	42.955	-0.011	0.146	62.793	0.918	0.134

Tabla 7-3 Tabla de errores df_errores_DT.

7.1.1.2.1 Media Móvil Doble

En la Figura 7-8 pueden observarse los valores de entrenamiento y predicción para este modelo junto a sus respectivos valores reales.

Se puede el valor de **N** que será el tamaño de la ventana de valores. Para este caso se ha tomado un valor de **N=7**, de esta forma se puede observar como capta la tendencia de una manera más conservadora y antes bajadas o subidas repentinas no cambia la tendencia, al estar estudiando un patrón con tendencia como puede ser el de un producto cada vez más vendido y aceptado, interesa que capte la tendencia que lleve a medio plazo, de esta forma ante circunstancias excepcionales no decaerá o se incrementará rápidamente.

La Tabla 7-3 muestra en la fila cero los valores de los errores para la media móvil doble. Interpretaciones:

1. ME (Mean Error) = -12.1482993

El ME mide el error promedio entre las predicciones y los valores reales. Un valor de -12.1483 indica que, en promedio, el modelo está subestimando la demanda por esta cantidad. Este sesgo negativo sugiere una tendencia del modelo a predecir valores más bajos de los reales.

2. MAE (Mean Absolute Error) = 64.2780045

El MAE mide el promedio de los errores absolutos. Un MAE de 64.278 indica que, en promedio, las predicciones del modelo se desvían de los valores reales en aproximadamente 64.278 unidades. Este valor sugiere un error considerable en las predicciones.

3. MPE (Mean Percentage Error) = - 0.0137222

El MPE mide el error promedio en términos porcentuales. Un MPE de -0.0137% es muy cercano a cero, lo que indica que, en promedio, el modelo no tiene un sesgo porcentual significativo. Sin embargo, el signo negativo sugiere una leve tendencia a la subestimación.

4. MAPE (Mean Absolute Percentage Error) = 0.1438945

El MAPE mide el error porcentual promedio en términos absolutos. Un MAPE de 0.1439 (14.39%) sugiere que, en promedio, las predicciones del modelo se desvían de los valores reales en un 14.39%.

5. RMSE (Root Mean Squared Error) = 86.2457849

El RMSE es una medida que pondera más fuertemente los errores grandes. Un RMSE de 86.2458 indica una magnitud considerable del error en las predicciones. Este valor es relativamente alto, lo que sugiere una variabilidad significativa entre las predicciones y los valores reales.

6. Correlación (corr) = 0.6972509

El coeficiente de correlación mide la relación lineal entre las predicciones y los valores reales. Un valor de 0.6973 indica una correlación moderada-alta positiva, lo que significa que las predicciones siguen la tendencia general de los datos reales.

7. MinMax = 0.1322438

El error Min-Max mide la diferencia relativa entre los valores predichos y los reales. Un valor de 0.1322 (13.22%) sugiere que la diferencia máxima entre los valores reales y predichos es del 13.22%.

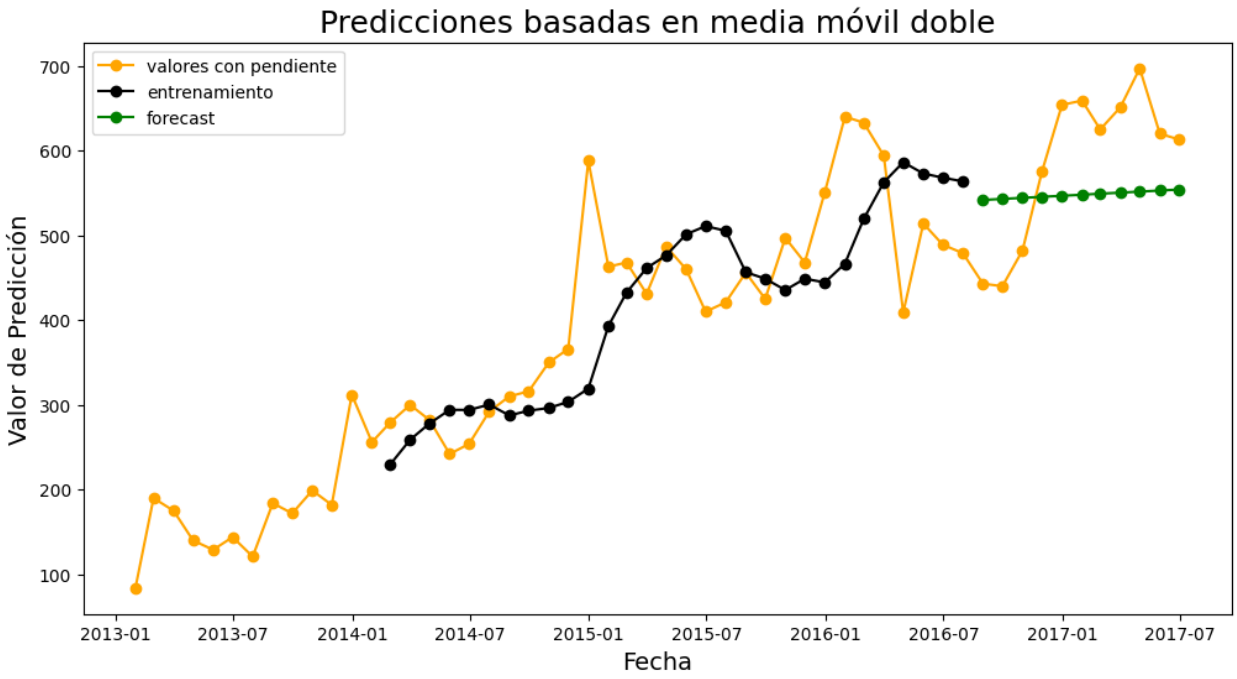

Figura 7-8 Representación Media Móvil Doble.

7.1.1.2.2 Ajuste Exponencial Doble

En la Figura 7-9 puede observarse los valores de entrenamiento y predicción del Ajuste Exponencial Doble junto a los valores del producto si tuviese pendiente.

Para este caso se ha tomado un valor de **t = 13** para poder comparar resultados con la Media Móvil Doble de la manera más correcta posible y **alfa = 0.2** para dar más peso a valores más lejanos y que pueda seguirse la pendiente de la manera más conservadora posible. La Tabla 7-3 muestra en la primera fila los valores de los errores para la media móvil doble. Interpretaciones:

1. **ME (Mean Error) = - 16.0876247**

El ME mide el error promedio entre las predicciones y los valores reales. Un valor de -16.0876 indica que, en promedio, el modelo está subestimando la demanda por esta cantidad. Este sesgo negativo sugiere una tendencia del modelo a predecir valores más bajos de los reales.

2. **MAE (Mean Absolute Error) = 50.6819739**

El MAE mide el promedio de los errores absolutos. Un MAE de 50.6819 indica que, en promedio, las predicciones del modelo se desvían de los valores reales en aproximadamente 50.6819 unidades. Este valor sugiere un error moderado en las predicciones.

3. **MPE (Mean Percentage Error) = - 0.0286915**

El MPE mide el error promedio en términos porcentuales. Un MPE de -0.0287% es muy cercano a cero, lo que indica que, en promedio, el modelo no tiene un sesgo porcentual significativo. Sin embargo, el signo negativo sugiere una leve tendencia a la subestimación.

4. **MAPE (Mean Absolute Percentage Error) = 0.1151804**

El MAPE mide el error porcentual promedio en términos absolutos. Un MAPE de 0.1152 (11.52%) sugiere que, en promedio, las predicciones del modelo se desvían de los valores reales en un 11.52%.

5. **RMSE (Root Mean Squared Error) = 72.8571900**

El RMSE es una medida que pondera más fuertemente los errores grandes. Un RMSE de 72.8572 indica una magnitud considerable del error en las predicciones. Este valor es relativamente alto, lo que sugiere una variabilidad significativa entre las predicciones y los valores reales.

6. **Correlación (corr) = 0.7878621**

El coeficiente de correlación mide la relación lineal entre las predicciones y los valores reales. Un valor de 0.7879 indica una correlación moderada-alta positiva, lo que significa que las predicciones siguen la tendencia general de los datos reales.

7. **MinMax = 0.1080027**

El error Min-Max mide la diferencia relativa entre los valores predichos y los reales. Un valor de 0.1080 (10.80%) sugiere que la diferencia máxima entre los valores reales y predichos es del 10.80%.

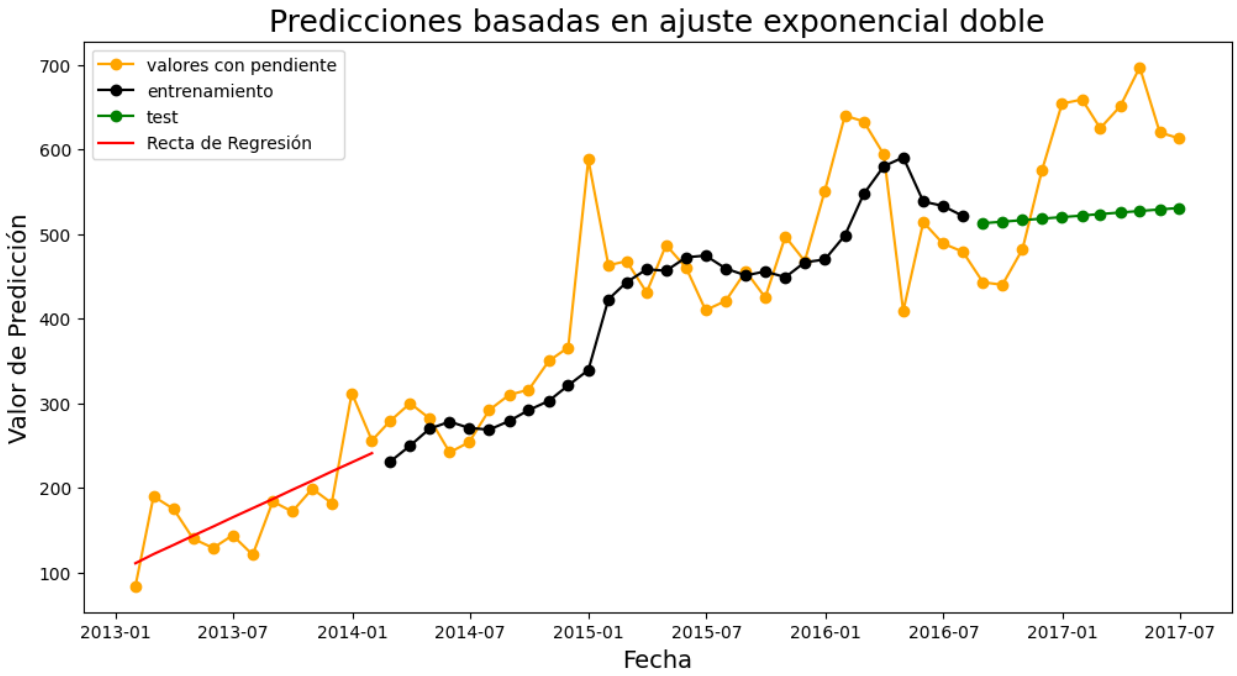

Figura 7-9 Representación Ajuste Exponencial Doble.

