

Optimización del inventario en tiendas minoristas a partir de modelos de pronóstico de demanda.

07 Abril 2025

The logo consists of the lowercase letters "viu" in white, centered within a solid orange circle.

viu

Universidad
Internacional
de Valencia

Titulación:

Máster en Inteligencia Artificial
Curso académico
2024 – 2025

Alumna:

Murias Palomer, María Soledad
D.N.I: F46325157

Director de TFM: Jesús Cigales
Canga

Convocatoria:

Tercera

De:

 Planeta Formación y Universidades

Índice

Resumen	7
Abstract	8
1. Introducción	9
1.1. Motivación del TFM	11
1.2. Estructura general del documento	11
2. Objetivos.....	13
2.1. Objetivo general	13
2.2. Objetivos específicos.....	13
3. Marco teórico y Estado del Arte	14
3.1. Minería de datos.....	14
3.2. <i>Forecast</i> de venta.....	17
3.2.1. Regresión lineal.....	18
3.2.2. Random Forest.....	19
3.2.3. Gradient Boosting Machine.....	19
3.2.4. Series de tiempo.....	21
3.2.5. Arquitecturas de aprendizaje profundo	22
3.3. Optimización de inventario	26
4. Desarrollo del proyecto y resultados.....	30
4.1. Planteamiento del problema	30
4.2. Desarrollo del proyecto.....	31
4.2.1. Comprensión del negocio	32
4.2.2. Comprensión de los datos	34
4.2.3. Preparación de los datos	37
4.2.4. Modelado.....	41
4.3. Resultados	47
4.3.1. <i>Forecast</i> de venta	47
4.3.2. Optimización de inventario.....	52
5. Conclusión y trabajos futuros.....	54
6. Glosario	56
7. Referencias	57
Apéndice I.....	62
Apéndice II.....	64



Apéndice III	65
Apéndice IV	66
Apéndice V	69
Apéndice VI	73
Apéndice VII	77

Índice de ilustraciones

Ilustración 1. Venta anual de los Estados Unidos entre los años 2010 y 2021 (Zippia, 2023)	9
Ilustración 2. Distribución del Producto Interno Bruto (PIB) de Chile por región, año 2023. Elaboración propia basada en los datos de Banco Central de Chile (n.d.)	10
Ilustración 3. Etapas de la metodología CRISP-DM para minería de datos (Shafique & Qaiser, 2014)	14
Ilustración 4. Evolución de procesos y metodologías de minería de datos (Plotnikova et al., 2020)	15
Ilustración 5. Beneficios de realizar predicción de ventas (Malik et al., 2024)	17
Ilustración 6. Clasificación de métodos de forecast según características de los datos (Punia et al., 2020)	18
Ilustración 7. Pseudocódigo de algoritmo Random Forest (Dudek, 2015)	19
Ilustración 8. Pseudocódigo del algoritmo de Grandient Boosting (Natekin & Knoll, 2013).	20
Ilustración 9. representación de la ecuación de SARIMA para cuatrimestres (s=4) (Hyndman & Athanasopoulos, 2018)	21
Ilustración 10. Esquema representativo de un DNN (Benidis et al., 2022)	22
Ilustración 11. Estructura de una CNN formada por una pila de tres capas convolucionales causales. La capa de entrada (verde) es no dilatada y las otras dos son dilatadas (Benidis et al., 2022)	23
Ilustración 12. Estructura de una RNN expandida, donde cada unidad recibe una entrada externa a_t y la salida de las unidades ocultas del paso temporal anterior h_{t-1}	24
Ilustración 13. Esquema de la unidad de una RNN (a) versus una red LSTM (b) creada a partir de imágenes disponibles en (Hewamalage et al., 2021). En ambos esquemas, x_t representa la entrada de la unidad en el instante t, z_t representa la salida de la unidad y h_t representa el estado oculto de la unidad.....	24
Ilustración 14. Esquema de la unidad de una red GRU (Hewamalage et al., 2021) ...	25
Ilustración 15. Esquema comparativo entre redes de eslabón único y redes de múltiples eslabones (Brightwork Research & Analysis, n.d.)	27
Ilustración 16. Representación gráfica de la estrategia de punto de recompra, donde S representa el punto de recompra, L el tiempo de servicio y R el periodo de revisión (Arkaresvimon, 2008).....	28
Ilustración 17. Flujo operacional de algoritmo genético según propuesta de Radhakrishnan et al., 2009	29
Ilustración 18. Plano de local de ventas, donde se diferencia la sala de ventas de la bodega	33
Ilustración 19. Diagrama comparativo de calzados atemporales y de temporada, para hombre y mujer.....	34
Ilustración 20. Unidades vendidas por Subfamilia y Familia entre el 01/01/23 y el 20/02/25.....	35
Ilustración 21. Información estadística de la venta diaria en unidades	36
Ilustración 22. Muestra base de datos de ventas de Guante y Gacel.....	36

Ilustración 23. Información estadística de la venta diaria en unidades luego de regularización	37
Ilustración 24. Unidades vendidas por evento comercial.....	38
Ilustración 25. Muestra base de datos de ventas agrupada por categoría (New_Cluster)	39
Ilustración 26. Gráficos de caja y bigote para la venta diaria de cada combinación Marca-Subfamilia.....	40
Ilustración 27. Diagrama ilustrativo del funcionamiento de One Hot Encoding (elaboración propia).....	41
Ilustración 28. Informe entregado por ARIMA tras su ajuste de hiperparámetros.....	48
Ilustración 29. Gráfica comparativa de datos de prueba y datos simulados con ARIMA	49
Ilustración 30. Listado de registros reales y predichos con ARIMA con mayor MAPE.50	
Ilustración 31. Gráfica comparativa de datos de prueba y datos simulados con XGBoost	51
Ilustración 32. Simulación de inventario diario (línea azul) por local usando la metodología EOQ con horizonte de 25 días, junto a la capacidad máxima por local (rojo) y la venta diaria (verde).	52
Ilustración 33. Simulación de inventario diario por local usando algoritmo de optimización genético (línea azul), junto a la capacidad máxima por local (rojo) y la venta diaria (verde).	52

Índice de tablas

Tabla 1: Aspectos clave de metodologías de minería de datos (adaptado de Plotnikova et al., 2020).....	16
Tabla 2. Características de variantes de algoritmo GBM. Elaboración propia.	20
Tabla 3. Valores nulos por columna para base de datos de ventas	37
Tabla 4. Valores nulos por columna para base de datos de ventas luego de regularización	38
Tabla 5. Resumen de métricas estadísticas de predicción de venta realizada con ARIMA	49
Tabla 6. Resumen actualizado de métricas estadísticas de predicción de venta realizada con ARIMA	50
Tabla 7. Comparación de métricas estadísticas de predicción de venta realizada con ARIMA y XGBoost	51
Tabla 8. Comparación de costos obtenidos con modelo de inventario EOQ y algoritmo de optimización genético.	53

Resumen

Muchas empresas de *retail* no cuenta con recursos para contratar *softwares* de predicción de venta y optimización de inventario. Esto lleva al desarrollo de metodologías manuales con gran participación humana, lo que puede generar errores con serias consecuencias económicas. Esto es particularmente cierto en la industria de la moda, donde los cortos ciclos de vida de los productos vuelven crítica la correcta distribución de inventario. En este trabajo se desarrolló una herramienta accesible para predecir la venta futura y optimizar el inventario de una red de tiendas minoristas con el fin de mejorar la satisfacción de los clientes y reducir costos de transporte.

Para predecir la venta futura, se compararon los algoritmos ARIMA y XGBoost entrenados con datos reales de una empresa comercializadora de calzados chilena. Para ambos casos, las métricas estadísticas, particularmente MAPE, se vieron afectadas por aquellos productos de menor volumen de venta. Sin embargo, XGBoost mostró un mejor ajuste a los datos reales y fue utilizado para la siguiente fase del trabajo.

La optimización de inventario se llevó a cabo comparando un modelo basado en puntos de recompra, o EOQ, con un algoritmo metaheurístico basado en la evolución de Darwin (algoritmo genético). Se tomaron en cuenta costos de transporte, venta perdida y una penalización por superar la capacidad máxima de cada local. El algoritmo genético fue capaz de obtener la solución con menor coste general, además de ser una opción menos rígida en escenarios con múltiples eslabones y productos.

Palabras clave: Optimización de inventario, Forecast de venta, XGBoost, ARIMA, Punto de recompra, Algoritmo genético, Inventario multi-eslabón.



Abstract

Many retail companies do not have the resources to hire sales prediction and inventory optimization software. This leads to the development of manual methodologies with significant human involvement, which can result in errors with serious economic consequences. This is particularly true in the fashion industry, where the short product life cycles make correct inventory distribution critical. In this work, an accessible tool was developed to predict future sales and optimize inventory for a retail network of stores, to improve customer satisfaction and reduce transportation costs.

To predict future sales, ARIMA and XGBoost algorithms were compared, trained with real data from a Chilean footwear distributor. In both cases, statistical metrics, particularly MAPE, were affected by products with lower sales volume. However, XGBoost showed a better fit to the real data and was used for the next phase of the work.

Inventory optimization was carried out by comparing a model based on reorder points, or EOQ, with a metaheuristic algorithm based on Darwin's evolution (genetic algorithm). Transportation costs, lost sales, and a penalty for exceeding the maximum capacity of each store were considered. The genetic algorithm was able to find the solution with the lowest overall cost, in addition to being a more flexible option in scenarios with multiple stores and products.

Keywords: Inventory optimization, Sales forecast, XGBoost, ARIMA, Reorder point, Genetic algorithm, Multi-echelon inventory.

1. Introducción

Tradicionalmente, la cadena de abastecimiento era dominada por los manufactureros y productores, mientras que distribuidores y comercios debían adaptarse a las prioridades de los primeros (Randall *et al.*, 2011). Sin embargo, el auge de las grandes marcas y tiendas por departamento ha cambiado el foco a aquellas organizaciones cercanas al cliente final (Brodie *et al.*, 2009). En este tipo de negocios, la disponibilidad del producto correcto en el lugar indicado es clave para la satisfacción del cliente y, por lo tanto, se requiere que la información para la toma de decisiones esté disponible para los distintos actores de la cadena (Afshari y Benam, 2011).

Por otro lado, la industria de la moda ha tenido un crecimiento sostenido (exceptuando la pandemia) en los últimos 30 años. Entre el 2000 y 2014, la cantidad de prendas compradas por persona aumentó un 60% y la producción de vestuario se duplicó (McKinsey & Company, 2023). Esto se explica principalmente por el surgimiento del *fast fashion*, y ha convertido a la industria de la moda en una de las más relevantes, aportando el 2% del GDP global (Zippia, 2023). La velocidad del mercado y de la obsolescencia de los productos de vestuario y calzado hacen aún más relevante poder predecir dónde y cuándo ocurrirá la demanda (Wen *et al.*, 2019).

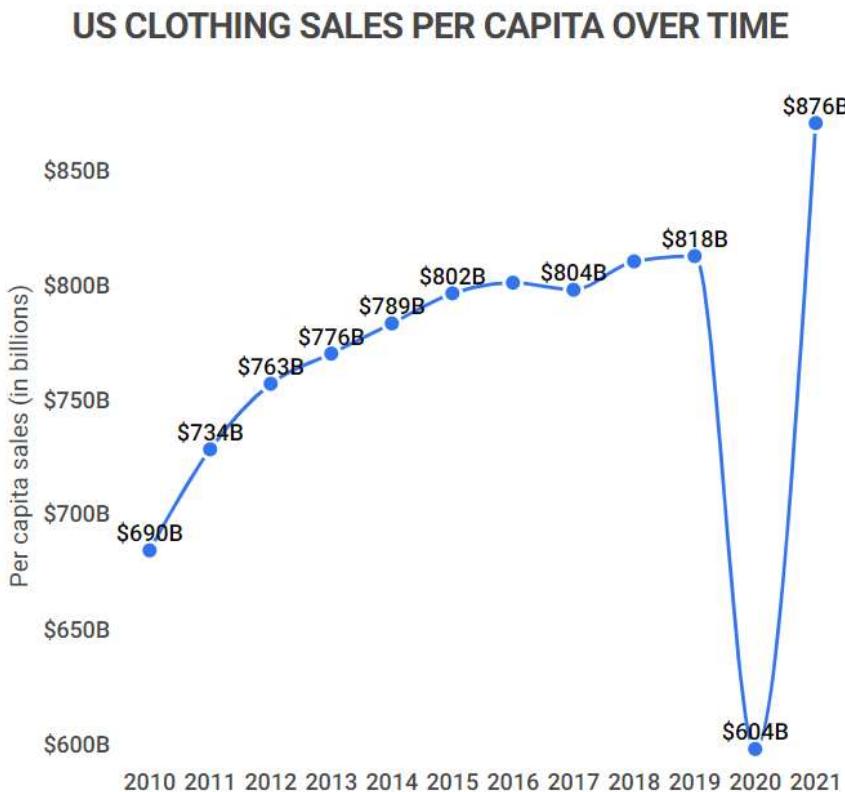


Ilustración 1. Venta anual de los Estados Unidos entre los años 2010 y 2021 (Zippia, 2023).

En Chile, La Región Metropolitana (R.M.), que contiene a la capital, Santiago de Chile, concentró el 43% del PIB nacional al año 2023, con un porcentaje de la población total similar (Banco Central de Chile, n.d.). La siguiente región que le sigue, Antofagasta, sólo alcanza un 12% del PIB total. Adicionalmente, la R.M. es hogar del mayor aeropuerto de Chile y está a 2 horas en camión de los puertos más importantes del país. Esto, junto a su geografía alargada, hace que Chile tenga una situación única en términos de cadena de abastecimiento: la mayor parte de las empresas de *retail* no tienen centros de distribución logístico fuera de la Región Metropolitana. Esto puede evidenciarse en una encuesta a operadores logísticos realizada por la Revista Logistec, donde un 92% indicó tener operaciones en la R.M., pero la siguiente región en importancia sólo tuvo un 38% de presencia (Revista Logistec, 2024). Es decir, no hay actualmente mercado para arriendo de bodegas en otras regiones.

Distribución del Producto Interno Bruto (PIB) de Chile por región, año 2023.

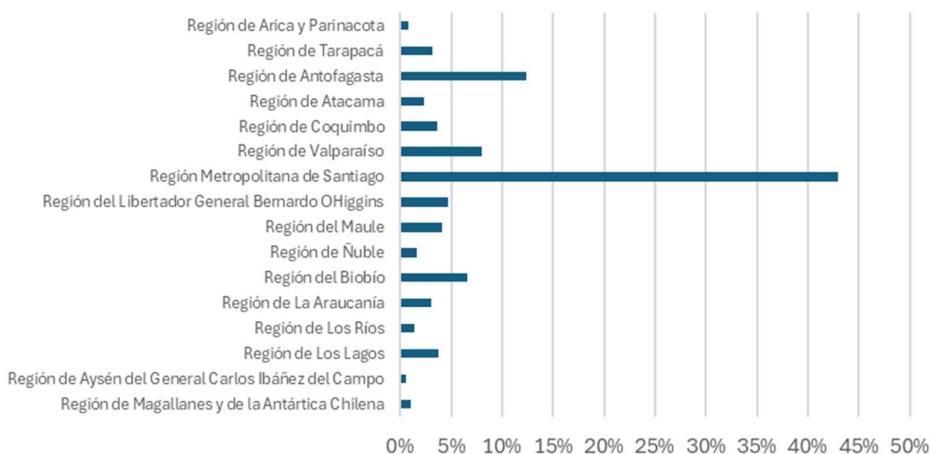


Ilustración 2. Distribución del Producto Interno Bruto (PIB) de Chile por región, año 2023. Elaboración propia basada en los datos de Banco Central de Chile (n.d.).

Así es como la mayoría de las empresas de *retail* en Chile trabajan con un Centro de Distribución único ubicado en la Región Metropolitana y manejan inventario en cada una de sus sucursales de venta. Por lo tanto, tiene una gran relevancia en los costos de cada empresa poder establecer un inventario óptimo en cada sucursal que maximice el nivel de servicio al mínimo coste (Macas *et al.*, 2021). Existen tecnologías que pueden apoyar en el manejo del inventario, pero suelen ser costosa. Una radiografía de empresas de comercio minorista en Chile muestra que un 89% de las empresas tienen ingresos anuales por menos de USD 1 millón, y que un 42% ingresa menos de USD 100 mil al año (Ministerio de Economía, Fomento y Turismo, 2024). Por lo tanto, son pocas las empresas que pueden costear un software con la inteligencia suficiente para predecir la venta por local y sugerir el inventario correcto para cada uno.

Para intentar resolver este problema, este proyecto se centra en diseñar un sistema que permita analizar la venta por sucursal, predecir la venta futura según patrones históricos

y la tendencia actual, y entregar una sugerencia de inventario de acuerdo con los parámetros de capacidad del local y frecuencia de reposición.

1.1. Motivación del TFM

El principal motor detrás de este Trabajo Final de Máster es mejorar la eficiencia y precisión en la toma de decisiones relacionadas con la distribución de inventario en el sector *retail*. Actualmente, muchas empresas dependen de analistas que, día tras día, deben examinar grandes volúmenes de datos de ventas e inventario para asignar manualmente el *stock* óptimo por SKU y por local. Este proceso no solo es altamente demandante en términos de tiempo, sino que también es propenso a errores y limita el verdadero potencial de los analistas, quienes podrían dedicar sus esfuerzos a tareas de mayor valor estratégico.

Cuando las empresas no pueden costear soluciones avanzadas de *software* o cuando las herramientas disponibles no cumplen con las expectativas de uso, los procesos manuales terminan prevaleciendo. Esto genera ineficiencias que afectan la rentabilidad y el desempeño del negocio. En el *retail*, donde los ciclos de vida de los productos son extremadamente cortos, la precisión en la asignación de inventario es clave para garantizar una temporada exitosa. Una mala distribución del *stock* puede resultar en productos quedándose sin vender en tiendas de baja demanda o en oportunidades perdidas en locales con alta rotación, afectando tanto la experiencia del cliente como los resultados financieros.

Desde un punto de vista técnico, este proyecto presenta un doble desafío: primero, desarrollar un modelo de *forecast* capaz de predecir la demanda de cada tienda con la mayor precisión posible; y segundo, diseñar un modelo de optimización que asigne el inventario de manera estratégica, minimizando costos y maximizando la disponibilidad de productos en los locales adecuados. Además de su impacto práctico, este trabajo representa una oportunidad invaluable para aplicar los conocimientos adquiridos a lo largo del máster. Implementar técnicas avanzadas de minería de datos, *machine learning* y optimización en un caso real permite no solo afianzar los conceptos estudiados, sino también contribuir con una solución concreta a un problema común en la industria.

1.2. Estructura general del documento

Este trabajo se compone de cinco capítulos en los cuales se expondrá el análisis realizado y sus conclusiones. Adicionalmente, tiene un sexto capítulo donde se listarán las referencias utilizadas a lo largo del mismo y luego anexos con información adicional pertinente.

En el segundo capítulo se presenta el objetivo general del trabajo y sus objetivos específicos, para luego exponer en el capítulo tres el estado actual de la literatura



científica relacionada al proyecto. El foco estará en aplicaciones de *deep learning* en *retail, forecast* de venta y estimación de inventario.

El cuarto capítulo corresponde al grueso del trabajo y es ahí donde se expondrá el desarrollo del proyecto y los resultados obtenidos. Éste estará separado por secciones que profundicen en cada uno de los distintos aspectos estudiados en el trabajo. Para terminar, en el capítulo cinco se recopilarán las conclusiones que desprenden de los resultados obtenidos y se plantearán acciones futuras que permitan continuar y mejorar los algoritmos y su aplicación.

2. Objetivos

2.1. Objetivo general

- Desarrollar un enfoque integrado de pronóstico de demanda y optimización de inventario para la toma de decisiones en la gestión de una red de tiendas minoristas, considerando restricciones logísticas y de capacidad.