El modelo de Ajuste Exponencial Doble demuestra un mejor rendimiento en casi todas las métricas de error en comparación con el modelo de Media Móvil Doble. Tiene menores valores de MAE, MAPE, y RMSE, lo que indica una mayor precisión y menor dispersión en los errores de predicción. Además, una mayor correlación sugiere que este modelo captura mejor las tendencias de los datos. Por lo tanto, el Ajuste Exponencial Doble es el modelo preferido entre los dos para esta serie temporal.

7.1.1.2.3 Modelo de Holt

Para este modelo se han hecho dos ejecuciones de código en Python distintas.

7.1.1.2.3.1 Modo Manual

En la Figura 7-10 se puede observar los valores de entrenamiento y predichos, así como la recta de regresión para obtener los valores iniciales del modelo.

Se podrá escoger el valor **t** para el número de valores de la recta regresión, así como los valores de alfa y beta del modelo. Para este caso se ha escogido **t = 13** para seguir comparando con los modelos anteriores, **alfa = 0.6** y **beta = 0.8**. Para este caso dado el mejor comportamiento del método se ha decidido tomar un alfa mayor para dar más peso a observaciones recientes y un valor de beta relativamente alto para que se le de más importancia a los cambios de pendientes recientes y poder seguir así mejor las tendencias.

La Tabla 7-3 muestra en la fila dos los valores de los errores para modo Holt manual. En comparación con Media Móvil Doble y Ajuste Exponencial Doble, el método de Holt Manual muestra un rendimiento inferior en todas las métricas de error. Específicamente, Holt Manual tiene:

- El ME positivo más alto, lo que sugiere sobreestimación.
- El MAE y MAPE más altos, indicando una menor precisión.
- El RMSE más alto, sugiriendo mayores errores grandes.
- La correlación más baja, indicando una menor relación con los valores reales.
- El valor Minmax más alto, indicando mayor variabilidad en los errores.

En resumen, el método de Holt Manual, según las métricas proporcionadas, tiene un rendimiento inferior en comparación con Media Móvil Doble y Ajuste Exponencial Doble para esta serie temporal específica. Esto podría deberse a una mala configuración de los parámetros (alfa y beta) o a que el modelo de Holt no es adecuado para los datos específicos analizados.

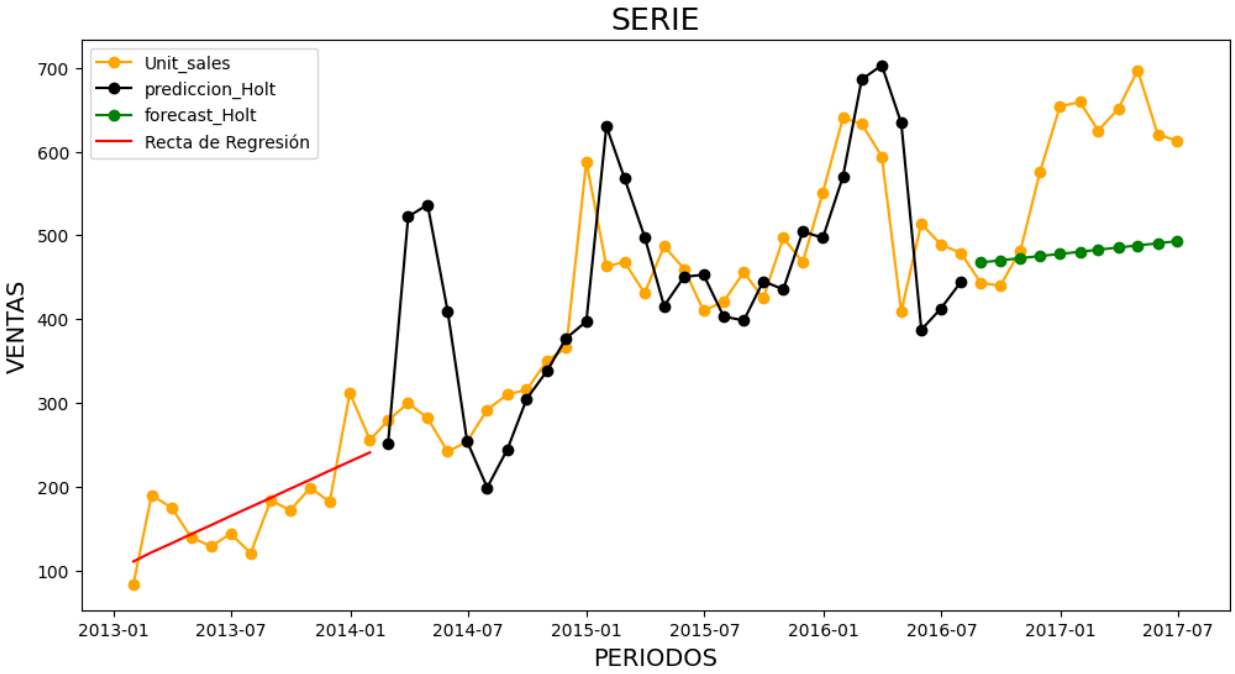

Figura 7-10 Representación del modelo donde se escogen los parámetros.

7.1.1.2.3.2 Modo Automático

En la Figura 7-11 se puede observar los valores de entrenamiento y predichos, así como la recta de regresión para obtener los valores iniciales del modelo.

Se podrá volver a escoger el tamaño de la recta de regresión ya que las funciones definidas escogerán los mejores valores de alfa y beta. Para este caso los mejores valores de alfa y beta son 0.788 y 0.247 respectivamente. A continuación, se pueden observar todas las variables:

```
                              Holt Model Results
================================================================================
Dep. Variable:     unit_sales_with_slope   No. Observations:                   30
Model:                              Holt   SSE                         163945.772
Optimized:                         False   AIC                            266.183
Trend:                          Additive   BIC                            271.788
Seasonal:                           None   AICC                           269.835
Seasonal Periods:                   None   Date:                Mon, 10 Jun 2024
Box-Cox:                           False   Time:                        17:01:45
Box-Cox Coeff.:                     None
================================================================================
                          coeff                 code              optimized
--------------------------------------------------------------------------------
smoothing_level            0.7876113                alpha                 False
smoothing_trend            0.2471433                 beta                 False
initial_level              279.00000                  l.0                 False
initial_trend              21.000000                  b.0                 False
--------------------------------------------------------------------------------
```

En cuanto a la interpretación de errores se puede comentar que, en comparación con Media Móvil Doble, Ajuste Exponencial Doble y Holt Manual, el método de Holt Automático muestra un rendimiento superior en la mayoría de las métricas de error. Específicamente, Holt Automático tiene:

- Un ME más cercano a cero, indicando menor sesgo.
- Un MAE similar al Ajuste Exponencial Doble y mucho menor que Holt Manual, indicando mayor precisión.
- Un MPE más cercano a cero, sugiriendo menor sesgo en términos porcentuales.
- El MAPE más bajo, indicando la mejor precisión porcentual.
- Un RMSE similar al Ajuste Exponencial Doble y mucho menor que Holt Manual, indicando menos

errores grandes.

- Una correlación igual a la de Ajuste Exponencial Doble, la más alta entre todos los modelos.
- El valor Minmax más bajo, indicando la menor variabilidad en los errores de pronóstico.

En resumen, el método de Holt Automático parece ajustar mejor los datos y proporcionar pronósticos más precisos y consistentes en comparación con los otros métodos, particularmente el Holt Manual, Media Móvil Doble y es comparable al Ajuste Exponencial Doble. Esto sugiere que el proceso automático para seleccionar los parámetros del método de Holt proporciona una ventaja significativa en términos de rendimiento.

En comparación con Holt Manual el mejor rendimiento del método de Holt Automático se debe a la optimización de los parámetros de suavizado α y β. El Holt Automático, con α = 0.787 y β = 0.2471, ajusta mejor a los datos históricos, respondiendo más rápidamente a los cambios recientes en el nivel de la serie temporal y suavizando adecuadamente la tendencia. Esto resulta en un modelo que captura de manera más precisa tanto el nivel actual como la tendencia subyacente, reduciendo la variabilidad en los errores y mejorando la precisión general del pronóstico en comparación con los parámetros manuales (α = 0.6 y β = 0.8).

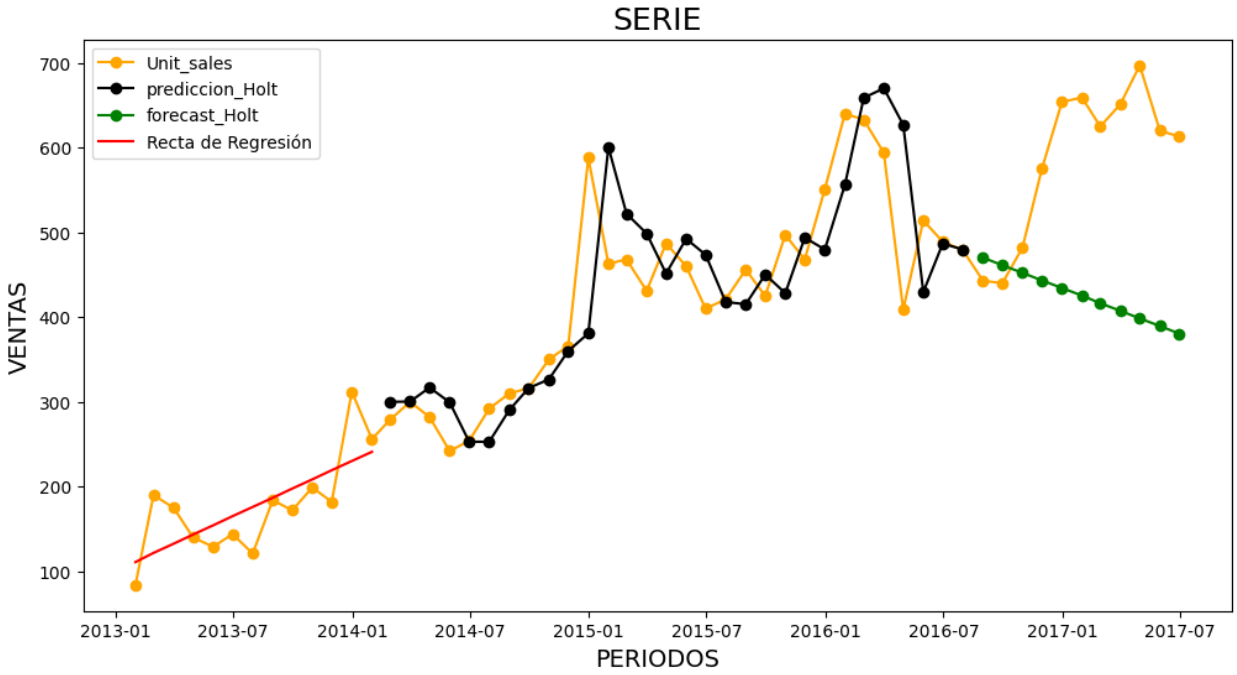

Figura 7-11 Representación modelo de Holt Óptimo.