2.2. Objetivos específicos

- Diseñar un mecanismo eficiente para recopilar y consolidar los datos de ventas históricas e inventarios actuales en toda la cadena (centro de distribución y red de tiendas).
- Implementar modelos de predicción para estimar la demanda futura de cada producto por tienda con alta precisión.
- Evaluar los modelos de predicción de demanda entrenados según métricas estadísticas
- Incorporar las limitaciones de capacidad de almacenamiento en cada tienda y los costos asociados al transporte en el diseño del sistema.
- Diseñar un algoritmo que determine el inventario óptimo para cada tienda, maximizando el nivel de servicio al cliente y minimizando los costos de transporte.
- Crear un sistema integrado que combine las predicciones de demanda con el modelo de optimización para generar sugerencias de inventario adaptadas a las condiciones operativas actuales.
- Realizar pruebas con datos reales para evaluar la precisión de las predicciones, la calidad de las decisiones de inventario y el cumplimiento de las restricciones.

3. Marco teórico y Estado del Arte

3.1. Minería de datos

La extracción de conocimientos a partir de datos se conoce como minería de datos, o *data mining*. Su objetivo es analizar grandes volúmenes de datos para identificar patrones, tendencias, relaciones y conocimientos útiles que no son evidentes a simple vista. Este proceso combina técnicas de estadística, aprendizaje automático e inteligencia artificial para descubrir patrones significativos.

Las metodologías de minería de datos se remontan a mediados de la década de los 90s, cuando Fayyad *et al.* (1996) proponen KDD (*Knowledge Discovery from Databases*). KDD presenta un modelo de proceso conceptual para la extracción de información a partir de datos. Este enfoque dominó la academia y la industria hasta el año 2000, cuando Chapman *et al.* (2000) introdujeron CRISP-DM (*Cross-Industry Standard Process for Data Mining*). Este nuevo proceso surgió como respuesta a las limitaciones de KDD en aplicaciones industriales, enfocándose en una mayor alineación con los objetivos de negocio. Las etapas de CRISP-DM son: comprensión del negocio, comprensión de los datos, preparación de los datos, modelado, evaluación y despliegue, como se ve en la Ilustración 3.

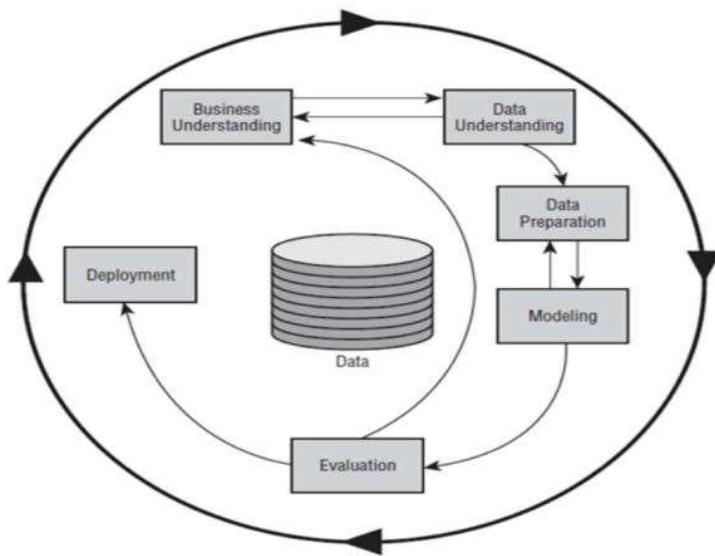


Ilustración 3. Etapas de la metodología CRISP-DM para minería de datos (Shafique & Qaiser, 2014)

Dentro de las metodologías más usadas se encuentra KDD (*Knowledge Discovery from Databases*) y CRISP-DM. La primera tiene un foco más académico que la segunda, ya que esta última incluye etapas de comprensión del negocio y puesta en marcha que sí son necesarias en una aplicación empresarial (Shafique & Qaiser, 2014).

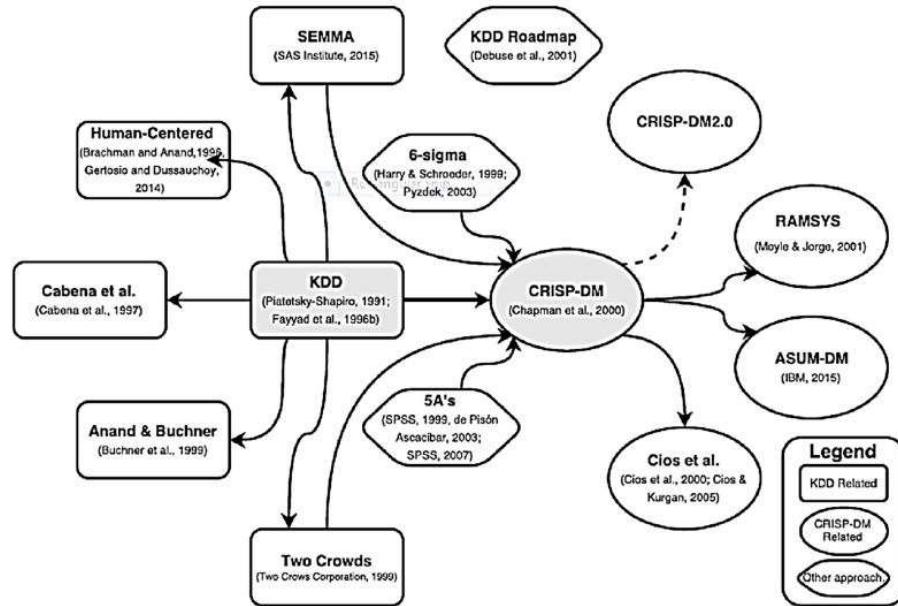


Ilustración 4. Evolución de procesos y metodologías de minería de datos (Plotnikova et al., 2020).

Como se ve en la Ilustración 4, ambos métodos han dado origen a una serie de procesos alternativos a lo largo de los años. Luego, en la Tabla 1, se describen algunas características relevantes de cada uno.

Nombre	Origen	Base	Concepto clave	Año
Human-Centered	Acade-mia	KDD	Proceso iterativo e interactividad (perspectiva del usuario y decisiones necesarias)	1996, 2004
Cabena et al.	Acade-mia	KDD	Enfoque en el procesamiento de datos y tareas de descubrimiento	1997
Anand y Buchner	Acade-mia	KDD	Pasos suplementarios e integración de minería web	1998, 1999
Two Crows	Industria	KDD	Definiciones modificadas de los pasos	1998
SEMMA	Industria	KDD	Específico de herramienta (SAS Institute), elimina algunos pasos	1998
5 A's	Industria	Inde-pen-diente	Pasos suplementarios	2003
6 Sigmas	Industria	Inde-pen-diente	Integración del paradigma de mejora de calidad Seis Sigma con el modelo DMAIC	2003

CRISP-DM	Industria y academia	KDD	Ejecución iterativa de pasos, refinamientos significativos en tareas y resultados	2000
Cios et al.	Academia	Crisp-DM	Integración de minería de datos y descubrimiento de conocimiento, mecanismos de retroalimentación, uso de conocimientos recibidos respaldados por tecnologías	2005
RAMSYS	Academia	Crisp-DM	Integración de aspectos colaborativos de trabajo	2001–2002
DMIE	Academia	Crisp-DM	Integración y adaptación al dominio de la Ingeniería Industrial	2001
Marban	Academia	Crisp-DM	Integración y adaptación al dominio de la Ingeniería de Software	2001
KDD road-map	Industria y academia	Independiente	Específico de herramienta, tarea de reasignación	2001
ASUM	Industria	Crisp-DM	Específico de herramienta, combinación de Crisp-DM tradicional con un enfoque ágil de implementación	2015

Tabla 1: Aspectos clave de metodologías de minería de datos (adaptado de Plotnikova et al., 2020).

3.2. Forecast de venta

La correcta predicción de la venta y asignación de recursos tiene un gran impacto en el éxito de los negocios. Particularmente, en el rubro de la moda, la mayoría de los productos son reemplazados en la temporada siguiente, por lo que no hay información histórica de ventas para cada ítem (Choi *et al.*, 2013). Esto implica que se usan predicciones para tomar decisiones cuyo impacto se verá en el corto plazo, sin muchas oportunidades de corrección. Para enfrentar este problema es que los métodos de predicción de ventas han sido ampliamente estudiados.



Ilustración 5. Beneficios de realizar predicción de ventas (Malik *et al.*, 2024).

Los métodos estadísticos, como suavizamiento exponencial, modelos de regresión o *Random Forest* (RF) han sido utilizados con mayor frecuencia (Beheshti-Kashi *et al.*, 2015) debido a su simplicidad o baja necesidad de recursos computacionales. También se utilizan modelos híbridos, que combinan diferentes enfoques para generar nuevas técnicas, para enfrentar dificultades como la simulación en el muy corto plazo o la de nuevos productos (Liu *et al.*, 2013).

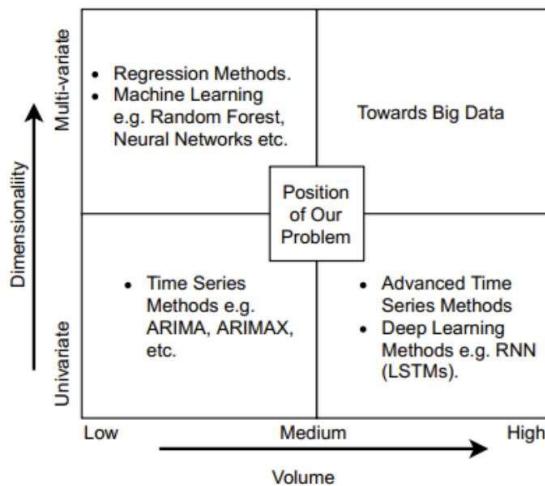


Ilustración 6. Clasificación de métodos de forecast según características de los datos (Punia et al., 2020)

En el último tiempo, han surgido nuevos modelos que hacen uso de redes neuronales (Eglite & Birzniece, 2022) para simular relaciones no lineales entre los datos. Algunas de estas técnicas incluyen *Convolutional Neural Networks* (CNN), *Long Short-Term Memory* (LSTM), y *Recurrent Neural Network* (RNN). Según los recursos y la disponibilidad de datos, se pueden usar modelos de predicción de venta estadísticos, heurísticos o híbridos. A continuación, se presentan los más relevantes.

3.2.1. Regresión lineal

Esta técnica fue desarrollada en los años 60 para estimar la probabilidad de ocurrencia de un proceso en función de otras variables (Domínguez-Almendros et al., 2011). Cuando la variable a estimar es binaria, nos referiremos a una regresión logística. De la misma forma, si el problema es multiclase, hablaremos de una regresión logística multinomial.

La regresión lineal ha sido utilizada con éxito en la predicción de venta para el retail. Cooper et al. (1999) desarrollaron el modelo PromoCast, que utiliza regresión lineal para predecir ventas durante promociones en un entorno minorista. Este modelo demostró ser efectivo al incorporar variables como descuentos, exhibiciones y efectos de temporada. Ali et al. (2009) evaluaron 30 métodos de predicción de ventas de SKU en un entorno de supermercados europeos y concluyeron que los modelos de regresión lineal con *pooling* de datos entre tiendas y categorías ofrecían buenos resultados de precisión.

La regresión lineal es un enfoque simple, fácil y rápido de ajustar, lo que la hace viable para problemas de pronóstico de ventas a gran escala en el nivel de producto. Sin embargo, como indica Fildes et al. (2022), este tipo de modelos tiende al sobreajuste ya que se usan cuando la disponibilidad de datos es baja.

3.2.2. Random Forest

Es un método de aprendizaje en conjunto que combina múltiples árboles de decisión binarios (*Classification And Regression Trees*, CART, en inglés) construidos en subconjuntos aleatorios de los datos y características (Dudek, 2015). Cada árbol se entrena con una muestra del conjunto de datos y, en cada nodo, se selecciona aleatoriamente un subconjunto de predictores para buscar el mejor punto de división. Los árboles no se podan y se crecen hasta su tamaño máximo.

1. For $k = 1$ to K :
 - 1.1. Draw a bootstrap sample L of size N from the training data.
 - 1.2. Grow a random-forest tree T_k to the bootstrapped data, by recursively repeating the following steps for each node of the tree, until the minimum node size m is reached.
 - 1.2.1. Select F variables at random from the n variables.
 - 1.2.2. Pick the best variable/split-point among the F .
 - 1.2.3. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_k\}_{k=1, 2, \dots, K}$.

To make a prediction at a new point \mathbf{x} :

$$f(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^K T_k(\mathbf{x}) \quad (1)$$

Ilustración 7. Pseudocódigo de algoritmo Random Forest (Dudek, 2015).

Esta técnica se ha probado con éxito en la literatura para hacer *forecast* de ventas en retail (Loureiro *et al.*, 2018; Eglite & Birzniece, 2022). Incluso, en los resultados de Loureiro *et al.* (2018), afirman que RF obtuvo mejor desempeño en algunas métricas evaluativas que, por ejemplo, el uso de redes neuronales.

3.2.3. Gradient Boosting Machine

Es una técnica de aprendizaje supervisado que construye un modelo predictivo fuerte a partir de una combinación secuencial de modelos débiles, generalmente árboles de decisión (Natekin & Knoll, 2013). El principio fundamental es minimizar una función de pérdida específica mediante gradientes descendentes iterativos.

Algorithm 1 Friedman's Gradient Boost algorithm

Inputs:

- input data $(x, y)_{i=1}^N$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

- 1: initialize \hat{f}_0 with a constant
 - 2: **for** $t = 1$ to M **do**
 - 3: compute the negative gradient $g_t(x)$
 - 4: fit a new base-learner function $h(x, \theta_t)$
 - 5: find the best gradient descent step-size ρ_t :
- $$\rho_t = \arg \min_{\rho} \sum_{i=1}^N \Psi[y_i, \hat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)]$$
- 6: update the function estimate:

$$\hat{f}_t \leftarrow \hat{f}_{t-1} + \rho_t h(x, \theta_t)$$
 - 7: **end for**
-

Ilustración 8. Pseudocódigo del algoritmo de Grandient Boosting (Natekin & Knoll, 2013).

Este algoritmo tiene distintas variantes que han surgido en respuesta a deficiencias del modelo inicial. Algunos de ellos son XGBoost, LightGBM y CatBoost (Szczepanek, 2022). El primero, el más frecuentemente usado, está diseñado para reducir el sobreajuste e incluye mejoras en velocidad, eficiencia y manejo de datos respecto al modelo base. El segundo, LightGBM, fue diseñada para ser más rápida y eficiente en memoria, y especialmente útil con grandes conjuntos de datos. Esto se debe a que, en lugar de niveles, expande las hojas con mayor reducción de pérdida, lo que mejora la precisión. Por último, CatBoost fue desarrollada específicamente para manejar datos categóricos de manera eficiente. Introduce dos nuevos conceptos al modelo: estadísticas objetivo ordenadas y refuerzo ordenado.

Característica	XGBoost	LightGBM	CatBoost
Velocidad	Rápido	Muy rápido	Rápido
Eficiencia en Memoria	Moderada	Alta	Moderada
Manejo de Categorías	Limitado (One-Hot)	Limitado (One-Hot)	Nativo
Tuning de Parámetros	Complejo	Complejo	Relativamente sencillo
Escalabilidad	Alta	Muy alta	Alta

Tabla 2. Características de variantes de algoritmo GBM. Elaboración propia.

En la literatura, hay variados ejemplos de uso de estos algoritmos. Hasan *et al.* (2022) utilizaron LightGBM para predecir ventas de Walmart, aunque comparativamente no fue el mejor modelo dentro de los evaluados. Jain *et al.* (2015) compararon XGBoost con RF y regresión lineal para predecir la venta de una farmacéutica europea, obteniendo el menor error con el primero. Andrade & Cunha (2023) aplicaron XGBoost para predecir la venta de Corporación Favorita, un *retailer* de alimentos ecuatoriano, con una reducción del error de un 26,7% respecto al modelo *Base-Lift*. Más recientemente, Malik

et al. (2024) realizaron predicciones de venta para una farmacéutica con tres algoritmos distintos, siendo XGBoost el que obtuvo los mejores resultados.

3.2.4. Series de tiempo

Una serie de tiempo es una secuencia de datos recolectados y organizados en intervalos de tiempo específicos, como días, semanas, meses, o años. Para poder analizar una serie de tiempo y predecir valores futuros es que se han desarrollado herramientas como ARIMA (*Autoregressive Integrated Moving Average*) y su variante estacional SARIMA (*Seasonal ARIMA*) (Liu *et al.*, 2013).

El modelo ARIMA se denota como ARIMA(p,d,q), donde p es el orden de la parte autoregresiva, d es el número de diferenciaciones y q es el orden de la media móvil (Valipour, 2015). Para migrar al modelo SARIMA, se deben agregar 4 variables más, que se denominan SARIMA(p,d,q)(P,D,Q,s). Las primeras 3 corresponden a su equivalente en ARIMA, y las 4 siguientes representan el orden autorregresivo estacional (P), el número de diferenciaciones estacionales (D), el orden de la media móvil estacional (Q) y el periodo de estacionalidad (s, por ejemplo, 12 para datos mensuales) (Valipour, 2015).

$$(1 - \phi_1 B) (1 - \Phi_1 B^4) (1 - B) (1 - B^4) y_t = (1 + \theta_1 B) (1 + \Theta_1 B^4) e_t.$$

↑ ↑ ↑ ↑
 (Non-seasonal) (Non-seasonal) (Non-seasonal) (Non-seasonal)
 (AR(1)) (difference) (MA(1)) (difference)
 (Seasonal) (Seasonal) (Seasonal) (Seasonal)
 (AR(1)) (MA(1))

Ilustración 9. representación de la ecuación de SARIMA para cuatrimestres (s=4) (Hyndman & Athanasopoulos, 2018)

Ambas variantes han sido exitosamente usados en predecir venta de comercio minorista. Si bien en el estudio de Malik *et al.* (2014) no tuvo tan buenos resultados como los demás modelos, en Eglite & Birzniece (2022) describen su uso en varias instancias, particularmente de alimentos perecibles que son más susceptibles a la estacionalidad. Lindfors (2021) por su parte, comparó ARIMA con varias metodologías como suavizamiento exponencial, árboles de decisión y ANN. Usando data real de Walmart, determinó que el mejor ajuste lo tuvo con ARIMA. Resultados similares tuvieron Arunraj & Ahrens (2015) y Hasan *et al.* (2022) en el mismo rubro. Por otro lado, en Khashei & Bijari (2011) usaron el modelo ARIMA de forma híbrida con redes neuronales para proponer una alternativa de mejores resultados.

3.2.5. Arquitecturas de aprendizaje profundo

Si bien hasta hace poco se pensaba que las redes neuronales no tenían un buen desempeño para predecir datos futuros (Hewamalage *et al.*, 2021), la aparición del *Big Data* ha facilitado las herramientas que estas técnicas necesitan para alcanzar su potencial. Los modelos que usan redes neuronales están particularmente bien facultados para manejar grandes volúmenes de información y obtienen mejores resultados en esos escenarios (Hewamalage *et al.*, 2021). De todas formas, las limitaciones en conocimiento y recursos son una barrera de entrada a este tipo de algoritmos.

Un primer tipo de red que se usa en *forecast* son las Redes Neuronales Profundas (DNN por sus siglas en inglés). Este consiste en capas de entrada y salida, junto a las que se denominan capas ocultas, como se ve en la Ilustración 10. Los nodos de cada capa están totalmente conectados a los nodos de la capa anterior.

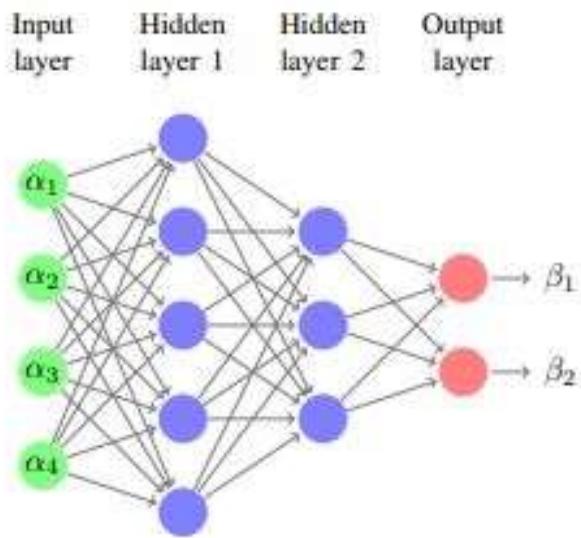


Ilustración 10. Esquema representativo de un DNN (Benidis *et al.*, 2022)

La salida de la última capa oculta puede verse como una representación no lineal de características (también llamada *embedding*) obtenida a partir de las entradas de la red. La capa de salida aprende una correspondencia entre estas características no lineales al objetivo real (Benidis *et al.*, 2022). Si bien esta arquitectura tiene algunas limitaciones, como el número fijo de entradas y salidas, suele utilizarse como la base de otros algoritmos más complejos.

Por otro lado, están las Redes Neuronales Convolucionales (CNN por sus siglas en inglés) que, aunque son más conocidas en *Computer Vision*, también se utilizan en *forecasting* al capturar patrones locales en ventanas de datos temporales. Las CNN son redes neuronales con conexiones locales que emplean capas convolucionales para

aprovechar la estructura inherente en los datos de entrada. Esto se logra aplicando una función de convolución a pequeñas regiones o vecindades de los datos. En este contexto, la convolución implica calcular sumas ponderadas de forma deslizante mediante un filtro o núcleo que se mueve sobre distintas secciones de los datos de entrada (Benidis *et al.*, 2022). En la Ilustración 11 se puede ver un esquema de las CNN.

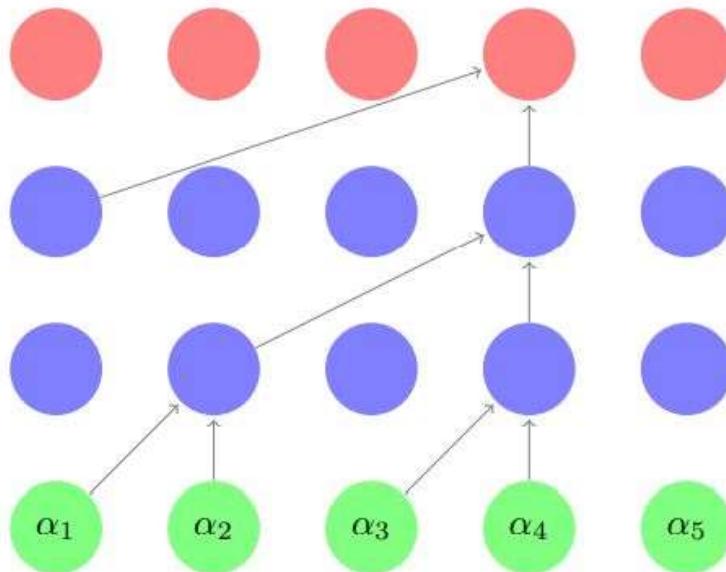


Ilustración 11. Estructura de una CNN formada por una pila de tres capas convolucionales causales. La capa de entrada (verde) es no dilatada y las otras dos son dilatadas (Benidis *et al.*, 2022)

Otro tipo de arquitectura que usa redes neuronales son las Redes Neuronales Recurrentes (RNN por sus siglas en inglés), las cuales son las más utilizadas en problemas de predicción de secuencia (Hewamalage *et al.*, 2021). La idea principal consiste en conectar de forma recurrente las unidades ocultas de las redes neuronales consigo mismas con un retardo de tiempo (Benidis *et al.*, 2022). Las RNN dotan a la red de una memoria dinámica al retroalimentar las unidades ocultas, que aprenden representaciones de características a partir de los datos de entrada. Un aspecto clave es que la misma red se utiliza en todos los pasos temporales, compartiendo los mismos pesos a lo largo de toda la secuencia. Esta idea de compartir pesos es comparable a las CNN, donde un mismo filtro se aplica a diferentes partes de la entrada. Gracias a esta característica, las RNN pueden manejar secuencias de longitud variable durante el entrenamiento y, lo más importante, generalizar a secuencias de longitudes no vistas previamente. En la Ilustración 12 se puede ver un esquema de la arquitectura de una RNN.

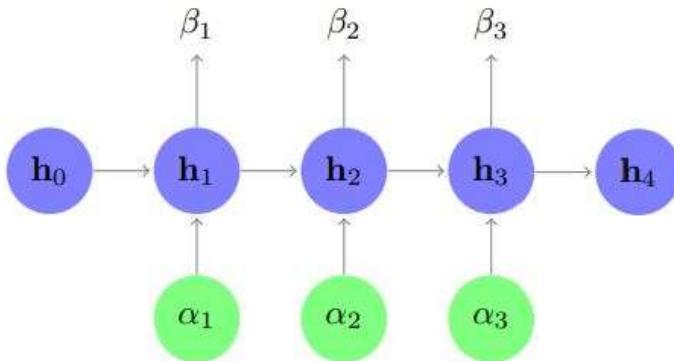


Ilustración 12. Estructura de una RNN expandida, donde cada unidad recibe una entrada externa α_t y la salida de las unidades ocultas del paso temporal anterior h_{t-1} .

Una variante de las RNN utilizada para predecir datos futuros es el modelo LSTM (*Long Short-Term Memory*), propuesta por Schmidhuber & Hochreiter (1997) para abordar el problema de desvanecimiento y explosión de gradiente que puede ocurrir en el proceso de *backpropagation* y el problema que tienen las RNN para recordar información relevante de pasos muy anteriores (Hewamalage *et al.*, 2021). Las LSTM introducen un mecanismo de memoria controlado por puertas que permiten decidir qué información almacenar, actualizar o descartar en cada paso.

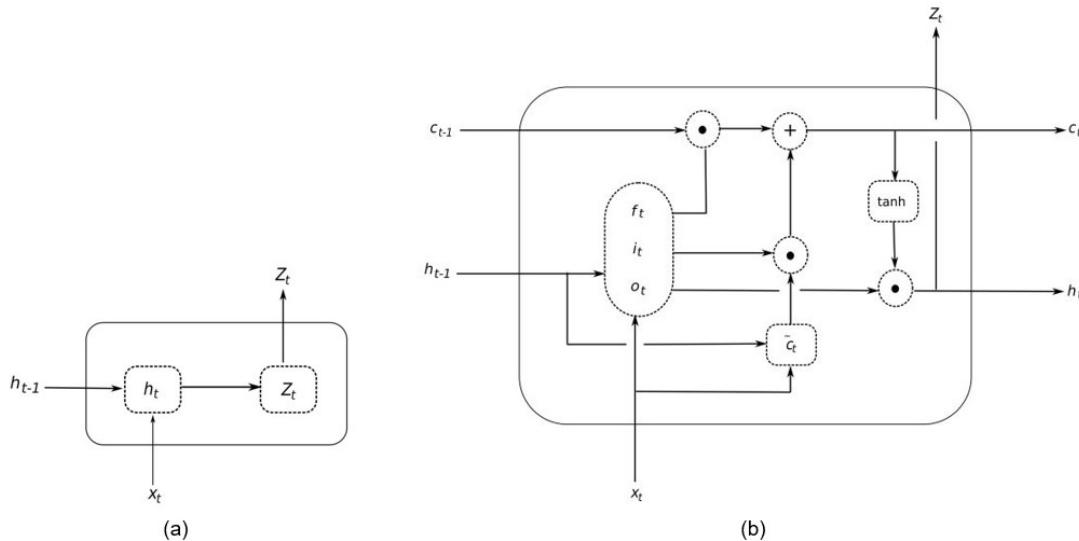


Ilustración 13. Esquema de la unidad de una RNN (a) versus una red LSTM (b) creada a partir de imágenes disponibles en (Hewamalage *et al.*, 2021). En ambos esquemas, x_t representa la entrada de la unidad en el instante t , z_t representa la salida de la unidad y h_t representa el estado oculto de la unidad.

También existe la variante de RNN llamada *Gated Recurrent Unit* (GRU), que consiste en una simplificación de las LSTM que también buscan resolver los problemas de las RNN, pero con una estructura más compacta y eficiente. Esto lo logra integrando las funciones de las puertas de entrada y olvido en una sola (Hewamalage *et al.*, 2021).

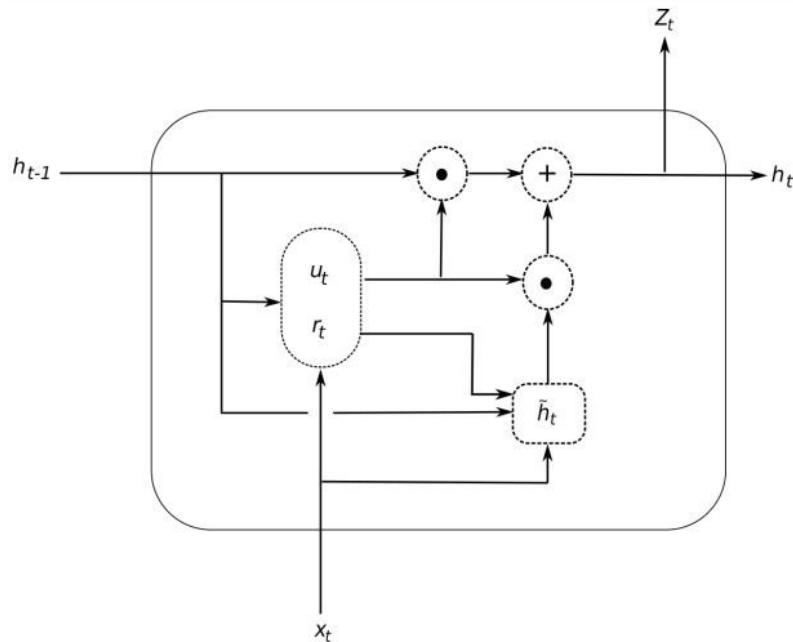


Ilustración 14. Esquema de la unidad de una red GRU (Hewamalage et al., 2021).

Dado su ascenso en popularidad del último tiempo, existen diversos estudios que analizan el potencial de algoritmos de *Deep Learning* comparados a métodos tradicionales. Loureiro et al. (2018) modeló la venta de una comercializadora de vestuario usando diversas técnicas (RF y DNN entre otras). En sus resultados, la DNN tuvo menor error relativo de entre todos los modelos (RMSE y MSE), pero RF tuvo menor error absoluto (MAE, MAPE). Punia et al. (2020) compararon LSTM con RF y regresión lineal múltiple para predecir la venta en un *retailer* de alimentos, obteniendo un menor error con el primero. Hewamalage et al. (2021) evaluó RNN contra el suavizado exponencial y ARIMA y concluyó que las RNN son competitivas frente a estos métodos más tradicionales. A nivel más general, Eglite & Birzniece (2022) concluyen en su reseña bibliográfica que el 82% de los artículos que revisaron vieron una mejora en la predicción al usar técnicas de *Machine Learning* comparado a otras metodologías.

3.3. Optimización de inventario

La optimización de inventarios es un área fundamental en la gestión de la cadena de suministro, ya que busca equilibrar la disponibilidad de productos con los costos asociados al almacenamiento y manejo de estos (Macas *et al.*, 2021). Por lo mismo, se han hecho grandes esfuerzos a lo largo de los años para encontrar distintos modelos y algoritmos que permitan resultados operativos aplicables. En esta sección se revisarán algunas de las propuestas en la literatura.

Tradicionalmente, los modelos de inventario asumían una demanda constante (Bes & Sethi, 1988); sin embargo, en escenarios reales, la demanda suele ser variable e incierta. La planificación de inventarios con demanda dinámica se ha convertido en un enfoque esencial para abordar esta variabilidad. Una adecuada estrategia de inventarios debe incorporar factores cambiantes en el tiempo, ya que los modelos estáticos no son robustos ante esta evidente variabilidad (Cárdenas *et al.*, 2015). Además, es necesario reconocer la gran incertidumbre de la demanda y utilizar métodos de pronóstico adecuados para optimizar los inventarios, considerando costos y niveles de servicio, entre otros aspectos. Es por esto que el *forecast* de venta es una parte crítica del presente trabajo.

La administración de inventarios en cadenas de suministro que involucran múltiples eslabones requiere enfoques específicos para garantizar la coordinación y eficiencia en toda la red (Hausman & Erkip, 1994). Silver, Pyke y Peterson (1998) identifican tres fallas importantes al aplicar un enfoque de inventario de eslabón único: primero, el enfoque de eslabón único asume que los niveles superiores de la cadena de suministro siempre cuentan con suficiente inventario para satisfacer la demanda de los niveles inferiores. Sin embargo, en la práctica, esto no es realista, ya que los eslabones superiores no disponen de un suministro ilimitado. Segundo, este enfoque no considera el impacto que tiene la política de inventario de un eslabón sobre los costos de los demás dentro de la red. Tercero, no contribuye a mitigar el efecto látigo (*bullwhip effect*). Aunque la demanda del producto final tenga poca variabilidad, los pedidos realizados a los niveles superiores de la cadena tienden a ser más grandes y menos frecuentes, lo que genera una mayor fluctuación en la demanda. Como resultado, los eslabones superiores pueden verse obligados a mantener un inventario de seguridad excesivo para compensar la irregularidad en los pedidos. Es por esto que la mayoría de las investigaciones usan el enfoque de múltiples eslabones.

Traditional vs. Multi-Echelon Planning

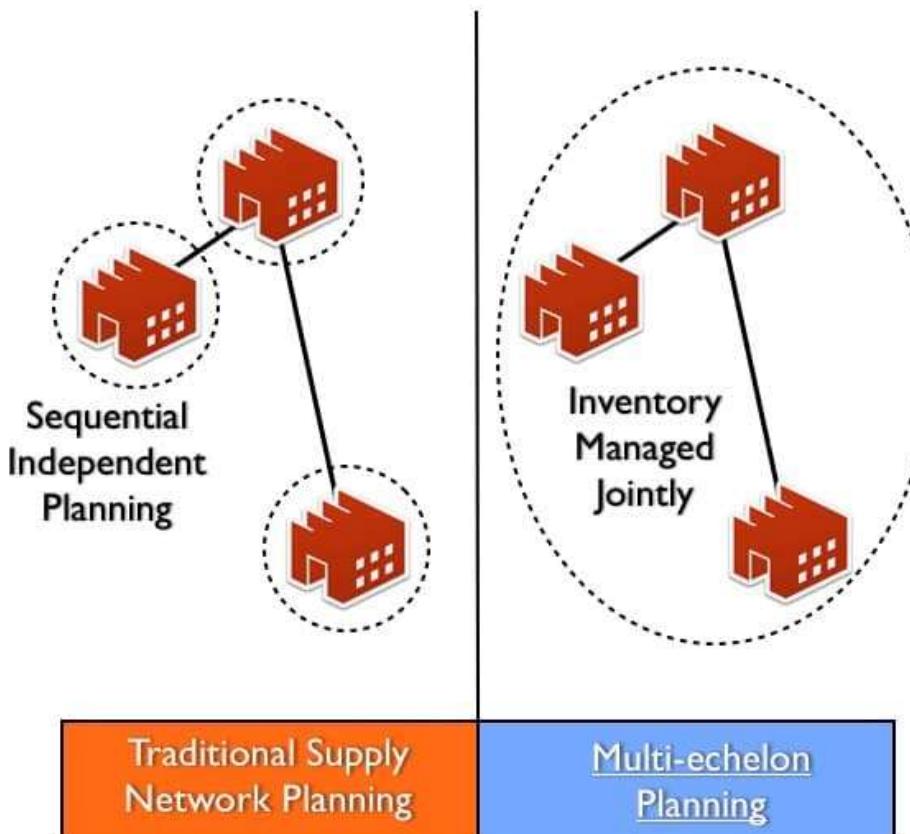


Ilustración 15. Esquema comparativo entre redes de eslabón único y redes de múltiples eslabones (Brightwork Research & Analysis, n.d.)

Una vez definida la estrategia para considerar la red a abastecer, se pueden explorar diversos métodos para optimizar el inventario. El método tradicional más utilizado corresponde a la política de revisión periódica y definición de punto de recompra, o modelo EOQ (Arkaresvimum, 2008). Esta es una estrategia de gestión de inventarios que determina el momento exacto en el que una empresa debe realizar un nuevo pedido de mercancía para evitar desabastecimientos y garantizar la continuidad de las operaciones. El punto de recompra es el nivel mínimo de inventario en el que se debe realizar un pedido de reabastecimiento. Se calcula considerando el tiempo de reposición y la demanda esperada durante ese período, como se muestra en la Ilustración 16. Sin embargo, este método no es recomendable para tiendas minoristas debido a la baja profundidad del inventario por producto.

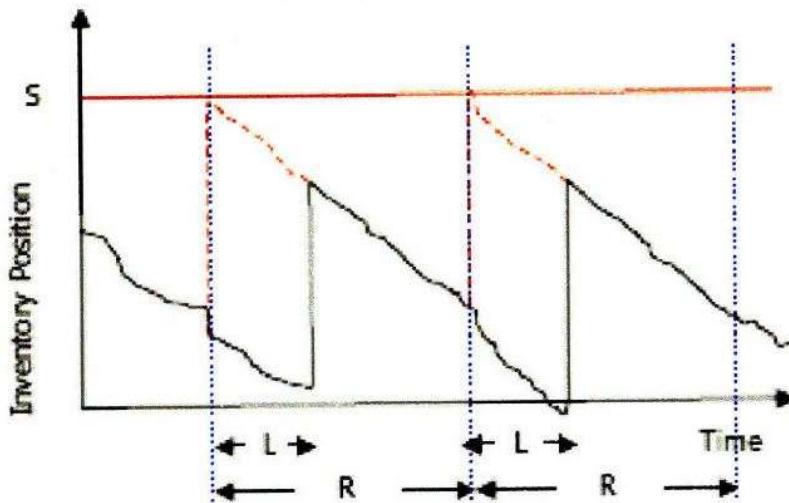


Ilustración 16. Representación gráfica de la estrategia de punto de recompra, donde S representa el punto de recompra, L el tiempo de servicio y R el periodo de revisión (Arkaresvimum, 2008).

Los métodos de programación lineal también han sido usados para optimizar inventario. El software CPLEX, aplicación práctica del algoritmo Simplex, se puede encontrar en la literatura como un motor utilizado para modelar distintos escenarios (Atamtürk *et al.*, 2012; Janssens & Ramaekers, 2011; Zhang & Unnikrishnan, 2016). Este se basa en recorrer los vértices de la región factible, moviéndose a lo largo de sus aristas hasta encontrar la solución óptima. Utiliza una representación en forma de tabla (tabla Simplex) y aplica iteraciones para mejorar la función objetivo, deteniéndose cuando no es posible mejorar más. Es ampliamente utilizado en optimización debido a su eficiencia en problemas de gran escala y su capacidad para manejar restricciones lineales de manera sistemática.

Sin embargo, algunos de estos mismos trabajos compararon CPLEX con algoritmos metaheurísticos y encontraron que los últimos presentan mejor desempeño en sus investigaciones (Janssens & Ramaekers, 2011). En efecto, la metaheurística también está presente en trabajos investigativos del último tiempo, particularmente los algoritmos genéticos (Ma & Fildes, 2017; Radhakrishnan *et al.*, 2009; Singh & Kumar, 2011). Basados en la selección natural de Darwin, los algoritmos genéticos emplean operadores como selección, cruce (*crossover*) y mutación para iterativamente mejorar soluciones a un problema. Inician con una población de soluciones candidatas, evaluadas mediante una función de aptitud (*fitness*), y evolucionan a través de generaciones hasta converger en una solución óptima o cercana. Son especialmente útiles para problemas complejos y de gran escala donde los enfoques tradicionales son ineficientes, como optimización combinatoria, redes neuronales y logística. Los algoritmos metaheurísticos son apropiados en situaciones donde es prácticamente imposible determinar la solución óptima global por la envergadura del problema, por lo que son ideales para optimización de inventario en una cadena de múltiples tiendas.