7.1.1.2.4 Modelo Arima No-Estacional

En el modelo Arima se ha hecho una distinción parecida a anteriores modelos, en una primera parte se expone el proceso de selección de variables y luego se usa la función **auto_Arima** que permite el cálculo de los mejores valores **p**, **d** y **q**.

Como se expuso en Modelos Arima no-estacionales Arima(p,d,q), una serie con tendencia o estacionalidad no puede ser diferenciada. Se expone el resultado obtenido tras ejecutar la prueba de estacionariedad:

```
Test estacionariedad serie original
```

```
-------------------------------------
ADF Statistic: -1.519691547301593, p-value: 0.5237486996957055

Test estacionariedad para serie diferenciada (order=1)
-------------------------------------------------
ADF Statistic: -8.913239712142355, p-value: 1.089926975028665e-14

Test estacionariedad para serie diferenciada (order=2)
-------------------------------------------------

ADF Statistic: -5.931721299161503, p-value: 2.3719698564110958e-07
```

Se muestra el valor **ADF** y el **p-value** de cada serie, la serie original queda descartada según (Burcu Özcan & Ilhan Öztürk, 2019) dado que **p-value** toma un valor superior a 0.05, la primera diferenciación es la que da un valor **ADF** más negativo y un **p-value** más bajo, siendo **d = 1** según (Hyndman, 2021) por tanto el valor más acertado para **d**.

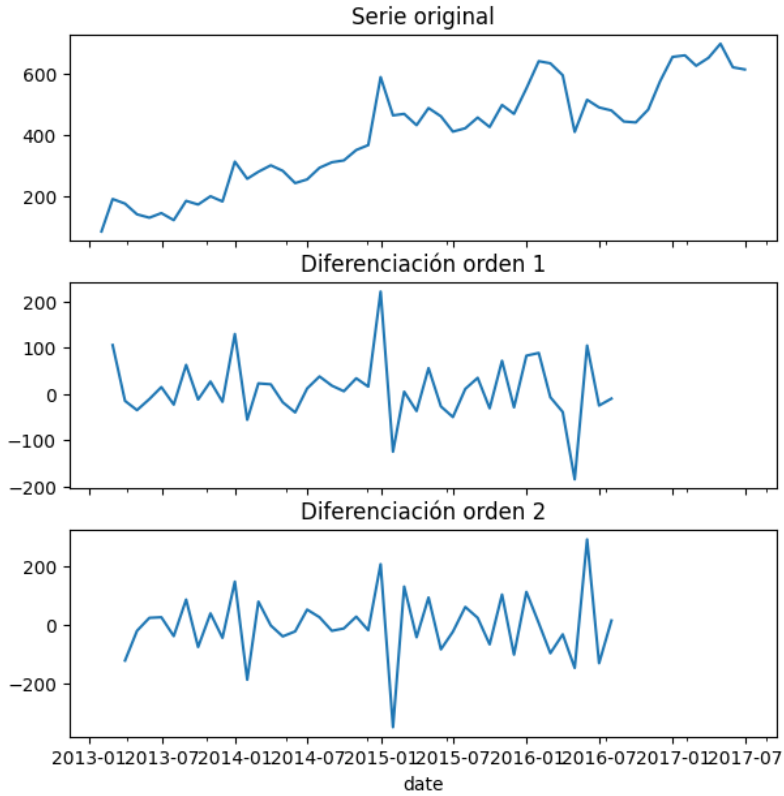

Figura 7-12 Diferenciación de la serie.

Tras esto es necesario analizar los gráficos **ACF** y **PACF.** Como se expone en apartado 9.1 de (Hyndman, 2021), si la gráfica **ACF** tiende a cero rápidamente es una serie estacionaria, como se puede ver en la Figura 7-13, la autocorrelación de la serie original tiende lentamente a cero lo que indica un componente de tendencia, la serie diferenciada tiende a cero mucho más rápidamente que la serie original teniendo por tanto más características de estacionariedad. Tras rearfimar la estacionariedad es necesario visualizar la serie diferenciada de orden 1 para determinar el valor de **q** y **p,** según el apartado 9.5 de (Hyndman, 2021) el valor correcto para el valor **q** es 1 dado que cae en el valor 1 en la Figura 7-13, al igual que con el valor **p** en la Figura 7-14.

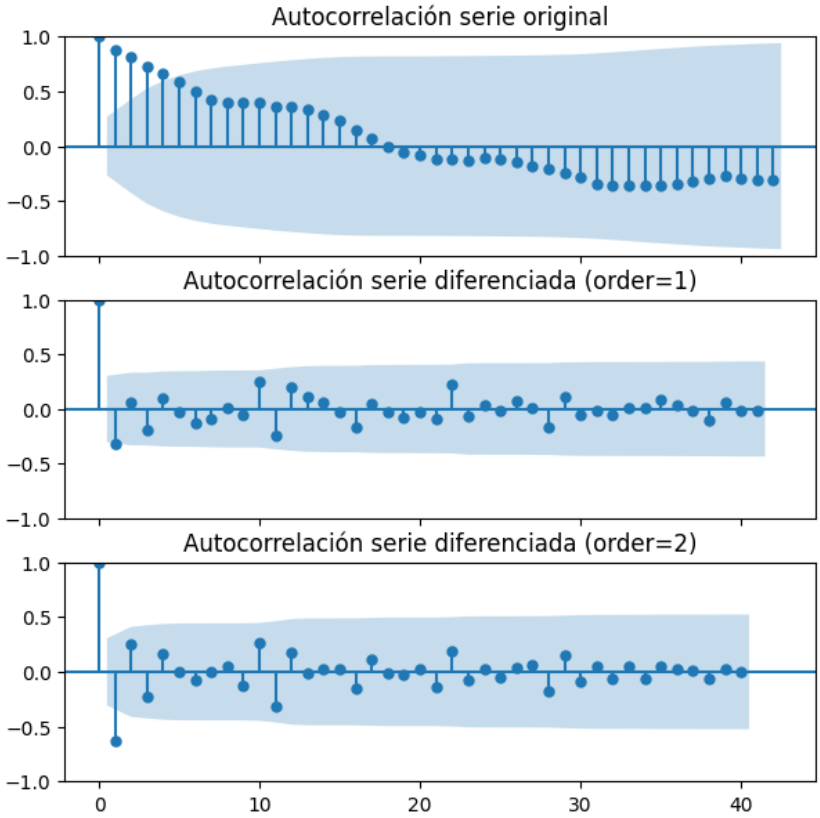

Figura 7-13 Gráficos ACF de serie original y serie diferenciada.

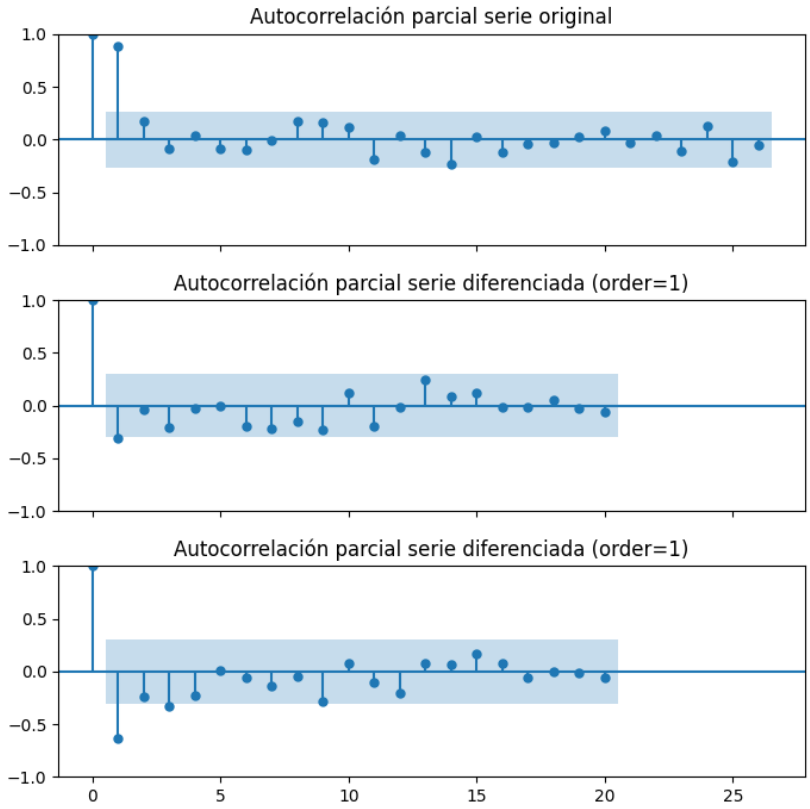

Figura 7-14 Gráficos PACF de serie original y diferenciada.

Tras ello y definir como valores adecuados p, d, q (1, 1, 1) se obtienen los valores de entrenamiento y predicción.