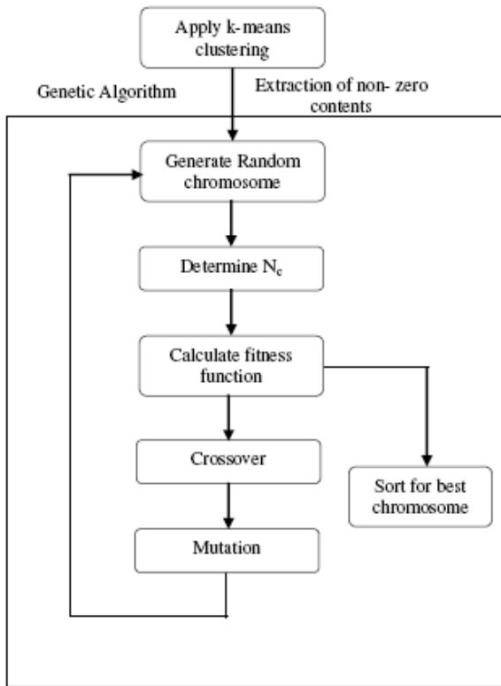


Ilustración 17. Flujo operacional de algoritmo genético según propuesta de Radhakrishnan et al., 2009.

Por último, se han comenzado a utilizar técnicas de *machine learning* para abordar la optimización de inventario (Deng & Liu, 2021; Guo et al, 2014; Thiruverahan & Subramanian, 2015). La mayor cantidad de información que las empresas guardan permite entrenar exitosamente redes neuronales. Los resultados obtenidos son prometedores y vislumbran claras mejoras respecto a algoritmos con mayores supuestos y limitaciones.

4. Desarrollo del proyecto y resultados

4.1. Planteamiento del problema

En el comercio minorista, especialmente en la industria de la moda, la gestión del ciclo de vida de los productos es un desafío clave para garantizar la rentabilidad y optimizar el uso del inventario. En este sector, cada artículo tiene un ciclo comercial relativamente corto, que suele durar aproximadamente nueve meses. Durante los primeros seis meses, los productos se venden a precio *full*, buscando captar la mayor cantidad de ventas a margen pleno. Posteriormente, en los últimos tres meses, los artículos ingresan en un período de liquidación, con descuentos progresivos cuyo objetivo principal es descontinuar el producto y liberar espacio para nuevas colecciones.

Uno de los principales retos en este modelo es que la mayoría de los artículos son importados, lo que significa que los tiempos de fabricación, transporte y distribución pueden ser largos e inciertos. La reposición de productos no siempre es viable dentro de una misma temporada, por lo que la planificación inicial de compras y distribución juega un papel crítico. Si un producto tiene una alta demanda en ciertas ubicaciones y escasea en otras, es posible que no se logren los objetivos de venta antes de que el artículo entre en liquidación, lo que puede afectar la rentabilidad del negocio. Por esta razón, es fundamental garantizar que cada producto esté en el lugar correcto y en el momento indicado, maximizando tanto las ventas como el nivel de servicio al cliente.

Las grandes empresas del sector suelen contar con softwares especializados que les permiten modelar la demanda y optimizar la distribución de inventarios en función de datos históricos, patrones de compra y variables externas como tendencias de moda y cambios estacionales. Estos sistemas avanzados utilizan inteligencia artificial y modelos de predicción para automatizar las decisiones de reposición y minimizar la pérdida de ventas. Sin embargo, muchas pequeñas y medianas empresas no tienen los recursos para acceder a estas herramientas y, en su lugar, deben desarrollar soluciones internas para gestionar su inventario.

En la mayoría de los casos, estas soluciones *in-house* se basan en hojas de cálculo (principalmente en Excel), donde un equipo de analistas revisa manualmente la historia de ventas por tienda, proyecta la demanda futura y determina los niveles de inventario adecuados para cada punto de venta. Esta información luego se ingresa en los sistemas de reposición de los centros de distribución, que se encargan de abastecer las tiendas de manera periódica. No obstante, la manualidad de este proceso y la enorme cantidad de combinaciones posibles entre productos y locales pueden generar errores e ineficiencias. En algunos casos, productos de alta rotación pueden terminar en tiendas con menor afluencia de público, reduciendo su exposición y limitando su potencial de venta a precio completo. Como resultado, la empresa puede perder oportunidades de rentabilidad y verse obligada a realizar descuentos antes de lo planeado, impactando sus márgenes y la eficiencia del negocio.

Dado este contexto, se propone desarrollar un modelo que proyecte la venta por local en una red de tiendas y luego optimice el inventario de cada una. El modelo tomará en cuenta la capacidad de almacenaje de cada local, la venta histórica y la tendencia actual. Esto asegurará que todo el *mix* de productos y todas las tiendas sean consideradas en simultáneo. Para ello, se usarán datos reales proporcionados por la empresa chilena comercializadora de calzado Bayona SPA, dueña de las marcas Guante (calzado masculino) y Gacel (calzado femenino). El modelo de todas formas será aplicable a otros rubros ya que el usuario únicamente deberá modificar la información de entrada y ajustarla a su realidad.

4.2. Desarrollo del proyecto

Para el desarrollo de este proyecto se utilizó el lenguaje de computación Python, uno de los lenguajes más usados en el campo de la ciencia de datos. Es reconocido también por su gran comunidad de desarrolladores y su sintaxis simple y fácil de entender. El entorno de programación usado fue local, en Jupyter Notebook, en un computador con las siguientes características:

- Maquina: Dell XPS 13 9310 2-in-1
- Procesador: 11th Gen Intel(R) Core(TM) i7-1165G7, 2.80GHz
- RAM Instalada: 16,0 GB
- Tipo de sistema: Sistema operativo de 64 bits, procesador basado en x64. Windows 10 Pro, versión 22H2
- Disco Duro: SSD 512 GB

Las siguientes bibliotecas fueron utilizadas para procesar datos, modelar las distintas soluciones y generar las visualizaciones necesarias:

- Numpy (v1.23.5): fundamental para el cálculo numérico en Python. Proporciona soporte para arrays multidimensionales y funciones matemáticas optimizadas, permitiendo realizar operaciones de álgebra lineal, estadísticas y generación de números aleatorios de manera eficiente.
- Pandas (v2.1.4): biblioteca diseñada para la manipulación y análisis de datos en estructuras tabulares, como hojas de cálculo o bases de datos. Permite trabajar con datos de forma eficiente mediante estructuras como DataFrame y Series, facilitando la limpieza, transformación y exploración de datos. Su integración con NumPy y otras bibliotecas hace que sea esencial en el análisis de datos y ciencia de datos.
- Matplotlib (v3.8.0): biblioteca de visualización que permite crear gráficos estáticos, interactivos y animados en Python. Ofrece una gran flexibilidad para personalizar gráficos de líneas, barras, histogramas, dispersión y más.
- Scikit-learn (sklearn) (v1.2.2): biblioteca de *machine learning* que incluye algoritmos para clasificación, regresión, *clustering* y reducción de dimensionalidad. Proporciona herramientas para preprocesamiento de datos, selección de modelos y evaluación del rendimiento. Su facilidad de uso y amplia

documentación la convierten en una de las bibliotecas más populares para implementar modelos de aprendizaje automático.

- TensorFlow (v2.12.0): es una biblioteca de código abierto desarrollada por Google para el aprendizaje profundo y la inteligencia artificial. Facilita la construcción y entrenamiento de redes neuronales mediante su API de alto nivel (Keras) y su arquitectura optimizada para GPU y TPU. Se utiliza ampliamente en visión por computadora, procesamiento de lenguaje natural y otras áreas de IA debido a su escalabilidad y eficiencia.
- Statsmodels (v0.14.0): biblioteca especializada en modelos estadísticos y econometría. Permite realizar regresión lineal, análisis de series temporales (como ARIMA) y pruebas estadísticas avanzadas.
- XGBoost (v2.1.4): biblioteca optimizada para modelos de *boosting* en *machine learning*.

Este trabajo emplea la metodología de minería de datos CRISP-DM (Chapman *et al.*, 2000), descrito en el capítulo 3.1. Ésta permite analizar una gran cantidad de datos y extraer información relevante a partir de ellos. También ofrece una mirada centrada en el negocio en el que enmarca el proyecto, considerando sus objetivos y la puesta en marcha de soluciones encontradas.

CRISP-DM establece que el ciclo de vida de un proyecto lo forman 6 etapas, las cuales se analizaran a continuación respecto a este proyecto.

4.2.1. Comprensión del negocio

En esta primera fase de la metodología se define claramente el problema de negocio, los objetivos estratégicos y los criterios de éxito. Esta información será la base para guiar las siguientes etapas, asegurando que el modelo de pronóstico y optimización de inventario esté alineado con las necesidades del negocio.

Como ya se ha discutido, el problema se da en el contexto de una empresa que vende calzados importados en una red de 56 tiendas a lo largo de todo el país de Chile, Guante (calzado masculino) y Gacel (calzado femenino). Adicionalmente, cada marca vende accesorios y productos asociados al calzado. En estas últimas categorías existen carteras, cinturones, calcetines, billeteras, productos de cuidado de calzado y plantillas, entre otros. Las tiendas se categorizan según su venta y pueden ser de tipo A (mayor venta), B (venta media), C (menor venta), D (outlet para productos descontinuados) y E (tiendas duales que venden ambas marcas).

Se definen dos colecciones al año por marca, una para productos de invierno y otra para productos de verano, con periodos de 3 meses a comienzos de cada temporada para liquidar la colección anterior. Los productos se fabrican principalmente en Brasil, China e India, lo que hace muy difícil realizar reposiciones de productos exitosos durante el avance de la temporada.

La red de locales se abastece al menos una vez a la semana a través de un sistema de reposición automática que se ejecuta en el ERP (*Enterprise Resource Planning*) de la empresa, en el módulo de planificación maestra. Éste se alimenta a través de un archivo plano csv que contiene una matriz con el código de la tienda, el SKU (*Stock Keeping Unit*) correspondiente y la cantidad de stock que debiese tener la tienda. Luego, el sistema compara el inventario real de la tienda con su ideal definido y genera una tarea de *picking* automática hacia el WMS (*Warehouse Management System*) de la empresa. Por lo tanto, el entregable de esta solución debería ser un archivo plano que se pueda cargar en el ERP y que cumpla con la estructura aceptable para éste.

No obstante a lo anterior, cada local tiene una capacidad de bodega (almacenaje) máxima y una capacidad de exhibición máxima que se debe considerar. La primera indica la cantidad de pares de zapatos que puede almacenar la tienda y la segunda, la cantidad de productos que se pueden exhibir al mismo tiempo en la sala de ventas. Puede ocurrir que una sea menor que la otra, por lo que ambas deben considerarse como restricción. El ERP actualmente no considera esta data en la reposición automática y permite cargar un inventario mayor al permitido en la práctica. La solución en cambio debe ser capaz de tomar esta información en cuenta y no superar los límites de las tiendas.



Ilustración 18. Plano de local de ventas, donde se diferencia la sala de ventas de la bodega

Por último, es posible separar los productos en dos categorías según su ciclo de ventas. Los productos atemporales son aquellos que se venden durante todo el año y su demanda no sufre mayores variaciones según el clima. Por el contrario, la demanda de los productos de temporada depende completamente del clima (por ejemplo, sandalias en verano y botas en invierno) y en general su mayor venta se da en períodos muy acotados. Para estos últimos, se hace muy relevante observar la curva de demanda histórica y la tendencia actual de venta. El modelo debe ser capaz de anticiparse al crecimiento explosivo de venta de estos productos y aumentar el stock de las tiendas. Con eso, el nivel de servicio al cliente no se verá afectado por quiebres de producto ante ventas inesperadas. Esto es aún más relevante en períodos de eventos comerciales como Navidad.



Ilustración 19. Diagrama comparativo de calzados atemporales y de temporada, para hombre y mujer.

4.2.2. Comprensión de los datos

En esta fase, se recopilan los datos provenientes de distintas fuentes y se realiza un análisis exploratorio para detectar patrones, tendencias y posibles problemas como datos faltantes o valores atípicos. Para este proyecto, se obtuvieron datos reales de venta de Guante y Gacel entre el 01 de enero del 2023 y el 20 de febrero de 2025. La base de datos contiene las siguientes columnas con información:

- Tipo de local: clasificación del local según parámetros de la empresa
- Nombre del local: nombre identificador del local
- Marca: identificador de la marca involucrada (Guante o Gacel)
- Submarca: identificador de la submarca involucrada. La submarca corresponde a una primera separación tipo de producto para diferenciaciones de marketing. Por ejemplo, en Guante, se separa por submarca las líneas premium de producto (submarcas Guante President y Guante 1928) y la línea Outdoor (submarca Guante Pro) del resto de productos (submarca Guante)
- Familia: identificador de la familia de productos. Puede ser Zapato o Accesorio
- Subfamilia: corresponde al tipo de producto. En Zapato, puede tomar valores como Sandalia, Zapatilla o Bota, entre otros. Para Accesorio, la subfamilia sería Billetera, Cartera, Crema, Escobilla o Mochila, entre otros. En la Ilustración 20 se muestra la totalidad de subfamilias, ordenadas por Familia y por unidades vendidas (descendente).

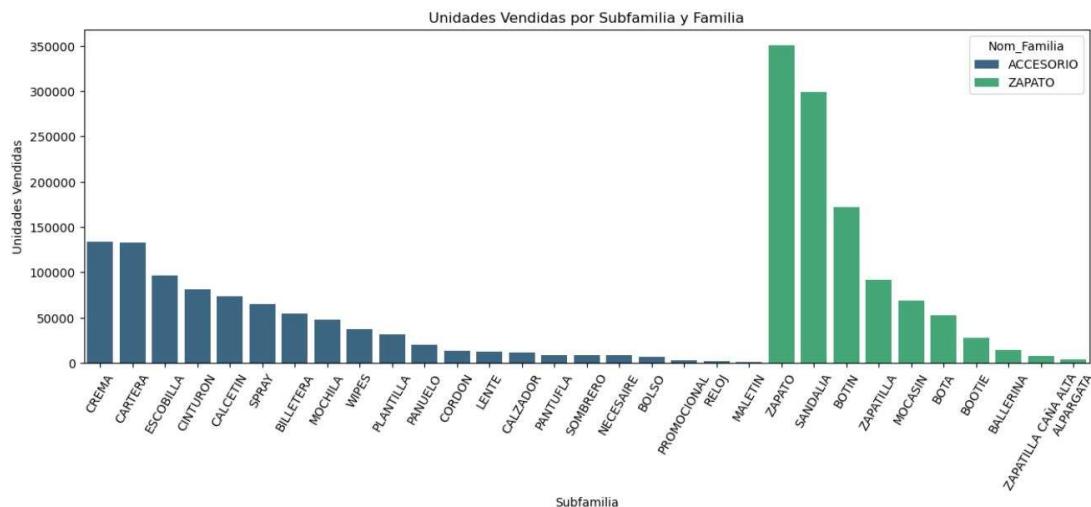


Ilustración 20. Unidades vendidas por Subfamilia y Familia entre el 01/01/23 y el 20/02/25

- Clúster: corresponde a la categorización de producto realizada por la empresa. Esta categoría pretende agrupar productos reemplazables para el cliente. En general, considera la Subfamilia y la ocasión de uso del producto.
- Estado Colección: indica la etapa del ciclo de vida del producto en la que se encuentra. Puede tomar valores de Alta Temporada (venta a precio completo), Liquidación (en periodo de descontinuación), No Colección (producto descontinuado con venta en outlet) o Saldo (agrupa productos con cantidades muy menores de stock en un único código; venta en outlets).
- Fecha: fecha en que se efectuó la venta
- Dia Sem: indica el día de la semana en que se realizó la venta. Toma valores del 1 (lunes) al 7 (domingo)
- Sem del año: semana del año en que se realizó la venta. Considera como 1 la semana en que se ubique el 01 de enero, independientemente de su día de semana.
- Feriado: indica si es que en esa fecha hubo algún festivo nacional
- Evento: indica si en esa fecha hubo algún evento comercial nacional, como el Día de la Madre o Navidad.
- Código producto: codificación interna que recibe cada producto
- Unidades: unidades que fueron vendidas. Al observar esta columna con más detalle, usando el comando Pandas `info()`, vemos que hay valores negativos correspondientes a devoluciones de productos. En la siguiente sección, reemplazaremos esos valores por 0.

```

count      1.754997e+06
mean      1.104340e+00
std       8.794033e-01
min       -5.000000e+01
25%       1.000000e+00
50%       1.000000e+00
75%       1.000000e+00
max       2.230000e+02
Name: Uns, dtype: float64

```

Ilustración 21. Información estadística de la venta diaria en unidades

- Monto neto: monto vendido en pesos chilenos

En la Ilustración 22 se muestra el resultado de ejecutar el comando `head()` al Pandas DataFrame con la información.

	Sub Canal	Cliente	Nom_Marca	Nom_SubMarca	Nom_Familia	Nom_SubFamilia	Cluster	Estado Colección	Fecha	Dia Sem	Sem del año	Feriado	Evento	Cod_Producto	Uns	Neto final
0	TIENDAS A	MALL COSTANERA CENTER GUANTE	GUANTE	GUANTE	ACCESORIO		WIPES	WIPES	Alta Temporada	2023-01-02	1	2	FERIADO	NaN	ch222	10 19098
1	TIENDAS C	MALL PLAZA DE LOS RIOS GUANTE	GUANTE	GUANTE	ACCESORIO		WIPES	WIPES	Alta Temporada	2023-01-02	1	2	FERIADO	NaN	ch222	8 19601
2	TIENDAS A	MALL ALTO LAS CONDES GUANTE	GUANTE	GUANTE	ACCESORIO		CALCETIN	CALCETIN	Alta Temporada	2023-01-02	1	2	FERIADO	NaN	ca899	7 58527
3	TIENDAS A	MALL ALTO LAS CONDES GUANTE	GUANTE	GUANTE	ACCESORIO		CORDON	CORDON	Alta Temporada	2023-01-02	1	2	FERIADO	NaN	ch824	7 17353
4	TIENDAS A	MALL PLAZA OESTE GUANTE	GUANTE	GUANTE	ACCESORIO		CREMA	CREMA	Alta Temporada	2023-01-02	1	2	FERIADO	NaN	ch216	7 40068

Ilustración 22. Muestra base de datos de ventas de Guante y Gacel

La base de datos contiene valores nulos en la columna “Evento”, indicando que no hay evento comercial ese día, como se muestra en la Tabla 3. Adicionalmente, las fechas de feriados irrenunciables, como Año Nuevo, no tienen datos, por lo que se deben agregar de forma manual al DataFrame.

Columna	Valores Nulos
Sub Canal	0
Cliente	0
Nom_Marca	0
Nom_SubMarca	0
Nom_Familia	0
Nom_SubFamilia	0
Cluster	0
Estado Colección	0
Fecha	0
Día Sem	0
Sem del año	0
Feriado	0
Evento	1409592
Cod_Producto	0
Uns	0
Neto final	0

Tabla 3. Valores nulos por columna para base de datos de ventas

4.2.3. Preparación de los datos

En esta etapa se realizar modificaciones en la base de datos inicial para formar el conjunto de datos definitivo, el cual se usará en la etapa de Modelado. Se realizan tareas de limpieza y transformación de datos para garantizar la idoneidad de estos para el análisis posterior. Un extracto con lo más relevante del código usado para llevar a cabo estas tareas se encuentra en el Apéndice I. Adicionalmente, el código completo está disponible en GitHub (<https://github.com/msmuriasp/MasterVIU>).