Respecto a la función summary() se obtiene el siguiente resultado:

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                       43
Model:                 SARIMAX(1, 1, 1)   Log Likelihood                  -230.954
Date:                  Mon, 10 Jun 2024   AIC                             469.908
Time:                          15:57:51   BIC                             476.859
Sample:                      01-31-2013   HQIC                            472.456
                           - 07-31-2016
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      5.6717      3.483      1.628      0.103      -1.155      12.498
ar.L1          0.4801      0.433      1.108      0.268      -0.369       1.330
ma.L1         -0.9997     51.726     -0.019      0.985    -102.381     100.381
sigma2      3275.7566    1.7e+05      0.019      0.985   -3.29e+05    3.36e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):                35.40
Prob(Q):                              0.78   Prob(JB):                         0.00
Heteroskedasticity (H):               2.44   Skew:                             0.78
Prob(H) (two-sided):                  0.11   Kurtosis:                         7.22

===================================================================================
```

Respecto a los errores que se pueden observar en la Tabla 7-3 se incluyen varios comentarios:

1. ME (Mean Error) = - 4.7864262

El ME mide el error promedio entre las predicciones y los valores reales. Un valor de -4.7864 indica que, en promedio, el modelo está subestimando la demanda por esta cantidad. Este sesgo negativo sugiere una tendencia del modelo a predecir valores más bajos de los reales.

2. MAE (Mean Absolute Error) = 40.3398362

El MAE mide el promedio de los errores absolutos. Un MAE de 40.3398 indica que, en promedio, las predicciones del modelo se desvían de los valores reales en aproximadamente 40.3398 unidades. Este valor sugiere un error moderado en las predicciones, que es menor comparado con otros modelos analizados anteriormente.

3. MPE (Mean Percentage Error) = - 0.0104999

El MPE mide el error promedio en términos porcentuales. Un MPE de -0.0105% es muy cercano a cero, lo que indica que, en promedio, el modelo no tiene un sesgo porcentual significativo. El signo negativo sugiere una leve tendencia a la subestimación.

4. MAPE (Mean Absolute Percentage Error) = 0.1380195

El MAPE mide el error porcentual promedio en términos absolutos. Un MAPE de 0.1380 (13.80%) sugiere que, en promedio, las predicciones del modelo se desvían de los valores reales en un 13.80%. Este nivel de error es aceptable y es más bajo que el de algunos otros modelos, lo cual indica una mejor precisión relativa.

5. RMSE (Root Mean Squared Error) = 59.1931618

El RMSE es una medida que pondera más fuertemente los errores grandes. Un RMSE de 59.1932 indica una magnitud moderada del error en las predicciones. Este valor es más bajo comparado con otros modelos analizados, lo cual sugiere menor variabilidad entre las predicciones y los valores reales.

6. Correlación (corr) = 0.9236705

El coeficiente de correlación mide la relación lineal entre las predicciones y los valores reales. Un valor de 0.9237 indica una correlación alta positiva, lo que significa que las predicciones siguen muy bien la tendencia general de los datos reales.

7. MinMax = 0.1272550

El error Min-Max mide la diferencia relativa entre los valores predichos y los reales. Un valor de 0.1273 (12.73%) sugiere que la diferencia máxima entre los valores reales y predichos es del 12.73%. Este valor es

razonable y muestra una menor variabilidad entre las predicciones y los valores reales.

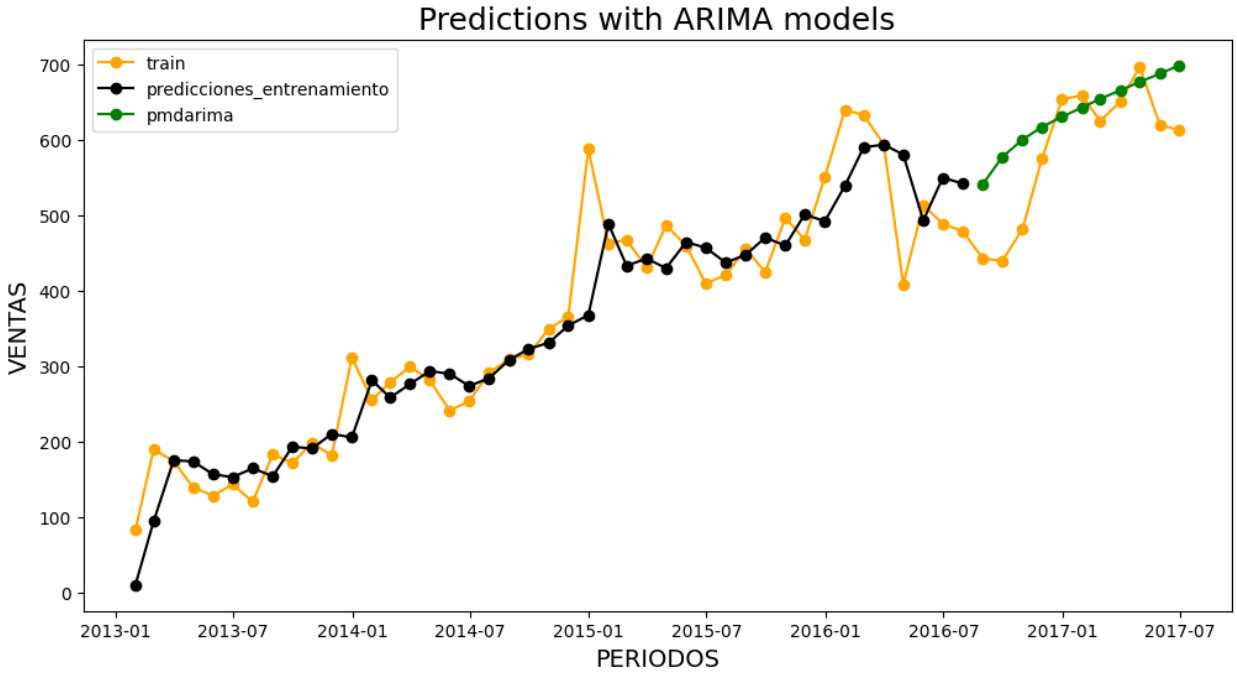

Figura 7-15 Representación Arima No-Estacional.

El modelo Arima No-Estacional con p, d, q elegido es el mejor entre todos los modelos comparados. Presenta el ME, MAE, MPE, y RMSE más bajos, además de la correlación más alta, lo que indica una mejor precisión y una relación más fuerte entre los pronósticos y los valores reales. Aunque el Holt Automático tiene un Minmax ligeramente mejor, las demás métricas clave favorecen claramente a Arima.

Tras esto se puede ver como la función **auto_Arima()** calcula los valores óptimos para los parámetros **p**, **q** y **q** siendo para este caso de (0,1,1) teniendo una leve diferencia respecto a lo expuesto anteriormente según la interpretación de los errores.

7.1.1.3 Métodos de previsión para patrones de demanda con estacionalidad

Para patrones de demanda con estacionalidad se usa los datos del archivo **train.csv**, se les aplica una pendiente junto a la estacionalidad de los primeros 12 meses y un incremento de la amplitud para que aumente dicha amplitud de la estacionalidad y sea cada vez mayor.

En la Figura 7-16 Valores de demanda con estacionalidad puede observarse los valores y la representación de los valores aplicada la estacionalidad de los primeros 12 meses.

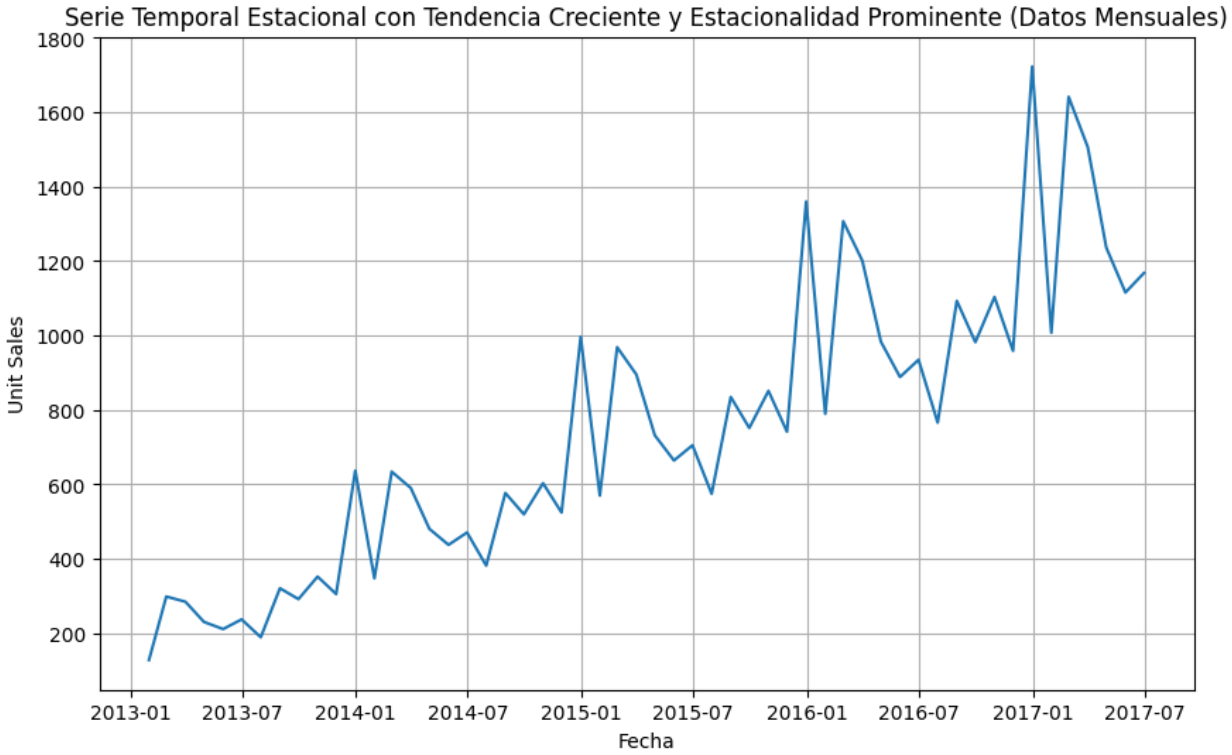

Figura 7-16 Valores de demanda con estacionalidad.

index	Modelo	me	mae	mpe	mape	rmse	corr	minmax
0	Holt-Winters Manual	0.498	38.109	-0.016	0.088	52.630	0.985	0.082
1	Holt-Winters aditivo	-0.008	38.169	-0.018	0.089	52.549	0.985	0.083
2	Holt-Winters multiplicativo	3.009	10.651	0.014	0.030	14.103	0.999	0.028
3	Arima	-3.371	39.775	-0.013	0.122	92.016	0.956	0.104

Tabla 7-4 Dataframe de errores (df_errores).

7.1.1.3.1 Holt-Winters

Para este modelo se ha creado en Python tres ejecuciones de código diferentes. Atendiendo a una forma completamente manual y las siguientes automáticas según el modelo que se trate. Se exponen los resultados en los siguientes apartados.

7.1.1.3.1.1 Holt-Winters Manual

En esta ejecución se puede escoger los diferentes valores de alfa, beta y gamma, así como el tipo de modelo, pudiendo experimentar modificando la tendencia o estacionalidad asignándole aditividad o multiplicidad dado que son entradas de la función **ExponentialSmoothing**.

El resultado expuesto a continuación toma los siguientes valores **alpha = 0.13**, **beta = 0.01**, **gamma = 0.02**, **estacionalidad = aditiva** y **tendencia = aditiva**. En la Figura 7-17 se muestran los resultados obtenidos para este método en **HW_MANUAL**

La Tabla 7-4 muestra en la fila cero los valores de los errores para este método. Se comentará al final junto a los demás modelos el modelo qué mejor ajuste.

Datos del entrenamiento ExponentialSmoothing Model Results

Dep. Variable:	endog	No. Observations:	43
Model:	ExponentialSmoothing	SSE	119104.982
Optimized:	True	AIC	372.842
Trend:	Additive	BIC	401.021
Seasonal:	Additive	AICC	401.342
Seasonal Periods:	12	Date:	Tue, 11 Jun 2024
Box-Cox:	False	Time:	14:53:49
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.1300000	alpha	False
smoothing_trend	0.0100000	beta	False
smoothing_seasonal	0.0200000	gamma	False
initial_level	160.44459	1.0	True
initial_trend	21.129701	b.0	True
initial_seasons.0	-107.91268	s.0	True
initial_seasons.1	214.03202	s.1	True
initial_seasons.2	133.16327	s.2	True
initial_seasons.3	-24.314387	s.3	True
initial_seasons.4	-101.54614	s.4	True
initial_seasons.5	-85.839834	s.5	True
initial_seasons.6	-215.69923	s.6	True
initial_seasons.7	-12.065643	s.7	True
initial_seasons.8	-89.703187	s.8	True
initial_seasons.9	-29.495456	s.9	True
initial_seasons.10	-128.98572	s.10	True
initial_seasons.11	323.96369	s.11	True

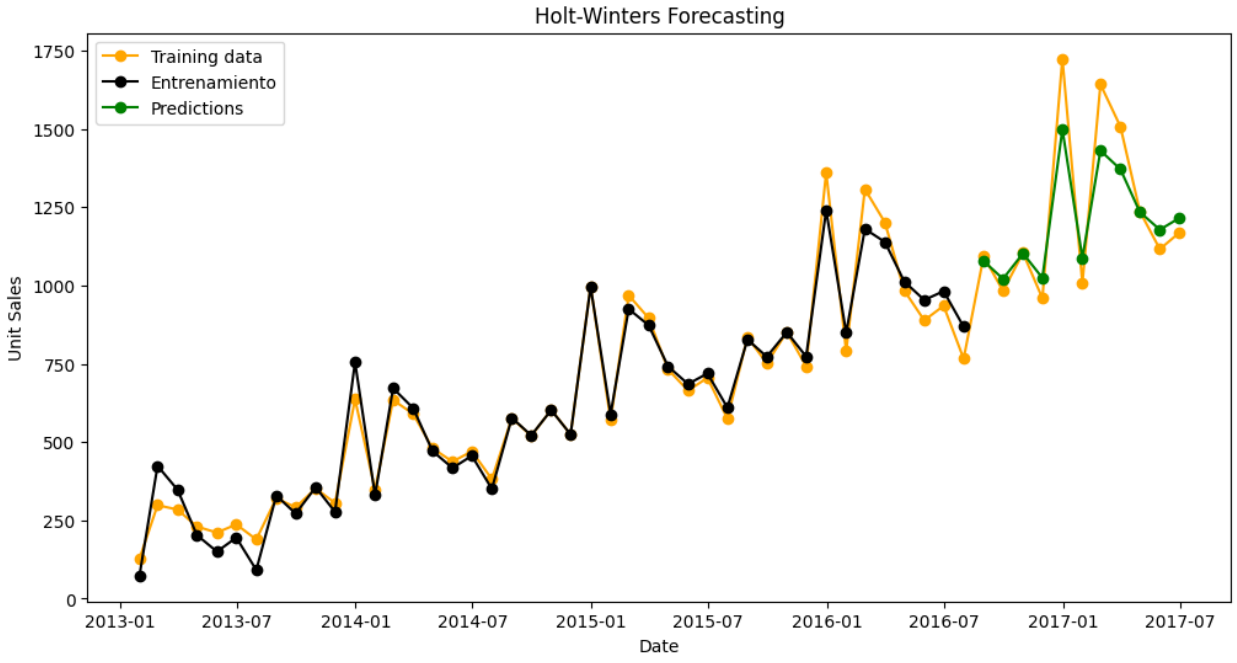

Figura 7-17 Representación de Holt-Winters con valores elegidos.

7.1.1.3.1.2 Holt-Winters Aditivo

En este método se puede ejecutar el código para observar cuáles son los mejores resultados que calcula la función **ExponentialSmoothing** para los parámetros alfa, beta y gamma cuando la tendencia y estacionalidad son aditivos. La Figura 7-18 muestra los resultados obtenidos frente a los datos estudiados.

Los resultados de las variables del modelo son las siguientes:

```
Datos del entrenamiento                       ExponentialSmoothing Model Results
==============================================================================
Dep. Variable:                      endog   No. Observations:                   43
Model:             ExponentialSmoothing   SSE                         118737.850
Optimized:                         True   AIC                            372.709
Trend:                         Additive   BIC                            400.889
Seasonal:                      Additive   AICC                           401.209
Seasonal Periods:                    12   Date:                 Thu, 23 May 2024
Box-Cox:                          False   Time:                         15:58:29
Box-Cox Coeff.:                    None
==============================================================================
                              coeff                 code              optimized
------------------------------------------------------------------------------
smoothing_level            0.1325779                alpha                 True
smoothing_trend            0.0025904                 beta                 True
smoothing_seasonal         0.0184876                gamma                 True
initial_level              157.86528                  l.0                 True
initial_trend              21.139892                  b.0                 True
initial_seasons.0         -107.06006                  s.0                 True
initial_seasons.1          214.85207                  s.1                 True
initial_seasons.2          134.02595                  s.2                 True
initial_seasons.3         -24.980709                  s.3                 True
initial_seasons.4         -100.57100                  s.4                 True
initial_seasons.5         -85.937214                  s.5                 True
initial_seasons.6         -215.74591                  s.6                 True
initial_seasons.7         -13.415360                  s.7                 True
initial_seasons.8         -90.998736                  s.8                 True
initial_seasons.9         -30.607239                  s.9                 True
initial_seasons.10        -130.12897                 s.10                 True
initial_seasons.11         323.55247                 s.11                 True
------------------------------------------------------------------------------
```

Donde se puede observar el valor de **alfa** como **smoothing_level**, el valor de **beta** como **smoothing_trend** y el valor de **gamma** como **smoothing_seasonal**. También se pueden observar valores como **AIC**, **BIC** y **AICC** que indican la calidad del modelo, cuanto más bajos sean los valores mejor calidad tendrá.

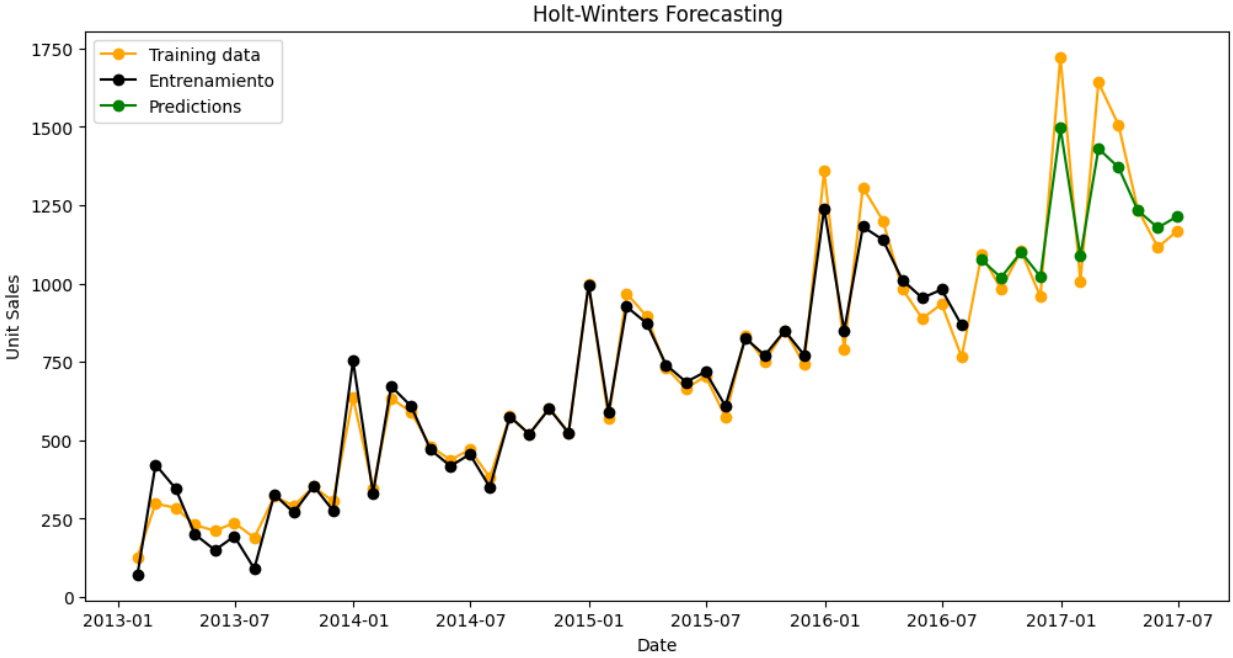

Figura 7-18 Representación del método de Holt-Winters aditivo.

7.1.1.3.1.3 Holt-Winters Multiplicativo

En este método se puede ejecutar el código para observar cuáles son los mejores resultados que calcula la función **ExponentialSmoothing** para los parámetros alfa, beta y gamma cuando la tendencia y estacionalidad son multiplicativos. La Figura 7-19 muestra los resultados obtenidos tanto de entrenamiento como de

previsiones.

Los resultados de las variables del modelo son las siguientes:

Datos del entrenamiento — ExponentialSmoothing Model Results

Dep. Variable:	endog	No. Observations:	43
Model:	ExponentialSmoothing	SSE	8552.327
Optimized:	True	AIC	259.589
Trend:	Multiplicative	BIC	287.768
Seasonal:	Multiplicative	AICC	288.089
Seasonal Periods:	12	Date:	Thu, 23 May 2024
Box-Cox:	False	Time:	15:59:01
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	0.4425327	alpha	True
smoothing_trend	0.4425213	beta	True
smoothing_seasonal	6.223e-11	gamma	True
initial_level	200.67443	1.0	True
initial_trend	1.1263499	b.0	True
initial_seasons.0	0.7428074	s.0	True
initial_seasons.1	1.2225711	s.1	True
initial_seasons.2	1.0937181	s.2	True
initial_seasons.3	0.8667356	s.3	True
initial_seasons.4	0.7616896	s.4	True
initial_seasons.5	0.7836519	s.5	True
initial_seasons.6	0.6219549	s.6	True
initial_seasons.7	0.8899488	s.7	True
initial_seasons.8	0.7791682	s.8	True
initial_seasons.9	0.8676647	s.9	True
initial_seasons.10	0.7344029	s.10	True
initial_seasons.11	1.3381175	s.11	True

A continuación, se expone la interpretación de errores a partir de la Tabla 7-4 de los tres resultados de Holt-Winters anteriormente expuestos, de los tres modelos Holt-Winters evaluados, **el modelo multiplicativo** es el que presenta las mejores métricas de error:

- **Menores valores de MAE, MAPE y RMSE**, lo que indica una mayor precisión en los pronósticos.
- **Correlación más alta (0.999)**, indicando una excelente relación entre los valores pronosticados y los valores reales.
- **Minmax más bajo (0.028)**, sugiriendo una menor variabilidad en los errores de pronóstico.

Aunque el modelo multiplicativo tiene un sesgo positivo pequeño (ME=3.009), sus otras métricas de error significativamente mejores lo hacen el mejor modelo entre los tres presentados.

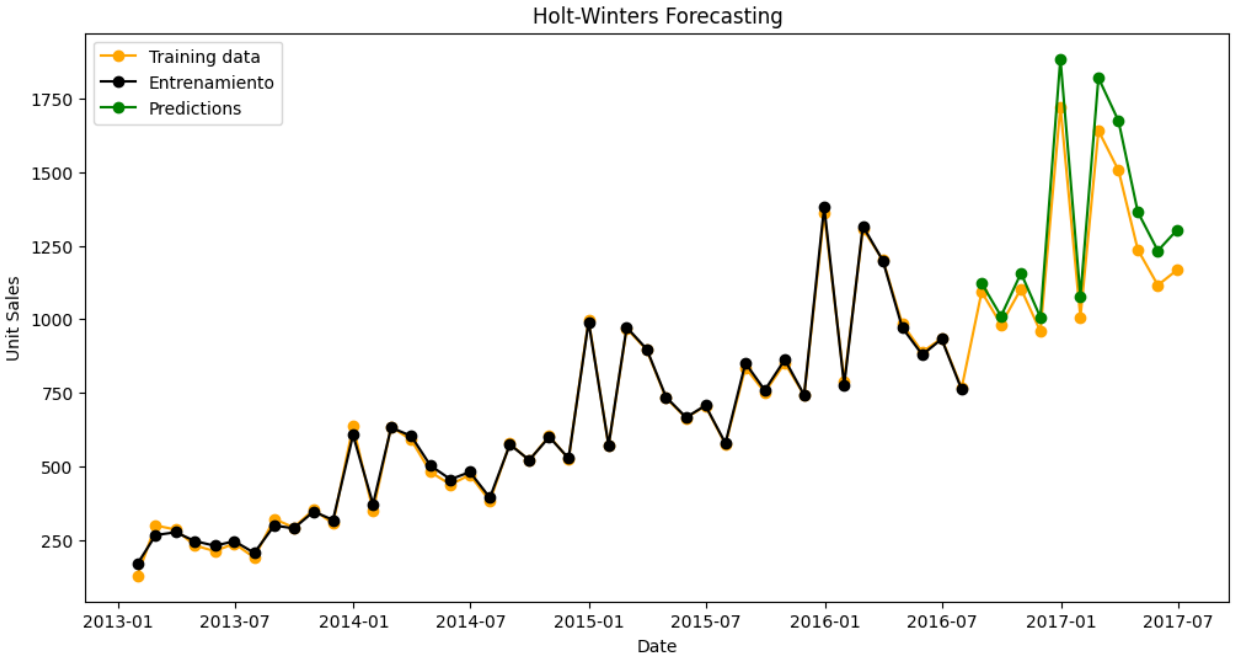

Figura 7-19 Representación método Holt-Winters Multiplicativo.

7.1.1.3.2 Modelos Arima estacionales

El modelo Arima estacional es similar al no-estacional salvo que es necesario elegir los parámetros adicionales de la parte estacional, es decir, P, D, Q, m.

Como se ha comentado en anteriores apartados una serie con tendencia o estacionalidad debe ser diferenciada, a continuación, se expone los resultados de la prueba de estacionariedad para el modelo Arima estacional.