Según lo observado en la sección anterior, el primer paso será reemplazar los valores de venta negativos por 0 y llenar los valores nulos en la columna “Evento” por un texto que indique que ese día no tiene Evento. Lo primero lo realizamos con la función Pandas `clip()`, indicando que los valores menores serán iguales a 0. En la Ilustración 23 se ve que el mínimo valor de las unidades vendidas ahora es 0.

```

count      1.754997e+06
mean       1.158387e+00
std        7.696575e-01
min        0.000000e+00
25%        1.000000e+00
50%        1.000000e+00
75%        1.000000e+00
max        2.230000e+02
Name: Uns, dtype: float64

```

Ilustración 23. Información estadística de la venta diaria en unidades luego de regularización

Para eliminar los valores NaN, usamos el comando de Pandas `fillna()` e indicamos el nuevo valor, que en este caso será “NOEVENTO”. Esta categoría será la más voluminosa en esta categoría, como se muestra en la Ilustración 24.

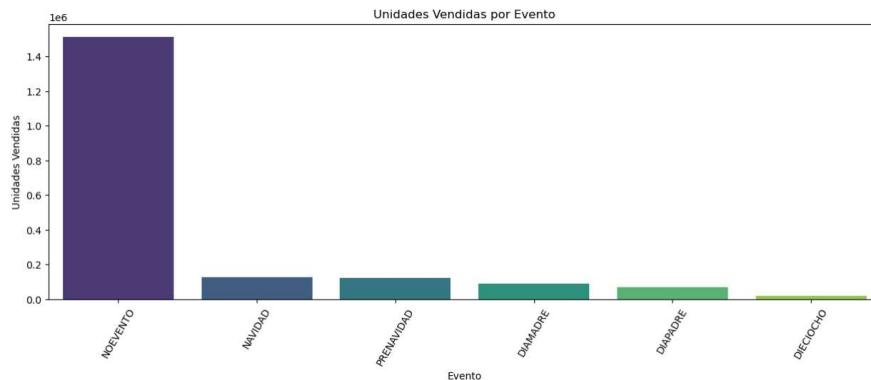


Ilustración 24. Unidades vendidas por evento comercial

En la Tabla 4 se ve que ya no quedan valores nulos en la base de datos.

Columna	Valores Nulos
Sub Canal	0
Cliente	0
Nom_Marca	0
Nom_SubMarca	0
Nom_Familia	0
Nom_SubFamilia	0
Cluster	0
Estado Colección	0
Fecha	0
Dia Sem	0
Sem del año	0
Feriado	0
Evento	0
Cod_Producto	0
Uns	0
Neto final	0

Tabla 4. Valores nulos por columna para base de datos luego de regularización

El siguiente paso será agrupar los datos por categoría, eliminando así el código de producto. Esto se hace para mejorar la precisión en el pronóstico de demanda, ya que la venta por producto puede ser muy errática y puede ser susceptible a la entrada y salida de productos específicos. Además, muchas tendencias y patrones estacionales afectan a la categoría en su conjunto, en lugar de aplicarse únicamente a productos individuales. Esto facilita la detección de aumentos en la demanda asociados a eventos específicos, como cambios de estación o campañas promocionales. Para optimizar la información disponible, se agregó una nueva columna llamada “New_Cluster”, que consiste en la concatenación de la Marca, Subfamilia y Clúster. Esto nos permite eliminar algunas columnas y agrupar la venta diaria en menos categorías. La Ilustración 25 muestra la ejecución del comando Pandas `head()` en la base resultante.

	Fecha	Dia Sem	Sem del año	Feriado	Evento	Sub Canal	Cliente	New_Cluster	Uns	Neto final
0	2023-01-02	1	2	FERIADO	NOEVENTO	TIENDAS A	GACEL TEMUCO	GACEL_BALLERINA_BALLERINA_VESTIR	1	41168
1	2023-01-02	1	2	FERIADO	NOEVENTO	TIENDAS A	GACEL TEMUCO	GACEL_BILLETERA_BILLETERA	1	14277
2	2023-01-02	1	2	FERIADO	NOEVENTO	TIENDAS A	GACEL TEMUCO	GACEL_BOOTIE_BOOTIE_CASUAL_MEDIA	1	42008
3	2023-01-02	1	2	FERIADO	NOEVENTO	TIENDAS A	GACEL TEMUCO	GACEL_CARTERA_CARTERA	3	71370
4	2023-01-02	1	2	FERIADO	NOEVENTO	TIENDAS A	GACEL TEMUCO	GACEL_LENTE_LENTE	1	16798

Ilustración 25. Muestra base de datos de ventas agrupada por categoría (New_Cluster)

Debemos también agregar aquellas fechas que no tienen venta por ser feriado irrenunciable. Para ello, debemos separar todas las combinaciones posibles de tiendas y categoría, asignarles venta 0, completar las columnas con la información de fecha y concatenar esta nueva base de datos con la original. Para no afectar la proyección, los datos luego son ordenados por fecha ascendente.

A continuación, revisamos la información de la venta en unidades. Para ello, agrupamos temporalmente la venta diaria por Marca y Subfamilia. Esto nos permitirá ver si algún valor diario en alguna categoría se escapa de la norma. Hay que considerar que, incluso agrupado, la venta diaria es muy variable, por lo que sólo regularizaremos aquellos valores que se escapen significativamente. De la Ilustración 26 vemos que hay *outliers* en las categorías de Gacel Cartera, Gacel Lente, Gacel Pañuelo, Guante Cinturón, Guante Zapato, Gacel Bootie, Guante Billetera y Gacel Bota. Estos valores fueron revisados individualmente y se eliminaron las líneas de aquellos productos específicos cuya venta estaba desviada. En el caso de Gacel Pañuelo, Gacel Bootie y Gacel Bota, esos registros no correspondían a alguna venta extraordinaria si no que al aumento de ventas por el Día de la Madre, el cual está diferenciado como Evento comercial.

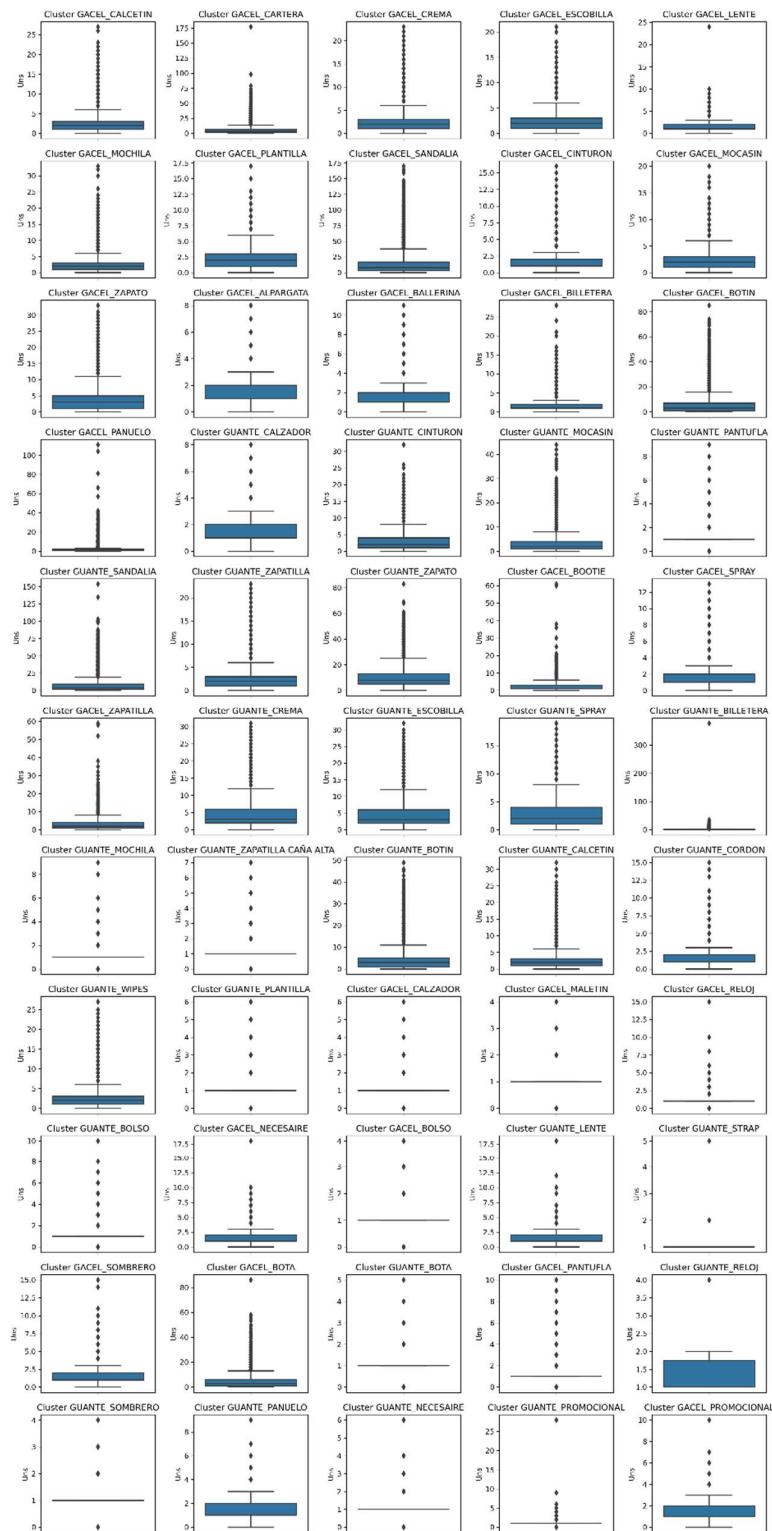


Ilustración 26. Gráficos de caja y bigote para la venta diaria de cada combinación Marca-Subfamilia

Posteriormente, reescribimos las variables categóricas Feriado, Evento y New_Cluster usando *One Hot Encoding*. Esta es una técnica para convertir variables categóricas en variables numéricas binarias, creando una columna para cada categoría con valores 0 o 1, como se muestra en la Ilustración 27. Esto permite que los modelos de *machine learning* procesen datos categóricos sin asumir relaciones ordinales entre las categorías.

Color	Rojo	Amarillo	Azul
Rojo	1	0	0
Rojo	1	0	0
Amarillo	0	1	0
Azul	0	0	1

Ilustración 27. Diagrama ilustrativo del funcionamiento de One Hot Encoding (elaboración propia)

Para realizar este proceso, usamos la clase *OneHotEncoder* de Scikit-learn según el código mostrado en el Apéndice I. Dado el gran número de categorías, la base resultante tiene 151 columnas.

Otras modificaciones que realizamos fue agregar columna con información del precio unitario, resultante de la división entre el monto vendido y las unidades, eliminar la columna con el monto vendido y usar la columna Fecha como índice del DataFrame.

4.2.4. Modelado

La cuarta etapa de CRISP-DM consiste en seleccionar y aplicar técnicas de modelado adecuadas para resolver el problema de negocio. Se define qué algoritmos usar, se ajustan sus hiperparámetros y se entrena el modelo con los datos preparados en la etapa de Preparación de Datos. Este proceso es iterativo ya que se pueden probar distintas alternativas si es que no se alcanzan los resultados esperados. Las secciones más relevantes de los códigos resultantes de esta etapa y descritos a continuación se encuentran en los Apéndices II a V. Adicionalmente, el código completo está disponible en GitHub (<https://github.com/msmuriasp/MasterVIU>).

4.2.4.1 Forecast de venta

Antes de realizar el *forecast* de venta, debemos separar nuestros datos en grupos de entrenamiento, validación y prueba. La estrategia utilizada fue dejar los datos del 2025 (aproximadamente un 7% de los datos) para prueba y, de los datos restantes, usar 20% para validar y 80% para entrenar. Los datos fueron separados sin romper la secuencia temporal. Este proceso se realiza para asegurar que los modelos son capaces de predecir venta para fechas y datos a los que no han estado expuestos.

Para predecir la venta futura, probamos tres metodologías distintas: ARIMA y XGBoost (descritas en la sección 3.2).

1. ARIMA

Para implementar un modelo ARIMA (*Autoregressive Integrated Moving Average*), primero debemos asegurar que la serie temporal esté estacionaria, es decir, que sus propiedades estadísticas no cambien con el tiempo. Para esto, se utiliza el test de Dickey-Fuller aumentado (ADF), el cual evalúa la presencia de una raíz unitaria en la serie temporal. El resultado de este test nos indica si la serie es estacionaria o si es necesario realizar transformaciones adicionales, como la diferenciación. Si el p-valor del test es mayor a 0.05, se concluye que la serie no es estacionaria, por lo que se aplica una diferenciación de primer orden. Para aplicar el test, ejecutamos el comando `adfuller()` de la biblioteca Statsmodels ingresando los datos de la columna 'Uns'.

El siguiente paso es determinar el mejor modelo ARIMA. ARIMA se configura con tres variables: p , d y q (que representan el número de términos autorregresivos, el número de diferenciaciones y el número de términos de media móvil, respectivamente). La función `auto_arima()` de la librería `pmdarima`. Este modelo realiza una búsqueda automática de los mejores parámetros para ARIMA, considerando distintas combinaciones y optimizando según un criterio de selección como el AIC (Criterio de Información de Akaike). El uso de `auto_arima` ayuda a reducir el tiempo y esfuerzo necesarios para ajustar el modelo, ya que evalúa diferentes combinaciones de parámetros y selecciona la mejor opción de manera eficiente.

Con los parámetros seleccionados, se ajusta un modelo SARIMAX (Seasonal ARIMAX), el cual es una extensión del ARIMA que incluye la capacidad de manejar regresores exógenos. Los regresores exógenos son variables externas que no dependen de la serie temporal misma, pero que influyen en el comportamiento de la variable que estamos tratando de predecir, como promociones, precios, días de la semana, entre otros. La ventaja de utilizar un modelo SARIMAX en lugar de un modelo ARIMA simple radica en la capacidad de incorporar estos factores adicionales, lo que mejora la precisión del modelo al permitirle aprender las interacciones entre la serie temporal y las variables externas.

Una vez ajustado el modelo, el siguiente paso es generar predicciones a futuro. El modelo puede ser utilizado para prever la variable y (unidades, en este caso) para los próximos 30 días, especificado en el parámetro `n_periods`. Para hacer esto, es necesario disponer de los valores de los regresores para los próximos días, ya que el modelo SARIMAX depende no solo de la serie temporal histórica sino también de las características exógenas. El pronóstico se realiza utilizando la función `get_forecast()`, que permite obtener tanto las predicciones medias como los intervalos de confianza para las futuras observaciones. Estos intervalos son cruciales para entender la incertidumbre asociada con las predicciones, ya que proporcionan un rango de valores posibles para las futuras ventas. Es posible ver un extracto del código usado para entrenar ARIMA en el Apéndice II.

2. XGBoost

Para implementar un modelo de XGBoost, primero debemos adaptar nuestros datos de entrenamiento, validación y prueba a una estructura soportada por él. Para esto, creamos objetos de tipo DMatrix, que es una estructura de datos optimizada para XGBoost. Estos objetos son esenciales para manejar grandes volúmenes de datos de manera eficiente y acelerar el proceso de entrenamiento. El comando `DMatrix()` de la biblioteca `xgboost` permite realizar esta conversión.

El siguiente paso es definir los parámetros del modelo. XGBoost permite una gran personalización, y los parámetros definidos en el código son claves para configurar el modelo según las necesidades del problema específico:

- 'objective': con este parámetro se debe especificar el problema a resolver y la función de pérdida a minimizar. En este caso, usamos 'reg:squarederror' ya que se trata de un problema de regresión y la función de pérdida es el error cuadrático medio (MSE por sus siglas en inglés), que es común en problemas de regresión.

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (1)$$

- 'eval_metric': Se utiliza para definir la métrica de evaluación que se empleará durante el entrenamiento. En este caso, se ha elegido la raíz del error cuadrático medio (RMSE por sus siglas en inglés), que es una medida comúnmente utilizada para evaluar el rendimiento de modelos de regresión.

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{n}} \quad (2)$$

- 'learning_rate': Este parámetro controla la velocidad con la que el modelo se ajusta a los datos durante el entrenamiento. Un valor más bajo significa que el modelo aprenderá más lentamente, pero puede evitar sobreajustes y mejorar la generalización del modelo. En este caso, se ha seleccionado un valor relativamente bajo (0.05) para garantizar que el modelo realice ajustes más graduales.
- 'max_depth': Este parámetro controla la profundidad máxima de los árboles que se construirán durante el entrenamiento. Un valor más alto permite una mayor complejidad y un mejor ajuste a los datos, pero también aumenta el riesgo de sobreajuste. En este caso, se ha limitado la profundidad a 4 para evitar que el modelo aprenda patrones demasiado complejos que no generalicen bien.
- 'subsample': Este parámetro indica el porcentaje de muestras de entrenamiento que se utilizarán en cada iteración de entrenamiento del árbol. Un valor de 0.8 significa que XGBoost seleccionará aleatoriamente el 80% de las muestras de entrenamiento en cada iteración. Este tipo de regularización ayuda a reducir el sobreajuste.
- 'colsample_bytree': Similar al parámetro anterior, pero en lugar de controlar la cantidad de muestras, este parámetro controla el porcentaje de características

(columnas) que se tomarán aleatoriamente para construir cada árbol. Este valor también está fijado en 0.8, lo que reduce el riesgo de sobreajuste al evitar que el modelo se ajuste demasiado a las características específicas de las muestras.

Con los parámetros definidos, el siguiente paso es entrenar el modelo utilizando la función *train()* de la biblioteca *xgboost*. Esta función se alimenta de los parámetros definidos en el paso previo, los datos de entrenamiento en formato DMatrix, los datos de validación en formato DMatrix y el número máximo de iteraciones (*num_boost_round*). Adicionalmente, se pueden agregar parámetros que hagan más eficiente la simulación, como *early_stopping_rounds*, que detiene la ejecución si no hay mejora en una cierta cantidad de pasos. Los elementos descritos están contenidos en el extracto de código ubicado en el Apéndice III.

4.2.4.2 Optimización de inventario

Para poder optimizar el inventario de la red de tiendas, debemos primero definir la función objetivo y las restricciones del problema. En este caso, deseamos maximizar la venta, es decir, contar con los productos que predecimos vamos a vender. Además, debemos minimizar los costos de transporte a cada tienda, ya que cada viaje tiene un valor asociado. Los costos de arriendo de cada tienda no se tomarán en cuenta ya que son independientes al inventario o la cantidad de visitas.

Adicionalmente, impondremos que cada local debe visitarse al menos una vez a la semana, por política de la empresa, y que el stock dado debe ser menor que la capacidad de bodegaje de la tienda. También deberemos calcular el inventario diariamente al restar la venta e incorporar cualquier reposición que el modelo indique. Queda entonces expresado el problema según la ecuación 3:

$$\min \sum_{t=1}^T |\hat{X}_t - S_t| + \sum_{t=1}^T c \cdot R_t \quad (3)$$

$$\text{sujeto a } \sum_{t=1}^T R_t \geq \text{int}\left(\frac{T}{7}\right) \cdot \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad (3.1)$$

$$S_t \leq L \forall t = 1, 2, \dots, T \quad (3.2)$$

$$S_t = S_{t-1} + \hat{S}_t \cdot R_t - \hat{X}_t \quad (3.3)$$

donde $T = \text{periododepredicción(días)}$

$n = \text{número de locales}$

$\hat{X} = \text{forecast de venta, matriz de dimensión } n \times T$

$S = \text{inventario, matriz de dimensión } n \times T$

$\hat{S} = \text{inventario reposición, vector de dimensión } n$

c = costo transporte, vector de dimensión n

R = matriz binaria para indicar reposición, dimensión $n \times T$

L = capacidad de bodega, vector de dimensión n

El problema de optimización será abordado de dos formas: en primer lugar, con un modelo EOQ y un algoritmo genético. Ambos fueron abordados en el capítulo 3.3. También se exploró las opciones de programación lineal y CPLEX, pero se descartaron ya que los paquetes disponibles no entregan resultados si es que no encuentran una solución óptima (desarrollo no presentado en este trabajo). Por este motivo se eligió usar heurística y algoritmos genéticos.