```
Test estacionariedad serie original
------------------------------------
ADF Statistic: -0.31675370143623566, p-value: 0.9230977619189328

Test estacionariedad para serie diferenciada (order=1)
------------------------------------------------------
ADF Statistic: -45.19656826658537, p-value: 0.0

Test estacionariedad para serie diferenciada (order=2)
------------------------------------------------------
ADF Statistic: -30.995704217962388, p-value: 0.0

Test estacionariedad para serie con diferencia estacional (m=12)
----------------------------------------------------
ADF Statistic: -4.800855437342766, p-value: 5.418137971159215e-05

Test estacionariedad para serie con diferencia estacional y diferencia (order=1)
---------------------------------------------------
ADF Statistic: -4.281148773493745, p-value: 0.00047881368548043107

Test estacionariedad para serie con diferencia estacional y diferencia (order=2)
---------------------------------------------------

ADF Statistic: -7.827964743902786, p-value: 6.4057338238470425e-12
```

De los resultados impresos se extrae que el mejor valor de **d** resultaría ser 1 ya que ADF es más negativo en la primera diferenciación y el p-value es 0. Para **D** el mejor valor resultaría en 2 según ADF y p-value.

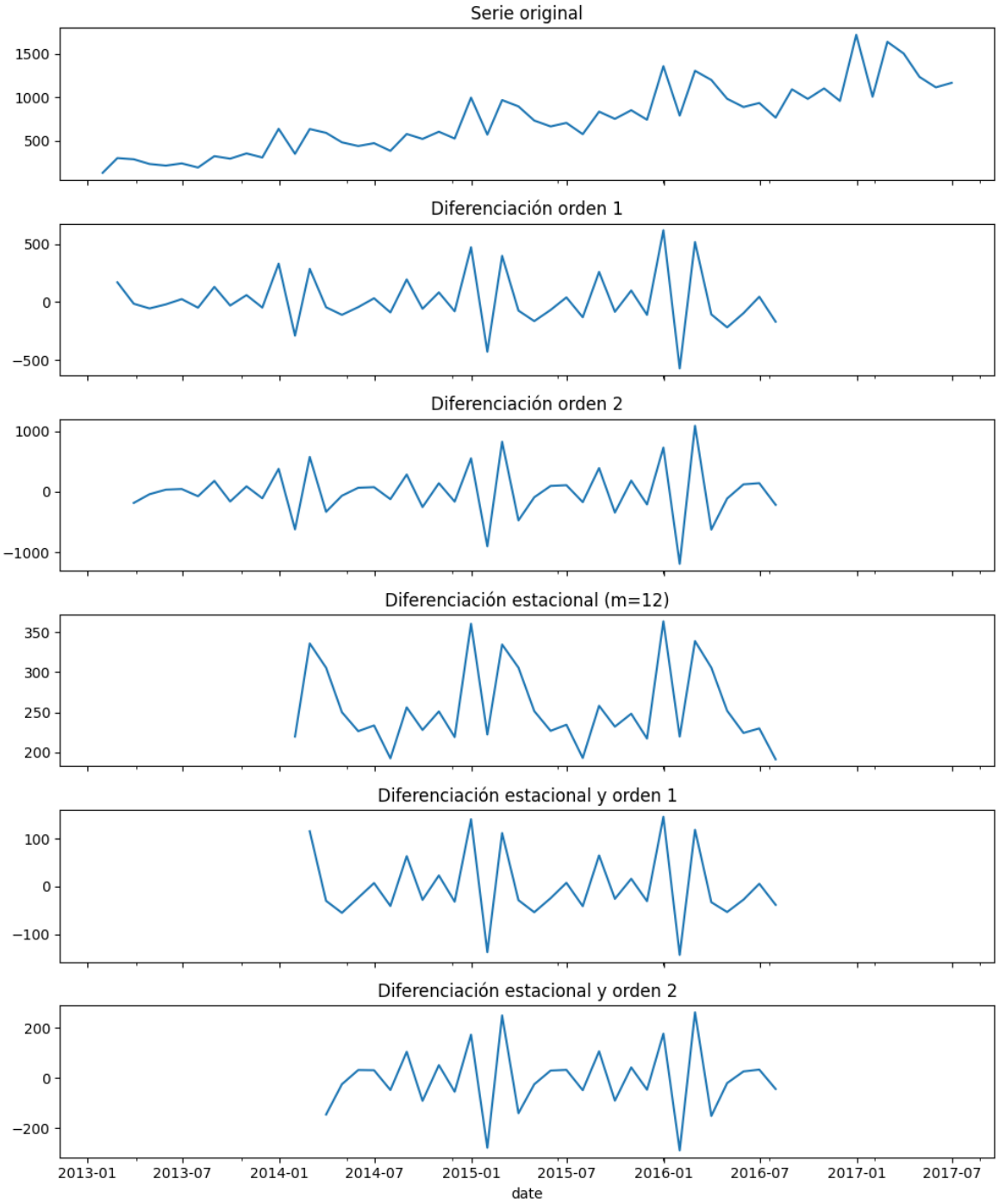

Figura 7-20 Gráficas de series diferenciadas de valores con patrones de estacionalidad.

P, Q, p y q se determinan siguiendo el mismo razonamiento que en el capítulo 9.9 de (Brendan Artley, 2022).

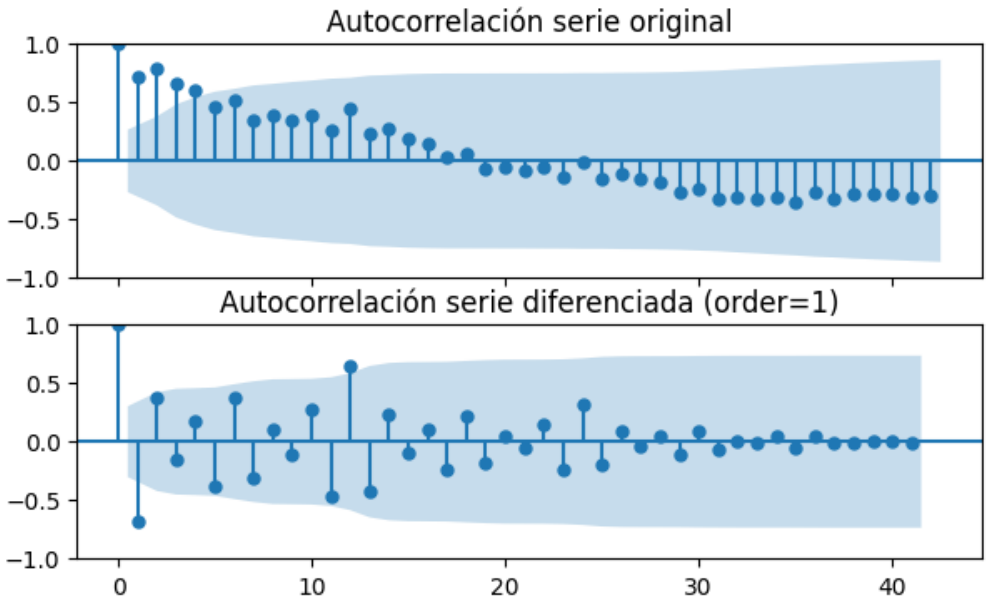

Figura 7-21 Gráfico ACF.

Si se analizan los resultados del gráfico ACF en la Figura 7-21, la espiga que sobresale en el retardo 2 de ACF de la serie diferenciada sugiere un término no estacional MA (2), las espigas sobresalientes en los retardos 12-13 sugieren un término estacional MA (1), es decir, se pueden interpretar valores q=2 y Q=1.

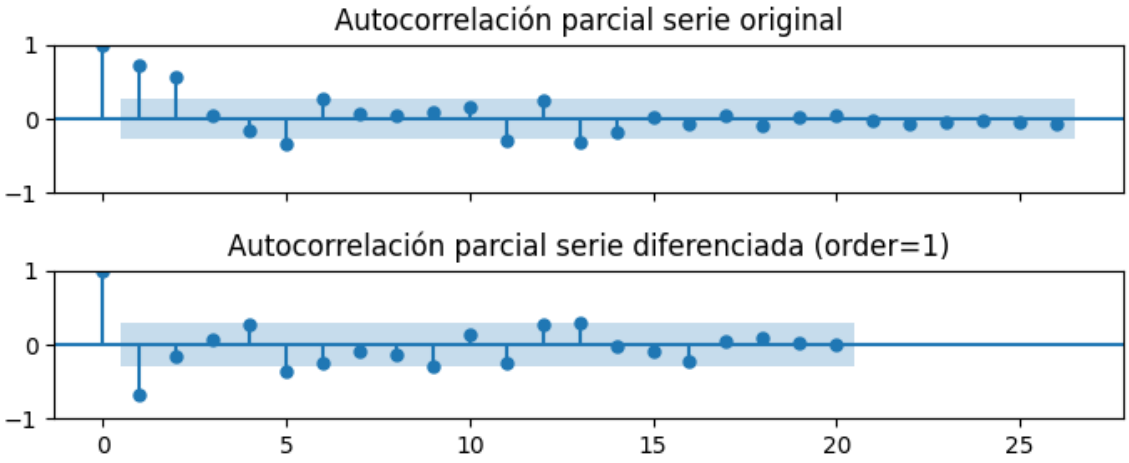

Figura 7-22 Gráfico PACF.

Si se analizan los resultados del gráfico PACF en la Figura 7-22, la espiga que sobresale en el retardo 2 de PACF de la serie diferenciada sugiere un término no estacional AR (2), las espigas sobresalientes en los retardos 12-13 sugiere el término AR (1), es decir, se pueden interpretar valores p = 2 y P = 1.

En última fila de la Tabla 7-4 se encuentran los resultados de los errores, se añaden los siguientes comentarios:

1. **ME (Mean Error) = - 3.370820**

Aunque el ME negativo sugiere una tendencia a subestimar los valores reales, este valor por sí solo no proporciona mucha información sobre la magnitud del error. Es importante considerarlo junto con otras métricas.

2. **MAE (Mean Absolute Error) = 39.774611**

Este valor es una medida directa de la precisión del modelo. Un MAE de 39.774611 indica que, en promedio, las predicciones están a 39.77 unidades del valor real.

3. **MPE (Mean Percentage Error) = - 0.012841**

Un MPE de - 0.012841 sugiere una subestimación muy leve en términos porcentuales. Es casi despreciable, lo

que indica que en términos relativos el modelo está bastante ajustado.

4. **MAPE (Mean Absolute Percentage Error) = 0.122471**

Un MAPE de 0.122471 (o 12.25%) indica que, en promedio, las predicciones están desviadas en un 12.25% del valor real.

5. **RMSE (Root Mean Squared Error) = 92.015886**

Un RMSE de 92.015886 indica una variabilidad considerable en los errores de las predicciones. Sugiere la presencia de algunos errores grandes en las predicciones.

6. **Correlación (corr) = 0.956156**

Una correlación de 0.956156 es muy alta, indicando que las predicciones del modelo tienen una relación lineal muy fuerte con los valores reales.

7. **MinMax = 0.103942**

Un valor de 0.103942 sugiere una variación relativa del 10.39% entre los valores mínimos y máximos predichos y los reales, indicando una buena precisión en términos de rango.

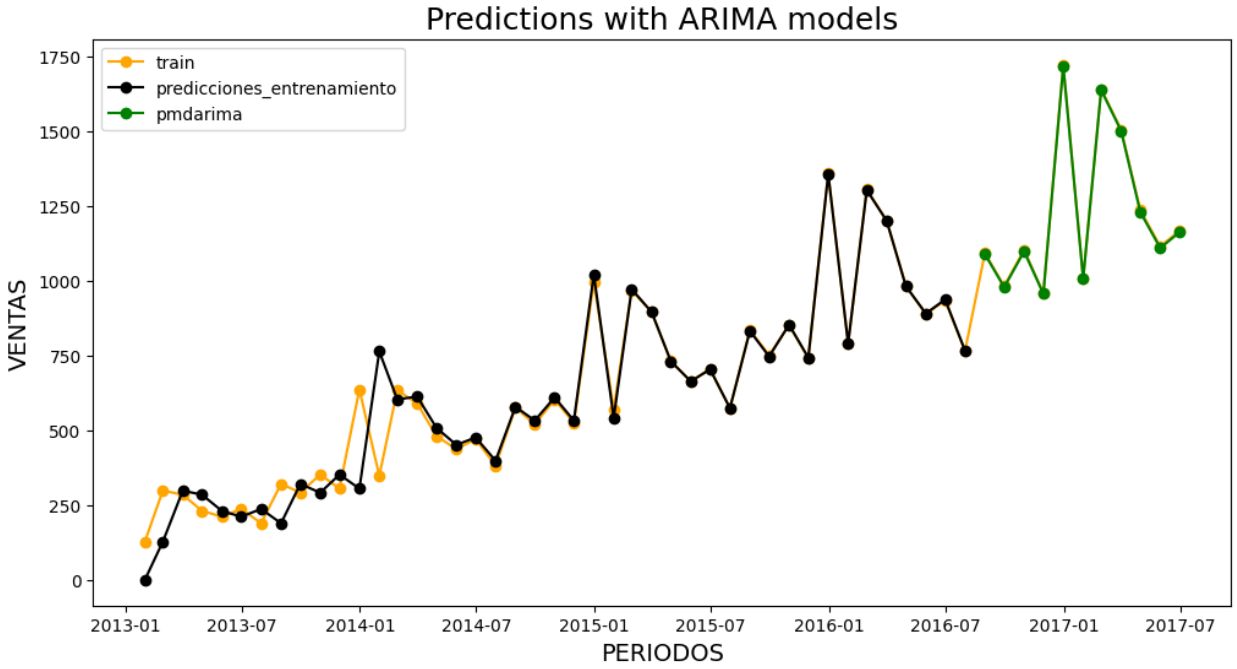

Figura 7-23 Representación de entrenamiento y predicción modelo Arima estacional.

7.1.2 Comparativa de resultados entre los métodos

Para realizar una comparativa justa basándonos en los resultados, se debe hacer entre los métodos expuestos para un mismo patrón de demanda.

Para los métodos con patrones de demanda constante se añaden los siguientes comentarios basados en los errores obtenidos:

El ajuste exponencial simple con alfa = 0.61 (óptimo) ofrece una mejora notable en términos de RMSE y un sesgo mínimo en comparación con los otros modelos, es decir, hay una pequeña diferencia promedio entre las predicciones del modelo y los valores reales. Aunque el MAPE y el MAE son similares a los del modelo de media móvil ponderada, la reducción en RMSE y la mejora en la correlación sugieren que este ajuste exponencial simple es más preciso y robusto. En general, este modelo parece ser el más adecuado entre los evaluados para prever la demanda en este contexto específico.

Para los métodos de patrones de demanda con tendencia se expone lo siguiente:

El modelo Arima No Estacional muestra una capacidad fuerte para seguir las tendencias de demanda, con un

coeficiente de correlación muy alto (0.9237), lo que indica que las predicciones siguen muy bien la tendencia general de los datos reales. Las métricas de error (MAE, MAPE, RMSE) son relativamente bajas en comparación con otros modelos, lo que sugiere una menor desviación y mayor precisión en las predicciones.

Para los métodos de patrones de demanda con estacionalidad se añade la siguiente conclusión:

El mejor modelo en términos generales es el Holt-Winters multiplicativo, ya que sobresale en casi todos los criterios evaluados: tiene el menor MAE, MAPE, RMSE, y la mayor correlación. Además, su MINMAX es el más bajo, indicando que sus predicciones tienen una dispersión mínima. Llegando por tanto a la conclusión de que el modelo más complejo no debe de ser siempre el más exacto ya que depende su mejor ajuste de la propia naturalidad de los datos, quizás si la estacionalidad de los datos no fuera tan evidente y más aleatoria, el modelo Arima que es más complejo, exibiría mejores resultados teniendo errores más bajos que los demás modelos y teniendo en cuenta que puede manejar variables exógenas es altamente probable que exhibiera mejor rendimiento.

8 CONCLUSIONES

El caso de estudio presentado se ha centrado en la previsión de la demanda de un producto comestible no perecedero de un supermercado de Salinas perteneciente a la cadena Supermaxi, los desarrollos se han realizado en el lenguaje de programación Python, en un entorno donde se puede interactuar añadiendo o modificando bloques de código existentes sin necesidad de tener nada instalado en un PC, esto permite disponer de mayor poder computacional sin necesidad de un PC de ciertas características.

Como conclusión global se puede exponer que ningún método es mejor que otro, sino que todos ellos son herramientas útiles para usar cuando se debe predecir algún tipo de demanda, aunque por norma general los métodos que mejor ajustan son los más complejos ya que se deben de manejar más variables y son un poco más modernos, es necesario advertir que las previsiones no van a ser nunca exactas por lo que se debe conocer el error estándar. Los métodos de previsión son una herramienta fundamental para anticipar la demanda junto con el conocimiento cualitativo de expertos y profesionales con años de experiencia en el sector donde se esté realizando un estudio de la previsión, pues los métodos se pueden usar con multitud de demandas diferentes y cada demanda de la vida real tiene sus propias características y particularidades. A pesar de la existencia de los modelos de previsión más complejos y exactos es necesario exponer que dada la facilidad que existe actualmente para calcular los métodos más sencillos es casi necesario que deban ser calculados, pues el poco poder computacional que emplean hace que merezca la pena conocer los resultados que estos métodos más sencillos puedan arrojar, métodos como la media móvil simple o media móvil ponderada para demanda constante o media móvil doble y ajuste exponencial doble para demandas con tendencia.

Como conclusión más particular se debe añadir que los mejores modelos entre los expuestos son los que están basados en Arima, aunque su precisión depende de las variables de entrada que estos métodos presentan, la selección de variables adecuadas junto al análisis de variables exógenas permitiría predecir con mayor exactitud los periodos futuros, sin embargo, como desventaja son mucho más complejos que los otros modelos debido al número de datos de entrada que se le deben incluir. La implementación de modelos Arima estacionales y otros métodos avanzados requiere un conocimiento profundo y puede ser una barrera para usuarios sin experiencia previa en análisis de series temporales.

La forma en la que está desarrollada el caso de estudio puede hacer pensar al lector que todos los métodos deben ser analizados con los mismos parámetros de errores, aunque es cierto que los diferentes tipos de errores son métricas confiables para comparar los modelos, existen otros tipos de métricas que devuelve como resultado las funciones de Arima en Python, estas métricas son AIC ó BIC que aunque se usan para determinar los mejores valores de las variables de entrada de Arima en este caso de estudio podrían usarse para obtener mejores conclusiones comparándo los demás modelos.

La implementación en Python utilizando Colab y librerías especializadas (como pandas, numpy, statsmodels, entre otras) ha facilitado la experimentación y comparación de los métodos de previsión. Python ha demostrado ser una herramienta potente y versátil para este tipo de análisis, permitiendo una fácil integración y visualización de los resultados, así como la capacidad de descomposición y visualización de las distintas partes que conforma a la demanda para conseguir un análisis más completo de elementos como tendencia o estacionalidad. Ha permitido también un rápido cálculo y ajuste de todos los modelos una vez obtenido el dataframe, aunque es cierto que se demora bastante tiempo en volcar los datos del archivo train.csv al dataframe donde se han alojado todos los datos diarios.

Como conclusión última añadir la importancia del uso adecuado de técnicas de previsión de la demanda ya que puede tener un impacto significativo en la optimización de la gestión logística y de la cadena de suministro. Predecir con precisión la futura demanda permite una mejor planificación de inventarios, reducción de costes asociados a productos desechados por ser perecederos o faltantes de stock, y mejora la eficiencia general del proceso industrial.

9 MEJORAS FUTURAS

Durante la realización del trabajo se identificaron una serie de posibles mejoras para obtener mejores resultados. También se incluyeron otras técnicas más actuales sobre previsión de demanda.

9.1. Mejoras en el proyecto

Para exponer mejor de cara al lector los métodos de previsión en Python sería conveniente que se explicase cómo funcionan las actuales librerías que se usan para creación de los modelos, es decir, que se explicase las formulas matemáticas que usan las clases, funciones y métodos que se exponen en este trabajo. Debido a que cuando se crea un modelo mediante una clase y se llama al método para que ajuste los valores de entrenamiento este crea o estima un valor inicial, es posible que los lectores puedan sentirse un tanto confusos respecto a cómo se ha obtenido ese valor.

Otra posible mejora sería la creación de una miniconsola que proporcione una interfaz visual con gráficas, errores y los valores de las predicciones de los métodos para que los usuarios que tengan más dificultades con el lenguaje de programación puedan interactuar con ello.

Para un mejor uso de los recursos computacionales, sería mejor el volcado de los archivos que se han usado durante el trabajo en una base de datos, así usando lenguajes de consulta SQL se podrían acceder más facilmente a los datos de entrenamiento sin necesidad de volcar los datos en un dataframe. Se ahorrarían más recursos computacionales lo que se traduciría en una ejecución más rápida y por tanto con una menor pérdida de tiempo.

9.2. Posibles mejoras en el campo de aplicación

Se exponen a continuación una serie de mejoras que podrían añadirse al Proyecto, de esta forma los usuarios podrían interactuar con modelos más actuales para predecir la demanda.