1. EOQ

El modelo EOQ (*Economic Order Quantity*) se un modelo clásico de gestión de inventario que busca determinar dos variables claves: el punto de pedido (s) y la cantidad de pedido (Q). El punto de pedido corresponde al nivel de inventario en el que se debe emitir un nuevo pedido. Este nuevo pedido será por un total de Q unidades.

Para calcular Q y s , usamos las ecuaciones 4 y 5, adaptadas de Hopp (2011):

$$Q = \bar{X} \cdot h \quad (4),$$

donde \bar{X} = media de venta diaria

h = horizonte de venta

$$s = \mu_{LD} + k \cdot \sigma_{LD} \quad (5),$$

donde $\mu_{LD} = \frac{\bar{X} \cdot LD}{T}$

$$\sigma_{LD} = \sigma_X \sqrt{\frac{LD}{T}}$$

k = parámetro asociado al nivel de servicio

σ_X = desviación típica de venta diaria

LD = días para surtir cada pedido

T = periodo total de análisis

Para esta resolución, se asumirá un nivel de servicio deseado del 95% y un periodo de 3 días para completar cada pedido.

Una vez obtenidos los valores de s y Q para cada local/categoría, simularemos la cantidad de visitas requeridas por local y el inventario diario. Para ello, asumiremos un inventario inicial igual al 80% de su capacidad máxima usando la distribución por categoría de datos reales de stock. El motivo de esto es porque el stock diario supera la capacidad máxima definida en algunos locales, lo que dificulta el cálculo del horizonte a considerar. Además, asumiremos que cada local se visita una vez a la semana, siguiendo la política actual de la empresa.

Antes de poder calcular s y Q debemos definir el horizonte de ventas que queremos cubrir. Buscaremos el valor que minimice el costo del modelo de abastecimiento en tres aspectos: el costo del transporte, la venta perdida y una penalización por exceder la capacidad máxima de cada local. El primero viene dado por datos de entrada y la cantidad de reposiciones realizadas, mientras que, para calcular el segundo, debemos tomar los días en que el inventario simulado por local/categoría es negativo y multiplicarlo por el precio promedio unitario de ese día. Por último, la penalización por sobrestock será la suma de locales diarios que superen su capacidad multiplicado por un millón. Este factor se determinó de tal forma que la penalización tuviera un orden de magnitud similar a los otros dos componentes del costo total. Las funciones definidas para calcular s y Q se muestran en el Apéndice IV.

2. Algoritmo genético

Los algoritmos genéticos son un tipo de algoritmo metaheurístico de optimización inspirado en el proceso de evolución biológica que ocurre naturalmente (Katoch *et al.*, 2021). En ellos, las posibles soluciones se representan como cromosomas y se van cruzando, mutando y seleccionando hasta obtener la versión “mejor adaptada”. La simulación transcurre en un número definido de generaciones y, en cada una de ellas, una cantidad (N) definida de cromosomas (llamada población) interactuarán entre ellos para producir los genes de la siguiente.

El algoritmo diseñado para este problema contará con las siguientes funciones anidadas, cuyo código se muestra en el Apéndice V:

- generar_poblacion(N): esta función recibe como input la cantidad de cromosomas que componen cada generación y entrega una población con N cromosomas
 - crear_solucion(): función auxiliar de generar_poblacion; crea de forma aleatoria un cromosoma. Cada cromosoma se compone de dos diccionarios: 'visitas' y 'stock_obj'. 'visitas' contiene datos binarios (0 o 1) indicando si se visitará una tienda en un determinado día. En 'stock_obj' estará el valor del inventario deseado después de una reposición. Es decir, en cada visita, se enviará al local la diferencia entre el inventario real y el valor de stock_obj para cada categoría/local.
- Cruzar(población, mutación): esta función es la encargada de crear una población ampliada que incluya los cromosomas “hijos” de cada cromosoma. Considera además un parámetro ‘mutacion’ que indica la probabilidad de que un cromosoma mute (para este caso usaremos 20%).
 - Descendencia([padre1, padre2], mutacion): función auxiliar de Cruzar() que genera cromosomas hijos a partir de dos cromosomas padres. La descendencia se genera al intercambiar 'visitas' y 'stock_obj' de cada cromosoma de origen.
 - Mutar(solucion, mutacion): función auxiliar de Descendencia() que recibe un cromosoma y determina si éste sufrirá una mutación. En caso

afirmativo, elige dos días aleatorios de ‘visitas’ e intercambia sus valores con el día siguiente.

- Factibilizar(solucion): función auxiliar de Descendencia() y Mutar() que modifica una solución para que cumpla con ciertos criterios mínimos del problema. En este caso, la matriz binaria de visitas a locales debe sólo tener valores 0 o 1 y la matriz stock:obj debe tener valores mayores o iguales a cero.
- Seleccionar(poblacion, N, elitismo): función de selección de la población. Recibe como parámetro una población, la cantidad total de elementos de ella y un parámetro de ‘elitismo’. Este parámetro (que tendrá valor del 10% en este trabajo) indica qué porcentaje de la población será eliminado según su bajo *fitness*. Para ello, se calcula la función objetivo para cada cromosoma y se ordena de forma descendiente según este valor y se eliminan aquellos equivalentes al ‘elitismo’% final. Para mantener el tamaño de la población, luego se agregan elementos aleatorios de los cromosomas descartados.
- EvaluarPoblacion(poblacion): función que recibe una población y evalúa cada elemento según una función objetivo. Devuelve el cromosoma cuya función objetivo sea menor que los demás y el valor de dicho resultado.
 - funcionObj(solucion): función auxiliar de Seleccionar() y de EvaluarPoblacion(). Ella consta de tres partes: el cálculo del costo de cada visita, la venta perdida si es que hubiera menos stock que la demanda del día y una penalización por superar la capacidad del local.
 - calcular_venta_perdida(solucion): función auxiliar de funcionObj() que calcula la venta total perdida mediante una simulación de inventario. Recibe un cromosoma con la información de las visitas a cada tienda y la cantidad a reponer.

El algoritmo genético se parametriza para simular 50 generaciones y entregas la solución con el mejor costo al final de este periodo. Luego, con la información de las visitas a tiendas se puede calcular el coste total de transporte para poder compararlo con la solución propuesta por EOQ. Adicionalmente, podemos calcular el inventario diario para verificar si supera la capacidad máxima de alguna tienda.

4.3. Resultados

4.3.1. Forecast de venta

Para evaluar el desempeño de cada modelo de *forecast*, usaremos las métricas R2, RMSE y MAPE. La segunda ya fue definida en la ecuación (2), por lo que falta por definir las demás.

El coeficiente de determinación, o R2, corresponde a un estadístico que mide la bondad de ajuste de un modelo. Se usa generalmente en regresiones y se calcula usando la ecuación (6). Por otro lado, MAPE corresponde a las siglas en inglés del error absoluto

medio. Esta métrica mide el error promedio en términos absolutos entre datos reales y valores predichos. Se calcula mediante la ecuación (7).

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \quad (6)$$

$$MAPE = 100 \cdot \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{(y_i + \varepsilon)} \right| \quad (7)$$

donde y_i son los valores reales, \hat{y}_i son los valores predichos, \bar{y} es el promedio de los valores reales y ε es una constante igual a 1e-6 para evitar divisiones nulas.

1. ARIMA

El modelo ARIMA determinó que los hiperparámetros ($p=3$, $d=1$, $Q=5$) entregaban el mejor ajuste, como muestra la Ilustración 28.

```
Best model: ARIMA(3,1,5)(0,0,0)[0]
Total fit time: 1701.490 seconds
SARIMAX Results
=====
Dep. Variable:                      y
No. Observations:                  64481
Model:                            SARIMAX(3, 1, 5)
Log Likelihood:                 -334861.231
Date:                Sat, 15 Mar 2025
Time:                    03:38:47
AIC:                         669740.463
BIC:                         669822.130
HQIC:                        669765.746
HQIC:                        669765.746
Sample:                           0
Covariance Type:                  opg
=====
coef      std err          z      P>|z|      [0.025      0.975]
-----
ar.L1      0.2669     0.007    39.843      0.000      0.254      0.280
ar.L2      0.3844     0.006    67.834      0.000      0.373      0.396
ar.L3     -0.8533     0.006   -149.011      0.000     -0.865     -0.842
ma.L1     -1.1171     0.007   -156.454      0.000     -1.131     -1.103
ma.L2     -0.2621     0.007   -37.514      0.000     -0.276     -0.248
ma.L3      1.1506     0.009   135.052      0.000      1.134      1.167
ma.L4     -0.6858     0.006   -118.761      0.000     -0.697     -0.674
ma.L5     -0.0629     0.003   -21.360      0.000     -0.069     -0.057
sigma2    1949.3327    2.199   886.464      0.000   1945.023   1953.643
=====
Ljung-Box (L1) (Q):            0.39
Prob(Q):                      0.53
Heteroskedasticity (H):        1.52
Prob(H) (two-sided):           0.00
Jarque-Bera (JB):             10149659.61
Prob(JB):                      0.00
Skew:                           4.93
Kurtosis:                      63.67
=====
```

Ilustración 28. Informe entregado por ARIMA tras su ajuste de hiperparámetros

En la Ilustración 29 se muestra un gráfico comparativo entre los últimos 30 días de venta reales y los datos predichos por ARIMA. Si bien la desviación de los resultados es grande (sombra gris), el modelo es capaz de aprender la periodicidad de la venta y replicar su tendencia.

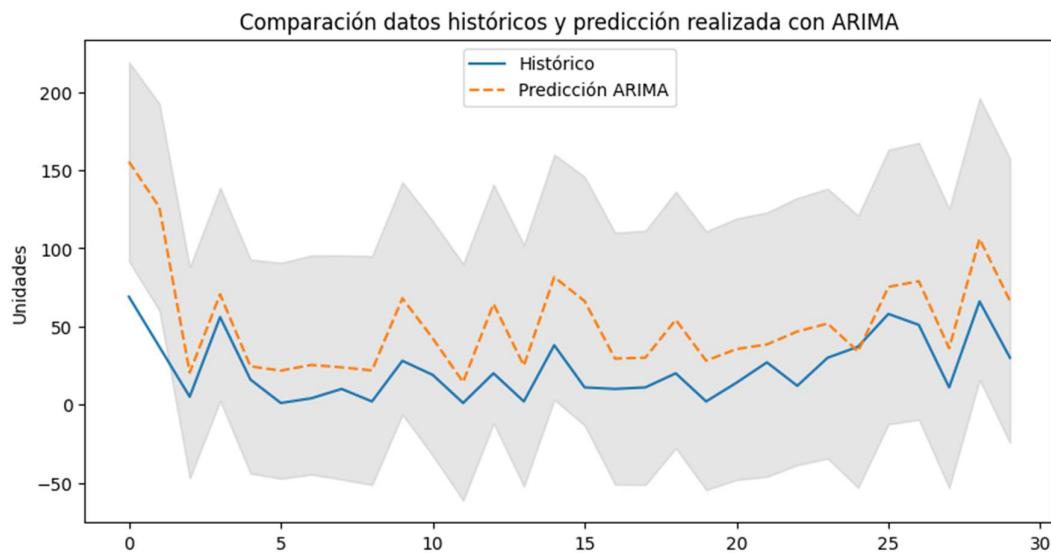


Ilustración 29. Gráfica comparativa de datos de prueba y datos simulados con ARIMA

Con los valores predichos, calculamos las métricas estadísticas que definimos al comienzo de la sección, las cuales están contenidas en la Tabla 5.

MÉTRICA	VALOR ARIMA
R2	0.2997
RMSE	33.9807
MAPE	12.848.653,3682

Tabla 5. Resumen de métricas estadísticas de predicción de venta realizada con ARIMA

Llama la atención el alto valor de MAPE, por lo que exploramos la situación. En la Ilustración 30 se puede ver un DataFrame con los registros con mayor MAPE y se observa que son instancias en que el valor real de venta es 0. Al haber aplicado la constante ϵ en la fórmula, fue posible obtener valores reales muy altos en vez de un error o “infinito”. Esta es una complejidad que tiene MAPE como métrica, ya que para productos de baja rotación puede presentar estas dificultades.

	y0_test	y0_pred	mape
91	0	66.709572	6.670957e+09
2149	0	41.651725	4.165172e+09
135	0	33.101476	3.310148e+09
697	0	29.957704	2.995770e+09
1795	0	29.375054	2.937505e+09
3021	0	28.381902	2.838190e+09
175	0	24.021507	2.402151e+09
2770	0	23.620763	2.362076e+09
1848	0	19.010619	1.981062e+09
146	0	18.917012	1.891701e+09
1191	0	18.614492	1.861449e+09
699	0	16.750758	1.675076e+09
3921	0	16.658626	1.665863e+09
1061	0	16.507484	1.650748e+09
886	0	15.692148	1.569215e+09
1407	0	14.799732	1.479973e+09
189	0	14.625283	1.462528e+09
493	0	14.238793	1.423879e+09
99	0	14.102260	1.410226e+09
3251	0	13.406515	1.340651e+09

Ilustración 30. Listado de registros reales y predichos con ARIMA con mayor MAPE

Para hacer una comparación más amplia entre los datos predichos con ARIMA y XGBoost, agregaremos dos métricas: MASE (*Mean Absolute Scaled Error*), como sugiere Hyndman (2015), y WAPE (*Weighted Absolute Percentage Error*), el cual es similar a MASE, pero da más peso a los errores relativos de las observaciones con valores más grandes, como se muestra en las ecuaciones 8 y 9.

$$MASE = \frac{\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|}{\frac{1}{n-m} \sum_{i=1}^n |y_i - y_{i-m}|} \quad (8),$$

donde $m = \text{periodos de estacionalidad}$

$$WAPE = 100 \cdot \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{\sum_{i=1}^n |y_i|} \quad (9)$$

La tabla actualizada de métricas de ARIMA se muestra en la

MÉTRICA	VALOR ARIMA
R2	0.2997
RMSE	33.9807
MAPE	12.848.653,3682
MASE	0.9183
WAPE	92,2636%

Tabla 6. Resumen actualizado de métricas estadísticas de predicción de venta realizada con ARIMA

2. XGBoost

Los resultados de XGBoost están representados en la Ilustración 31. Al igual que en el caso anterior, el modelo fue capaz de aprender las variaciones entre días.

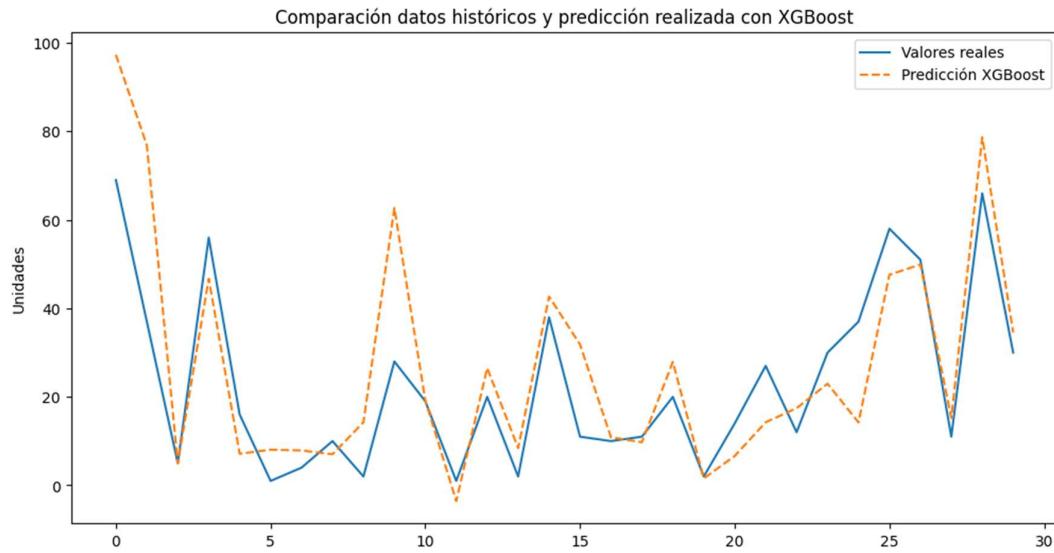


Ilustración 31. Gráfica comparativa de datos de prueba y datos simulados con XGBoost

Con los valores predichos, calculamos las métricas estadísticas que definimos al comienzo de la sección, las cuales están contenidas en la Tabla 7.

MÉTRICA	VALOR ARIMA	VALOR XGBOOST
R2	0.2997	0,7919
RMSE	33.9807	18,5244
MAPE	12.848.653,3682	4.284.138,9773
MASE	0.9183	0,0039
WAPE	92,2636%	40,7277%

Tabla 7. Comparación de métricas estadísticas de predicción de venta realizada con ARIMA y XGBoost

Comparando estas cifras podemos concluir que XGBoost predijo de mejor forma la venta que ARIMA, aunque hay indicios de que podría estar sobreajustado. En efecto, un valor demasiado bajo de MASE puede deberse a sobreajuste y sería necesario en futuras aplicaciones comprobar si esto es así.

4.3.2. Optimización de inventario

1. MODELO EOQ

El primer paso para generar el modelo EOQ fue elegir un horizonte de venta. Para ello, comparamos el costo total del modelo de abastecimiento para valores entre 1 y 30 días. El horizonte óptimo encontrado fue de 25 días, con un costo total de CLP 1.001,82 millones. De éstos, un 41,8% corresponde a costos de transporte y un 55,1%, a venta perdida, como se muestra en la Tabla 8.

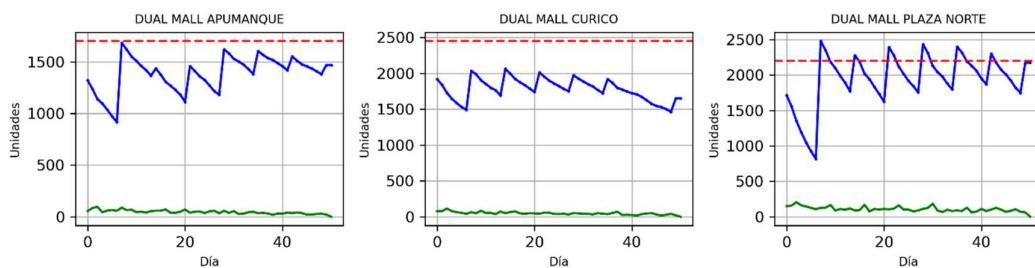


Ilustración 32. Simulación de inventario diario (línea azul) por local usando la metodología EOQ con horizonte de 25 días, junto a la capacidad máxima por local (rojo) y la venta diaria (verde).

En el Apéndice VI se muestra la evolución diaria de inventario de cada tienda (línea azul), junto a su límite de capacidad (línea roja) y su venta diaria (línea verde). Se puede apreciar la clásica forma de sierra del modelo EOQ y que en general no se supera el límite de capacidad de las tiendas. En la Ilustración 32 se muestra un caso particular de esta simulación.

1. ALGORITMO GENÉTICO

Luego de ejecutar 50 generaciones de 100 cromosomas, el algoritmo genético encontró una solución cuyo costo total es de CLP 369,37 millones, como se muestra en la Tabla 8. Los tres factores del costo total tuvieron valores positivos en la solución encontrada, siendo el costo de transporte un 26,4% del costo, la venta perdida un 24,9% y la penalización por sobrestock un 48,7%. El costo encontrado es un 36.86% del costo total del modelo EOQ, lo cual es un resultado muy positivo.