- Las redes neuronales recurrentes (RNN) son una clase de redes neuronales especializadas en el procesamiento de secuencias de datos. A diferencia de las redes neuronales tradicionales, las RNN pueden utilizar su memoria interna para procesar secuencias de entradas, lo que las puede hacer ideales para tareas que involucran datos temporales, como la previsión de demanda. Según (Corchado, 2000), las RNN se caracterizan por tener bucles que permiten que la información persista. En cada paso temporal, una RNN toma una entrada y la combinación del estado anterior, generando una salida y un nuevo estado que se retroalimenta en la red.

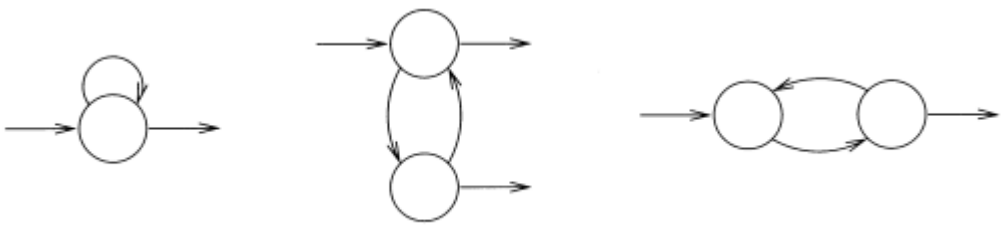

Figura 9-1 Conexiones recurrentes en neuronas (Corchado, 2000).

- Otra futura mejora que se podría incluir sería la adición de factores exógenos, de hecho, están en los archivos de la plataforma desde donde se descargaron los datos, factores como días festivos, ofertas o promociones de los productos, productos con faltas de existencias etcétera. Los modelos ARIMAX se caracterizan por la posibilidad de añadir variables exógenas al modelo. (Rob J Hyndman, n.d.). Para

no alargar el proyecto se decidió no incluir variables exógenas en los modelos Arima, incluyendo este tipo de variables los modelos pueden mejorar notablemente la precisión o exactitud de las previsiones dando resultados bastantes más cercanos a los valores reales.

- Se propone estudiar al detalle las métricas de exactitud (AIC, BIC o HQIC) en los métodos Arima, este estudio permitirá tener una visión más global sobre la exactitud del método en sí, no serviría de mucho compararlo con otros modelos puesto que ya se usan los distintos errores para ello, sino que más bien para escoger mejores 'inputs' en los modelos Arima y comparar resultados de distintos valores de entrada.
- Se comentó en el capítulo 7 la inclusion de métodos sistemáticos que mejoren los pesos de la media móvil ponderada de forma que seleccione o explore los mejores valores según el tipo de datos y el comportamiento que tomen los valores estudiados.
- También se puede incluir para la previsión técnicas como 'Random Forest' o Bosques Aleatorios, Random Forest es un algoritmo de aprendizaje automático que combina múltiples árboles de decisión para mejorar la precisión de predicciones y reducir el riesgo de sobreajuste. Se utiliza ampliamente en problemas de clasificación y regresión en datos estructurados. Como se expone en (Jason Brownlee, 2020), para aplicar Random Forest en la previsión de series temporales, es necesario transformar los datos en un formato supervisado utilizando una ventana deslizante. Este enfoque convierte el problema de serie temporal en uno adecuado para el aprendizaje supervisado.

BIBLIOGRAFÍA

Box, G. E. P., & Jenkins, G. (2016). *Time series analysis : forecasting and control* (G. E. P. Box, Ed.; Fifth edition.) [Book]. Wiley.

Brendan Artley. (2022). *Time Series Forecasting with Arima , SARIMA and SARIMAX*. https://towardsdatascience.com/time-series-forecasting-with-Arima-sArima-and-sArimax-ee61099e78f6

Burcu Özcan, & Ilhan Öztürk. (2019). *Environmental Kuznets Curve (EKC)*. Elsevier. https://doi.org/10.1016/C2018-0-00657-X

Chase, R. B. (2014). *Administración de operaciones : producción y cadena de suministros* (F. R. Jacobs & N. J. Aquilano, Eds.; 13ª ed.) [Book]. McGraw-Hill.

Corchado, J. M. (2000). *Redes neuronales artificiales : un enfoque práctico* (J. M. Corchado, Ed.) [Book]. Universidade de Vigo, Servicio de Publicacións.

Corporación Favorita. (2024). https://www.corporacionfavorita.com/

Cruz Fernández, A. (2018). *Planificación y gestión de la demanda* (1st ed.) [Book]. IC Editorial.

Gobierno de Ecuador. (2023). *Censo*. https://www.censoecuador.gob.ec/resultados-censo/

Google Maps. (2024).

Gujarati, D. N., & Porter, D. C. (2010). *Econometría*.

Hanke, J. E. (2010). *Pronósticos en los negocios* (D. W. Wichern & A. Enríquez Brito, Eds.; Novena edición.) [Book]. Pearson Educación.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Holt, C. C. (1957). *Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages*.

Hyndman, R. J. , & A. G. (2021). *Forecasting: Principles and Practice. (3rd ed.)*. https://otexts.com/fpp3/

Jason Brownlee. (2020). *Random Forest for Time Series Forecasting*.

Jason Brownlee. (2021). *Optimization for Machine Learning*. https://machinelearningmastery.com/function-optimization-with-scipy/

Jihyun (Jenny) Kim. (2024). *Exploring the Seasonal Arima (SARIMA) Model for Forecasting: Differences from Arima*. https://medium.com/@jihyun.kim423/exploring-the-seasonal-Arima-sArima-model-for-forecasting-differences-from-Arima-e30c3488e5f6

Joaquín Amat Rodrigo, & Javier Escobar Ortiz. (2023). *Modelos Arima y SARIMAX con Python*. https://www.cienciadedatos.net/documentos/py51-modelos-Arima-sArimax-python.html

Jorge A. Perez Prieto. (2019). *Curso de Python para Astronomía*. Https://Research.Iac.Es/Sieinvens/Python-Course/Index.Html.

Kaggle, G. L. (2017). *Corporación Favorita Grocery Sales Forecasting*. https://www.kaggle.com/c/favorita-grocery-sales-forecasting/overview

Onieva Giménez, L. (2017). *Diseño y gestión de sistemas productivos* (A. Escudero Santana, P. Cortés Achedad, J. Muñuzuri Sanz, & J. Guadix Martín, Eds.) [Book]. Dextra.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). *Scikit-learn: Machine Learning in Python* (pp. 2825–2830).

Peña, D. (2005). *Análisis de series temporales*. Alianza Editorial.

Rob J Hyndman. (n.d.). *The ARIMAX model muddle*. 2010. Retrieved May 28, 2024, from https://robjhyndman.com/hyndsight/Arimax/

Sánchez Fernández, J. (2004). *Introducción a la Estadística Empresarial. Capítulo 4.-Series temporales*.

Seabold, Skipper, & Josef Perktold. (2010). *statsmodels: Econometric and statistical modeling with python*. Proceedings of the 9th Python in Science Conference. https://www.statsmodels.org/dev/index.html

Supermaxi. (2024). https://www.supermaxi.com/

Taylor G. Smith and others. (2017). *Arima estimators for Python*. http://www.alkaline-ml.com/pmdArima

The pandas development team. (2020, February). *pandas-dev/pandas: Pandas*. https://pandas.pydata.org/

Van Rossum, G. (2020). *The Python Library Reference, release 3.8.2*. Python Software Foundation.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Vázquez-Baeza, Y. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Winters, P. R. (1960). *Forecasting Sales by Exponentially Weighted Moving Averages*.