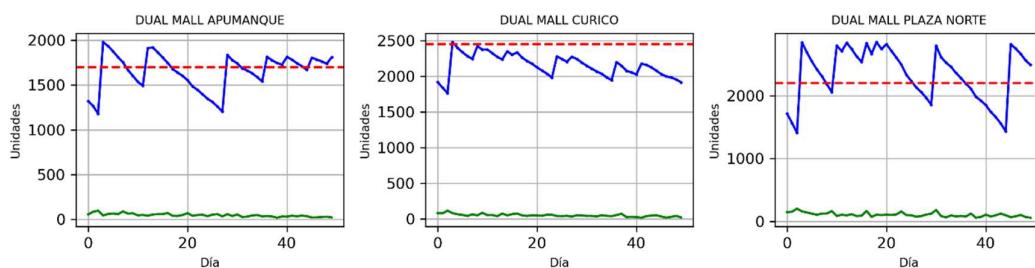


Ilustración 33. Simulación de inventario diario por local usando algoritmo de optimización genética (línea azul), junto a la capacidad máxima por local (rojo) y la venta diaria (verde).

En el Apéndice VII se muestra la evolución diaria de inventario de cada tienda (línea azul), junto a su límite de capacidad (línea roja) y su venta diaria (línea verde), mientras que en la

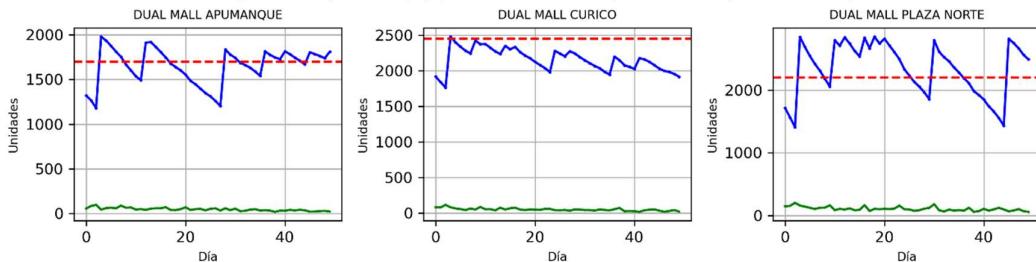


Ilustración 33 se muestra un caso particular. Vemos que al menos 9 de las 59 tiendas superan su capacidad máxima en algún día de simulación, lo que no es recomendable para el flujo operativo de las tiendas. Sin embargo, la venta perdida fue el menor de los tres componentes del coste total, lo que podría influenciar a los responsables de manejar el inventario de tiendas a hacer concesiones.

COSTO	EOQ	ALGORITMO GENÉTICO
TRANSPORTE	418.900.000,00	97.500.000,00
VENTA PERDIDA	551.917.581,52	91.874.822,99
PENALIZACIÓN SOBRESTOCK	31.000.000,00	180.000.000,00
TOTAL	1.001.817.581,52	369.374.822,99

Tabla 8. Comparación de costos obtenidos con modelo de inventario EOQ y algoritmo de optimización genético.

5. Conclusión y trabajos futuros

En general, se cumplieron los objetivos del trabajo en cuanto a que se desarrolló una herramienta capaz de predecir la venta futura y proponer un inventario óptimo por local. Los resultados tanto para la predicción de venta como para la optimización de inventario reflejan el comportamiento esperado para cada variable y se ajustan mayormente a las limitaciones intrínsecas del problema. De todas maneras, existen áreas en las cuales se puede continuar trabajando para mejorar los resultados y compatibilizarlos con la operación diaria de la compañía.

Ambos modelos de *forecast* fueron capaces de aprender la morfología diaria de venta, lo cual se refleja en los gráficos mostrados y en el RMSE obtenido. Sin embargo, para ambos algoritmos fue un desafío enfrentar aquellos días en que la venta real fue cero, lo cual es algo habitual en tiendas de menor exposición o productos de baja rotación. Si bien XGBoost fue la metodología que mejor ajuste tuvo, existe evidencia para pensar que puede haber sobreajuste, como el bajo valor de MASE. Futuras expansiones al trabajo podrían considerar técnicas para mitigar este problema, como ajustar hiperparámetros o incluir variables especializadas como Lasso (L1) y Ridge (L2). Adicionalmente, los modelos entrenados consideran la categoría del producto, por lo que también es necesario definir la técnica para asignar venta futura a cada modelo/color.

Los datos usados para verificar las predicciones corresponden a la venta de enero y la mitad de febrero. Por ser periodo estival, estos meses son los de menor venta en el año y no se encuentran cerca de ningún evento comercial. Por esto, es muy relevante en el futuro monitorear el comportamiento de las predicciones cerca de fechas comerciales relevantes (como el día de la Madre) o periodos de venta explosiva de productos estacionales. Como explicamos en un comienzo, la predicción de venta en estos escenarios es la más delicada y la más relevante para tomar decisiones. Antes de decidir si implementar una solución de este tipo, se debe verificar que sea capaz de predecir correctamente el comportamiento de fechas o productos sensibles. En caso de no ser así, se puede explorar agregar más columnas caracterizadoras para la fecha, como el clima, las precipitaciones o la actividad sísmica.

Respecto a la optimización de inventario, se verificó lo señalado en la literatura de que los modelos basados en puntos de *reorder* son menos flexibles para redes de muchos nodos. Al tener distintos parámetros por cada local/categoría, fue necesario fijar un calendario de visitas semanal para evitar que el modelo se gatillara diariamente y aumentaran los costos. Por el contrario, el algoritmo de optimización genético da mayor flexibilidad para agrupar visitas y agregar restricciones. Esto hace que la metaheurística sea adecuada para múltiples locales. Sin embargo, al tener usar una penalización por superar la capacidad máxima en vez de una restricción, ocurrió que en varias tiendas el inventario diario superaba el límite. Exploraciones futuras podrían aumentar el peso de la penalización para que el modelo se aleje de soluciones que incurran en ella. Si bien



el límite de bodega es una recomendación, no es apropiado para el flujo de trabajo que los locales tengan producto almacenado en pasillos o con apilados riesgosos.

Una de las dificultades encontradas fue la de incorporar la capacidad de exhibición de tiendas. Como ya comentamos, la predicción de venta se realizó por categoría y en este trabajo no se asignó por producto, lo que hizo imposible saber el tamaño propuesto de exhibición. Además, cada modelo de calzado debe ir acompañado de una serie de tallas, por lo que al abrir el inventario de categoría a producto es probable que este aumente para acomodar todas las numeraciones. Este sería un punto para mejorar del trabajo, que lo acercaría más a la realidad de los locales.



6. Glosario

- **ANN:** Artificial Neural Network
- **ARIMA:** Autoregressive Integrated Moving Average
- **CART:** Classification And Regression Trees
- **CNN:** Convolutional Neural Networks
- **CRISP-DM:** Cross-Industry Standard Process for Data Mining
- **DNN:** Deep Neural Network
- **EOQ:** Economic Order Quantity
- **ERP:** Enterprise Resource Planning
- **GRU:** Gated Recurrent Unit
- **KDD:** Knowledge Discovery from Databases
- **LSTM:** Long Short-Term Memory
- **RF:** Random Forest
- **RNN:** Recurrent Neural Network
- **SARIMA:** Seasonal ARIMA
- **SKU:** Stock Keeping Unit

7. Referencias

- Afshari, H., & Benam, F. H. (2011). Retail logistics. *Logistics operations and management. Concepts and models*, 267-289.
- Ali, Ö. G., Sayın, S., Van Woensel, T., & Fransoo, J. (2009). SKU demand forecasting in the presence of promotions. *Expert Systems with Applications*, 36(10), 12340-12348.
- Andrade, L. A. C., & Cunha, C. B. (2023). Disaggregated retail forecasting: A gradient boosting approach. *Applied Soft Computing*, 141, 110283.
- Arkaresvimon, R. (2008). *Inventory optimization in a retail multi-echelon environment* (Doctoral dissertation, Massachusetts Institute of Technology).
- Arunraj, N. S., & Ahrens, D. (2015). A hybrid seasonal autoregressive integrated moving average and quantile regression for daily food sales forecasting. *International Journal of Production Economics*, 170, 321-335.
- Atamtürk, A., Berenguer, G., & Shen, Z. J. (2012). A conic integer programming approach to stochastic joint location-inventory problems. *Operations Research*, 60(2), 366-381.
- Banco Central de Chile. (n.d.). PIB regional a precios nominales (base 2018). Recuperado el 19 de enero de 2025, de https://si3.bcentral.cl/Siete/ES/Siete/Cuadro/CAP_ESTADIST_REGIONAL/MN_REGIONAL1/CCNN2018_PIB_REGIONAL_N/638459358735081532
- Beheshti-Kashi, S., Karimi, H. R., Thoben, K. D., Lütjen, M., & Teucke, M. (2015). A survey on retail sales forecasting and prediction in fashion markets. *Systems Science & Control Engineering*, 3(1), 154-161.
- Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, Y., Maddix, D., Turkmen, C., ... & Januschowski, T. (2022). Deep learning for time series forecasting: Tutorial and literature survey. *ACM Computing Surveys*, 55(6), 1-36.
- Bes, C., & Sethi, S. P. (1988). Concepts of forecast and decision horizons: Applications to dynamic stochastic optimization problems. *Mathematics of Operations Research*, 13(2), 295-310.
- Brightwork Research & Analysis. (n.d.). *How to best understand multi echelon planning*. Recuperado de <https://www.brightworkresearch.com/multiechelon-inventory-optimization-defined/>
- Brodie, R. J., Whittome, J. R., & Brush, G. J. (2009). Investigating the service brand: A customer value perspective. *Journal of business research*, 62(3), 345-355.

- Cárdenes, M. V., Serna, F. J. D., & Morales, J. C. C. (2015). Inventory planning with dynamic demand: A state of art review. *DYNA: revista de la Facultad de Minas. Universidad Nacional de Colombia. Sede Medellín*, 82(190), 182-191.
- Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., & Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide. *SPSS inc*, 9(13), 1-73.
- Choi, T. M., Hui, C. L., & Yu, Y. (Eds.). (2013). *Intelligent fashion forecasting systems: models and applications*. Springer Science & Business Media.
- Cooper, L. G., Baron, P., Levy, W., Swisher, M., & Gogos, P. (1999). PromoCast™: A new forecasting method for promotion planning. *Marketing Science*, 18(3), 301-316.
- Deng, C., & Liu, Y. (2021). A Deep Learning-Based Inventory Management and Demand Prediction Optimization Method for Anomaly Detection. *Wireless Communications and Mobile Computing*, 2021(1), 9969357.
- Domínguez-Almendros, S., Benítez-Parejo, N., & Gonzalez-Ramirez, A. R. (2011). Logistic regression models. *Allergologia et immunopathologia*, 39(5), 295-305.
- Dudek, G. (2015). Short-term load forecasting using random forests. In *Intelligent Systems' 2014: Proceedings of the 7th IEEE International Conference Intelligent Systems IS'2014, September 24-26, 2014, Warsaw, Poland, Volume 2: Tools, Architectures, Systems, Applications* (pp. 821-828). Springer International Publishing.
- Eglite, L., & Birzniec, I. (2022). Retail sales forecasting using deep learning: Systematic literature review. *Complex Systems Informatics and Modeling Quarterly*, (30), 53-62.
- Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3), 37-37.
- Fildes, R., Ma, S., & Kolassa, S. (2022). Retail forecasting: Research and practice. *International Journal of Forecasting*, 38(4), 1283-1318.
- Guo, X., Liu, C., Xu, W., Yuan, H., & Wang, M. (2014, July). A prediction-based inventory optimization using data mining models. In *2014 Seventh International Joint Conference on Computational Sciences and Optimization* (pp. 611-615). IEEE.
- Hasan, M. R., Kabir, M. A., Shuvro, R. A., & Das, P. (2022). A comparative study on forecasting of retail sales. *arXiv preprint arXiv:2203.06848*.
- Hausman, W. H., & Erkip, N. K. (1994). Multi-echelon vs. single-echelon inventory control policies for low-demand items. *Management Science*, 40(5), 597-602.
- Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1), 388-427.

- Hopp, W. J. (2011). *Supply chain science*. Waveland Press.
- Hyndman, R. (2015). Another Look at Forecast Accuracy Metrics for Intermittent Demand. *Foresight: The International Journal of Applied Forecasting* 4, 43–46.
- Hyndman, R.J., & Athanasopoulos, G. (2018) *Forecasting: principles and practice*, 2nd edition, OTexts: Melbourne, Australia. Recuperado el 28 de enero de 2025, de: <https://otexts.com/fpp2/>.
- Jain, A., Menon, M. N., & Chandra, S. (2015). Sales forecasting for retail chains. *San Diego, California: UC San Diego Jacobs School of Engineering*.
- Janssens, G. K., & Ramaekers, K. M. (2011). A linear programming formulation for an inventory management decision problem with a service constraint. *Expert Systems with Applications*, 38(7), 7929-7934.
- Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, 80, 8091-8126.
- Khashei, M., & Bijari, M. (2011). A novel hybridization of artificial neural networks and ARIMA models for time series forecasting. *Applied soft computing*, 11(2), 2664-2675.
- Lindfors, A. (2021). Demand Forecasting in Retail: A Comparison of Time Series Analysis and Machine Learning Models.
- Liu, N., Ren, S., Choi, T. M., Hui, C. L., & Ng, S. F. (2013). Sales forecasting for fashion retailing service industry: a review. *Mathematical Problems in Engineering*, 2013(1), 738675.
- Loureiro, A. L., Miguéis, V. L., & Da Silva, L. F. (2018). Exploring the use of deep neural networks for sales forecasting in fashion retail. *Decision Support Systems*, 114, 81-93.
- Ma, S., & Fildes, R. (2017). A retail store SKU promotions optimization model for category multi-period profit maximization. *European Journal of Operational Research*, 260(2), 680-692.
- Macas, C. V. M., Aguirre, J. A. E., Arcentales-Carrión, R., & Peña, M. (2021, March). Inventory management for retail companies: A literature review and current trends. In *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)* (pp. 71-78). IEEE.
- Malik, S., Khan, M., Abid, M. K., & Aslam, N. (2024). Sales Forecasting Using Machine Learning Algorithm in the Retail Sector. *Journal of Computing & Biomedical Informatics*, 6(02), 282-294.
- McKinsey & Company. (2023, 7 de diciembre). What is fast fashion? <https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-fast-fashion#/>

Ministerio de Economía, Fomento y Turismo. (2024). *Informe de resultados: Séptima Encuesta Longitudinal de Empresas.* <https://www.economia.gob.cl/wp-content/uploads/2024/12/informe-de-resultados-ele7-final.pdf>

Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7, 21.

Plotnikova, V., Dumas, M., & Milani, F. (2020). Adaptations of data mining methodologies: A systematic literature review. *PeerJ Computer Science*, 6, e267.

Punia, S., Nikolopoulos, K., Singh, S. P., Madaan, J. K., & Litsiou, K. (2020). Deep learning with long short-term memory networks and random forests for demand forecasting in multi-channel retail. *International journal of production research*, 58(16), 4964-4979.

Radhakrishnan, P., Prasad, V. M., & Gopalan, M. R. (2009). Inventory optimization in supply chain management using genetic algorithm. *International Journal of Computer Science and Network Security*, 9(1), 33-40.

Randall, W. S., Gibson, B. J., Clifford Defee, C., & Williams, B. D. (2011). Retail supply chain management: key priorities and practices. *The International Journal of Logistics Management*, 22(3), 390-402.

Revista Logistec. (2024, 21 de Octubre). Radiografía del mercado de las bodegas en Chile: Innovación, crecimiento y servicios clave. Recuperado el 19 de enero de 2025, de <https://www.revistalogistec.com/empresas/punto-de-vista-2/5896-radiografia-del-mercado-de-las-bodegas-en-chile-innovacion-crecimiento-y-servicios-clave>

Schmidhuber, J., & Hochreiter, S. (1997). Long short-term memory. *Neural Comput*, 9(8), 1735-1780.

Shafique, U., & Qaiser, H. (2014). A comparative study of data mining process models (KDD, CRISP-DM and SEMMA). *International Journal of Innovation and Scientific Research*, 12(1), 217-222.

Silver, E. A., Pyke, D. F., & Peterson, R. (1998). *Inventory management and production planning and scheduling* (Vol. 3, p. 30). New York: Wiley.

Singh, S. R., & Kumar, T. (2011). Inventory optimization in efficient supply chain management. *International Journal of Computer Applications in Engineering Sciences*, 1(4), 428-434.

Szczepanek, R. (2022). Daily streamflow forecasting in mountainous catchment using XGBoost, LightGBM and CatBoost. *Hydrology*, 9(12), 226.

Thiruverahan, N., & Subramanian, N. (2015). Data mining and machine learning based approach to inventory prediction-a case study. In International conference on Control, Automation and Artificial Intelligence (CAAI) (pp. 195–202).



Valipour, M. (2015). Long-term runoff study using SARIMA and ARIMA models in the United States. *Meteorological Applications*, 22(3), 592-598.

Wen, X., Choi, T. M., & Chung, S. H. (2019). Fashion retail supply chain management: A review of operational models. *International Journal of Production Economics*, 207, 34-55.

Zhang, Z. H., & Unnikrishnan, A. (2016). A coordinated location-inventory problem in closed-loop supply chain. *Transportation Research Part B: Methodological*, 89, 127-148.

Zippia (2023, 15 de Junio). "28 Dazzling Fashion Industry Statistics [2023]: How Much Is The Fashion Industry Worth", <https://www.zippia.com/advice/fashion-industry-statistics/>

Apéndice I

El Apéndice I contiene un extracto del código escrito en lenguaje Python que permite obtener y procesar los datos de venta para dejarlos en un formato apropiado para los modelos de *forecast*.

```
df = df.sort_values(by='Fecha', ascending=True)
df.describe()

# Rellenar valores nulos en columna Evento
df['Evento'] = df['Evento'].fillna('NOEVENTO')
# Reemplazar los valores negativos de venta por 0
df['Uns'] = df['Uns'].clip(lower=0)
# Unificar la subfamilia de sandalias para mayor simpleza
df['Nom_SubFamilia'] = df['Nom_SubFamilia'].apply(lambda x: 'SANDALIA'
if x.startswith('SANDALIA') else x)

# Detección de outliers
aux = df.copy()
aux['New_Cluster'] = aux['Nom_Marca'] + '_' + aux['Nom_SubFamilia']
aux2 = aux.groupby(['Fecha', "Cliente", "New_Cluster"], as_index=False)[["Uns"]].sum()
aux2 = aux2[~aux2['New_Cluster'].isin(['GACEL_PUNTA', 'GACEL_STRAP',
'GUANTE_GUANTE', 'GUANTE_CAMPERA', 'GACEL_GUANTE', 'GUANTE_MASCARILLA',
'GACEL_OTRO ACCESORIO', 'GUANTE_BUFDANA'])]

# Crear subgráficos
fig, axes = plt.subplots(nrows=9, ncols=7, figsize=(20, 30))

# Obtener los valores únicos de 'New_Cluster'
clusters = aux2['New_Cluster'].unique()

# Dibuja un boxplot para cada valor único de 'New_Cluster'
for i, (cluster, ax) in enumerate(zip(clusters, axes.flat)):
    sns.boxplot(data=aux2[aux2['New_Cluster'] == cluster], y='Uns',
    ax=ax)
    ax.set_title(f'Cluster {cluster}')

plt.tight_layout()
plt.savefig("grafico_boxplot.png", dpi=300, bbox_inches='tight')
plt.show()

# Eliminación de outliers
indice = [261724, 261725, 486766, 486769, 960525, 960526, 1174913,
1174914]
df = df.drop(index=indice)

# Creamos una nueva base donde se elimina la columna Cod_Producto y se
agrupa según las demás columnas, dejando el total de unidades y venta
por dia/cluster
data = df.copy()
```

```
data['New_Cluster'] = data['Nom_Marca'] + ' ' + data['Nom_SubFamilia']
+ ' ' + data['Cluster']
data = data.groupby(["Fecha", "Dia Sem", "Sem del año", "Feriado",
"Evento", "New_Cluster"], as_index=False)[["Uns", "Neto final"]].sum()

data.head()

# Agregamos el precio unitario (eliminando los nulos y errores)
data['PrecioUnit'] = data['Neto final'] / data['Uns']
data["PrecioUnit"] = data["PrecioUnit"].replace([np.inf, -np.inf],
np.nan) # Reemplazar infinitos por NaN
data["PrecioUnit"].fillna(0, inplace=True)

# Revisamos si hay valores nulos
data.isnull().sum()

# Estandarización de datos categóricos
cat_features = ["Feriado", "Evento", "New_Cluster"]
num_features = ["Fecha", "Dia Sem", "Sem del año", "Uns", 'PrecioU-
nit']

data_stnd = data.copy()

label_encoders = {}
encoded_dfs = [] # Lista para almacenar DataFrames codificados

for col in cat_features:
    encoder = OneHotEncoder(sparse_output=False, handle_unknown="ig-
nore")
    encoded_array = encoder.fit_transform(data_stnd[[col]])

    # Convertir a DataFrame con nombres de columnas adecuados
    encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_fea-
ture_names_out([col]), index=data_stnd.index)

    encoded_dfs.append(encoded_df) # Guardamos el DataFrame codifi-
cado
    label_encoders[col] = encoder # Guardamos el encoder para rever-
tir si es necesario

# Concatenar todas las columnas codificadas con las numéricas origina-
les
data_stnd = pd.concat([data_stnd[num_features]] + encoded_dfs, axis=1)

data_stnd.set_index('Fecha', inplace=True)
```

Apéndice II

El Apéndice II contiene un extracto del código escrito en lenguaje Python que entrena un modelo ARIMA para predecir venta.

```
atrain = data_arima.iloc[:val]
atest = data_arima.iloc[val:]

X0_train = atrain.drop(columns=['Uns'])
y0_train = atrain['Uns']
X0_test = atest.drop(columns=['Uns'])
y0_test = atest['Uns']

resultado = adfuller(y0_train)
print(f"p-valor: {resultado[1]}")

if resultado[1] > 0.05: # Si la serie no es estacionaria, aplicamos
    # diferenciación
    y0_train = y0_train.diff().dropna()
    X0_train = X0_train.loc[y0_train.index] # Asegurar que X tenga el
    # mismo índice que y tras la diferenciación

modelo_auto = auto_arima(
    y0_train, exogenous=X0_train, seasonal=False, trace=True,
    stepwise=True
)

print(modelo_auto.summary())

# Extraer los mejores parámetros obtenidos de auto_arima
p, d, q = modelo_auto.order

# Ajustar el modelo SARIMAX con regresores exógenos
modelo = ARIMA(y0_train, order=(p, d, q), exog=X0_train)
modelo_arima = modelo.fit()
print(modelo_arima.summary())

n_periodos = 30 # Predecir 30 días adelante

# Para predecir a futuro, necesitamos valores de los regresores también
X0_forecast = X0_test.iloc[-n_periodos:] # Tomamos los últimos valores conocidos como estimación
y0_forecast = y0_test.iloc[-n_periodos:]

forecast = modelo_arima.get_forecast(steps=n_periodos, exog=X0_forecast)
predicciones = forecast.predicted_mean
```

Apéndice III

El Apéndice III contiene un extracto del código escrito en lenguaje Python que implementa un modelo XGBoost para predecir venta.

```
dtrain = data_xgb.iloc[:train]
dval = data_xgb.iloc[train:val]
dtest = data_xgb.iloc[val:]

# Definir variables predictoras y objetivo
X1_train = dtrain.drop(columns=['Uns'])
y1_train = dtrain['Uns']
X1_val = dval.drop(columns=['Uns'])
y1_val = dval['Uns']
X1_test = dtest.drop(columns=['Uns'])
y1_test = dtest['Uns']

# Crear DMatrix para XGBoost
dmtrain = xgb.DMatrix(X1_train, label=y1_train)
dmval = xgb.DMatrix(X1_val, label=y1_val)
dmtest = xgb.DMatrix(X1_test, label=y1_test)

# Definir parámetros del modelo
params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'learning_rate': 0.05,
    'max_depth': 4,
    'subsample': 0.8,
    'colsample_bytree': 0.8
}

# Entrenar el modelo
model = xgb.train(params, dmtrain, num_boost_round=1000,
evals=[(dmval, 'test')], early_stopping_rounds=10)

# Predicción
y1_pred = model.predict(dmtest)
```

Apéndice IV

El Apéndice IV contiene un extracto del código escrito en lenguaje Python con las funciones diseñadas para crear una metodología EOQ de inventario.

```
# Definimos una función que calcula la cantidad a reponer dado un forecast de venta y un horizonte de tiempo que se desea provisionar
def EOQ(forecast, horizonte):
    # Estadísticos de venta en el periodo
    mu = forecast.groupby(['Cliente', 'New_Cluster'], as_index=False)['Uns'].mean()
    sigma = forecast.groupby(['Cliente', 'New_Cluster'], as_index=False)['Uns'].std()
    # Demanda total
    D = mu['Uns']
    # Demanda diaria
    D_day = D
    # Lead Time
    LD = 3
    # Cantidad a pedir Q
    Q = D_day * horizonte

    # Nivel de servicio = 95%
    CSL = 0.95

    # Venta promedio en el periodo
    mu_ld = mu['Uns'] * LD / periodo
    # Desv Est de venta en el periodo
    sigma_ld = sigma['Uns'] * math.sqrt(LD / periodo)
    # Calculamos k según el nivel de servicio deseado
    k = round(stats.norm(0, 1).ppf(CSL), 2)
    # Punto de reorder
    s = mu.copy()
    s['Reorder'] = mu_ld + k * sigma_ld
    s = s.reset_index()

    s['Reorder'] = s['Reorder'].fillna(0)

    return s, Q

def stock_diario(forecast, s, Q):
    # s, Q son diccionarios

    # Obtener el listado de todos los clientes y categorías (New_Cluster)
    clientes = forecast['Cliente'].unique()
    categorias = forecast['New_Cluster'].unique()

    # Convertir stock_inicial a un diccionario para acceso rápido
    stock_inicial_dict = stock_inicial.to_dict(orient='index')  #
    Clientes como claves y categorías como columnas
    # Preprocesar forecast en un DataFrame más útil
```

```
forecast['Fecha'] = pd.to_datetime(forecast['Fecha']) # Asegurar que las fechas estén en formato datetime
forecast['t'] = (forecast['Fecha'] - forecast['Fecha'].min()).dt.days # Convertir las fechas a días (t)

# Inicializar el diccionario de stock_diario
stock_diario = {}

# Asignar stock inicial al t=0 en el stock_diario
for cliente in clientes:
    for categoria in categorias:
        if cliente in stock_inicial_dict and categoria in stock_inicial_dict[cliente]:
            stock_diario[(cliente, categoria, 0)] = stock_inicial_dict[cliente].get(categoria, 0)

# Crear diccionario para acceder rápidamente a las ventas diarias
forecast_dict = {(row['Cliente'], row['New_Cluster'], row['t']): row['Uns'] for _, row in forecast.iterrows()}

# Calcular el stock diario para cada periodo (t)
for t in range(1, periodo + 1):
    for cliente in clientes:
        for categoria in categorias:
            # Obtén las ventas diarias (si existen) del forecast
            venta = forecast_dict.get((cliente, categoria, t), 0)

            # Calcular el stock del día t
            stock_diario[(cliente, categoria, t)] = stock_diario.get((cliente, categoria, t - 1), 0) - venta

            # Verificar si es necesario realizar un reabastecimiento (según s_dict y Q_dict)
            reorder_point = s_dict.get((cliente, categoria), 0)
            if stock_diario[(cliente, categoria, t)] < reorder_point:
                reorder_quantity = Q_dict.get((cliente, categoria), 0)
                stock_diario[(cliente, categoria, t)] += reorder_quantity # Reabastecer

    return stock_diario

def stock_local_diario(stock_diario):
    # Inicializamos un diccionario para almacenar el stock por cliente
    stock_local_diario = {}

    # Iteramos sobre el diccionario de stock_diario
    for (cliente, categoria, t), stock in stock_diario.items():
        # Aseguramos que la clave cliente exista en el diccionario
        if cliente not in stock_local_diario:
            stock_local_diario[cliente] = np.zeros(periodo + 1) # Creamos un array de ceros para cada cliente
```

```

# Sumamos el stock por cada cliente y cada período t
stock_local_diario[cliente][t] += stock

return stock_local_diario

# Debemos ahora calcular 'horizonte' para no sobrepasar el límite de
capacidad de las tiendas
# Usaremos un algoritmo de tipo 'Greedy' para buscar

horizonte_opt = 20
tiendas_sobrerestock = 1000

for i in range(20,0,-1):
    s, Q = EOQ(forecast, i)
    s_dict = s.set_index(['Cliente', 'New_Cluster'])['Uns'].to_dict()
    Q_dict = {(row['Cliente'], row['New_Cluster']): Q[i] for i, row in
    s.iterrows()}
    stock_diario_opt = stock_diario(forecast, s_dict, Q_dict)
    stock_local_diario_opt = stock_local_diario(stock_diario_opt)
    stock_df = pd.DataFrame(stock_local_diario_opt, columns=L['Cli-
ente'].unique())
    stock_df = pd.DataFrame(stock_local_diario_opt, columns=L['Cli-
ente'].unique()).T
    stock_df = stock_df.reset_index().rename(columns={"index": "Cli-
ente"})
    stock_df.sort_values(by=['Cliente'], inplace=True)

    df_merged = stock_df.merge(L, on='Cliente')
    df_merged['Conteo'] = (df_merged.iloc[:, 1:-1].gt(df_merged['Ca-
pacidad'], axis=0)).sum(axis=1)
    resultado = df_merged[['Cliente', 'Conteo']]

    if (df_merged['Conteo'] > 0).sum() == 0:
        horizonte_opt = i
        break

print(f"El horizonte óptimo es: {horizonte_opt}")

```

Apéndice V

El Apéndice V contiene un extracto del código escrito en lenguaje Python con el algoritmo genético desarrollado para optimizar inventario.

```
#Evalua la población y devuelve el mejor individuo
def EvaluarPoblacion(poblacion):
    mejor_solucion = []
    mejor_costo = 10e100
    for p in poblacion:
        costo_referencia = funcionObj(p)
        if costo_referencia < mejor_costo:
            mejor_solucion = p
            mejor_costo = costo_referencia
    return mejor_solucion, mejor_costo

# Definimos la función objetivo a minimizar
def funcionObj(solucion):
    visitas, stock_obj = solucion

    costo_visitas = sum(visitas[cliente][dia] * c_dict[cliente] for
cliente in visitas for dia in visitas[cliente])

    # Simulación de venta perdida
    venta_perdida = calcular_venta_perdida(solucion)

    penalizacion_capacidad = sum(max(0, stock_obj[cliente][categoria] -
L_dict[cliente])
                                   for cliente in stock_obj for categoria
in stock_obj[cliente]))

    return costo_visitas + 10000 * venta_perdida + 100000 * penaliza-
cion_capacidad

# Función para calcular la venta perdida
def calcular_venta_perdida(solucion):
    visitas, stock_obj = solucion

    # Diccionario para almacenar el stock diario, inicializamos con
valores de stock_inicial para t=0
    stock_diario = {}
    for cliente in locales:
        stock_diario[cliente] = {}
        for categoria in categorias:
            stock_diario[cliente][categoria] = {}
            for dia in range(periodo+1):
                # Asignar un valor aleatorio para Uns
                stock_diario[cliente][categoria][dia] = 0

    # Inicializamos el stock diario en t=0 con el stock_inicial
    for cliente in locales:
        for categoria in categorias:
```

```

        stock_diario[cliente][categoria][0] = stock_inicial_dict[cliente][categoria]

    venta_perdida_total = 0

    # Iteración por cada día t (de 1 a periodo)
    for t in range(1, periodo):
        for cliente in locales:
            for categoria in categorias:
                # Si es un día de reposición
                if visitas[cliente][t] == 1:
                    # Actualizamos el stock con la reposición
                    stock_diario[cliente][categoria][t] =
max(stock_obj[cliente][categoria],stock_diario[cliente][categoria][t-1])

                # Luego, restar la venta esperada del stock disponible
                venta = forecast_dict.get((cliente, categoria, t-1),
0)
                stock_diario[cliente][categoria][t] = stock_diario[cliente][categoria][t] - venta

                # Cálculo de venta perdida
                venta_perdida = max(0, venta - stock_diario[cliente][categoria][t])
                venta_perdida_total += venta_perdida

    return venta_perdida_total

# Definimos función que crea una solución aleatoria
def crear_solucion():
    visitas = {cliente: {dia: random.choice([0, 1]) for dia in
range(periodo)} for cliente in locales}
    stock_obj = {cliente: {new_cluster: random.uniform(5, 10) for
new_cluster in categorias} for cliente in locales}

    return [visitas, stock_obj]

#Genera una poblacion inicial de soluciones de tamaño N
def generar_poblacion(N):
    return [crear_solucion() for _ in range(N)]

#Funcion de cruce. Recibe una poblacion(lista de soluciones) y devuelve la población ampliada con los hijos.
# Todos los individuos de la población son seleccionados para el cruce(si la población es par)
# Podría aplicarse un proceso previo de selección para elegir los individuos que se desea cruzar.
def Cruzar(poblacion, mutacion):
    poblacion_copia = copy.deepcopy(poblacion)
    poblacion_final = []

    while len(poblacion_copia) > 1:
        padre1, padre2 = random.sample(poblacion_copia, 2)

```

```

    poblacion_copia.remove(padre1)
    poblacion_copia.remove(padre2)
    # Generar hijos
    hijos = Descendencia([padre1, padre2], mutacion)
    poblacion_final.extend(hijos)

    return poblacion_final

#Funcion para generar hijos a partir de 2 padres:
def Descendencia(padres,mutacion):
    #La descendencia se hará combinando las matrices de reposición y
    stock de ambos padres
    visitas1, stock_obj1 = padres[0]
    visitas2, stock_obj2 = padres[1]
    hijo1 = [visitas1, stock_obj2]
    hijo2 = [visitas2, stock_obj1]
    return [hijo1,hijo2,Mutar(hijo1,mutacion),Mutar(hijo2,mutacion) ]

#Funcion de mutación.
#La mutación consistirá en invertir en la matriz de reposición dos pa-
rejas de días
def Mutar(solucion, mutacion):
    if random.random() < mutacion:
        visitas, stock_obj = solucion
        # Seleccionamos dos días aleatorios para mutar
        dias_cambiar = random.sample(range(len(visitas[next(iter(visi-
tas))])), 2) # Asegura que el índice esté basado en los días
        # Intercambiar los valores de las visitas en esos días
        for cliente in visitas:
            visitas[cliente][dias_cambiar[0]], visi-
            tas[cliente][dias_cambiar[1]] = visitas[cliente][dias_cambiar[1]], vi-
            sitas[cliente][dias_cambiar[0]]
            # Devolver la solución mutada
        return [visitas, stock_obj]
    else:
        return solucion

#Funcion de selección de la población. Recibe como parametro una po-
blación y
# devuelve una población a la que se ha eliminado individuos poco ap-
tos(fitness bajo) y para mantener una población estable de N indivi-
duos
#Se tiene en cuenta el porcentaje elitismo pasado como parámetro
# Para los individuos que no son de la élite podríamos usar una selec-
ción de ruleta(proporcional a su fitness)
def Seleccionar(poblacion, N, elitismo):
    #Se ordena la población según el fitness(tamaño del recorrido) en
    una lista de elementos [costo, solucion]
    poblacion_ordenada = sorted([ [funcionObj(solucion), solucion] for
    solucion in poblacion ], key= lambda x:x[0] )

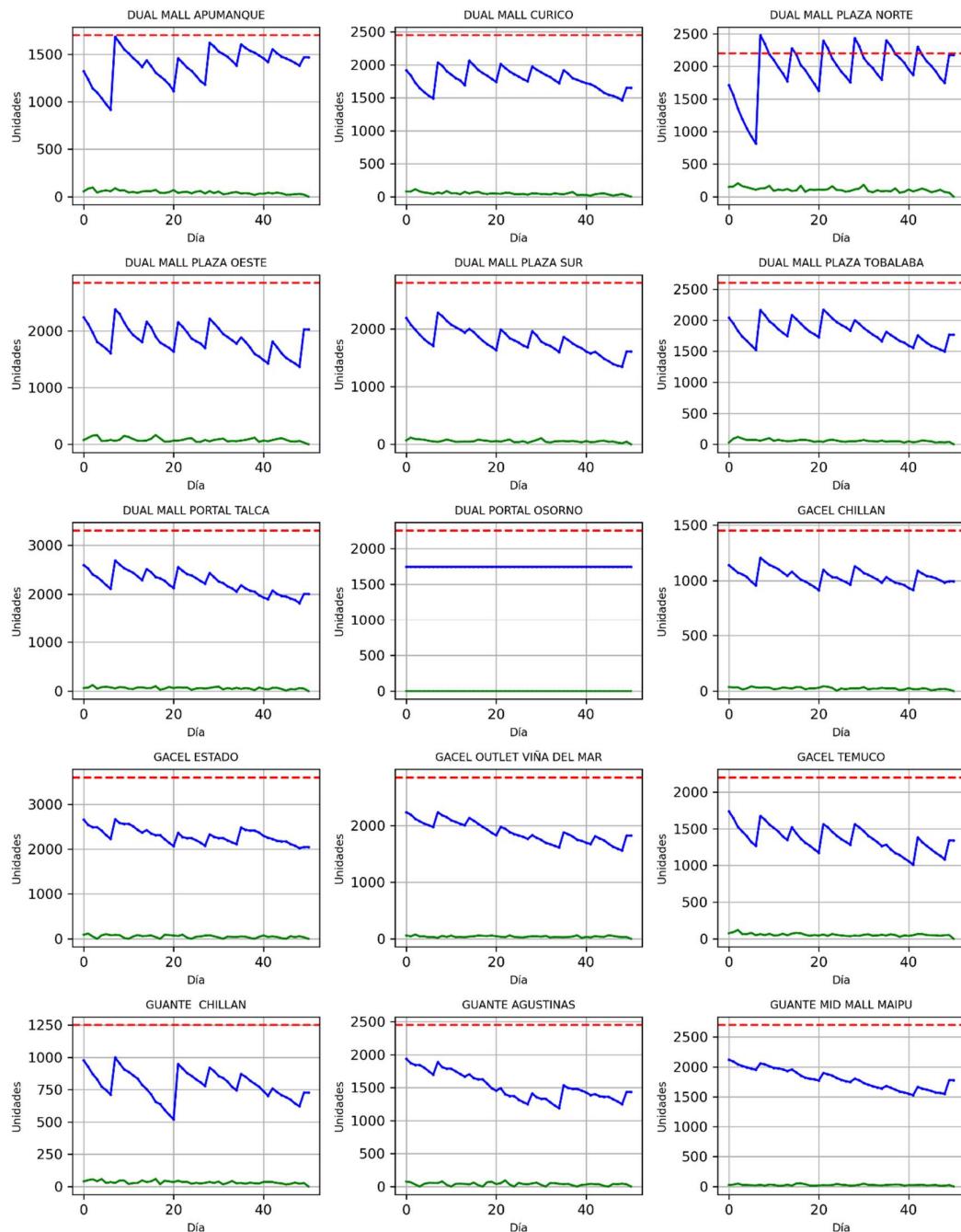
    #Devolvemos elitismo% y el resto se eligen aleatoriamente

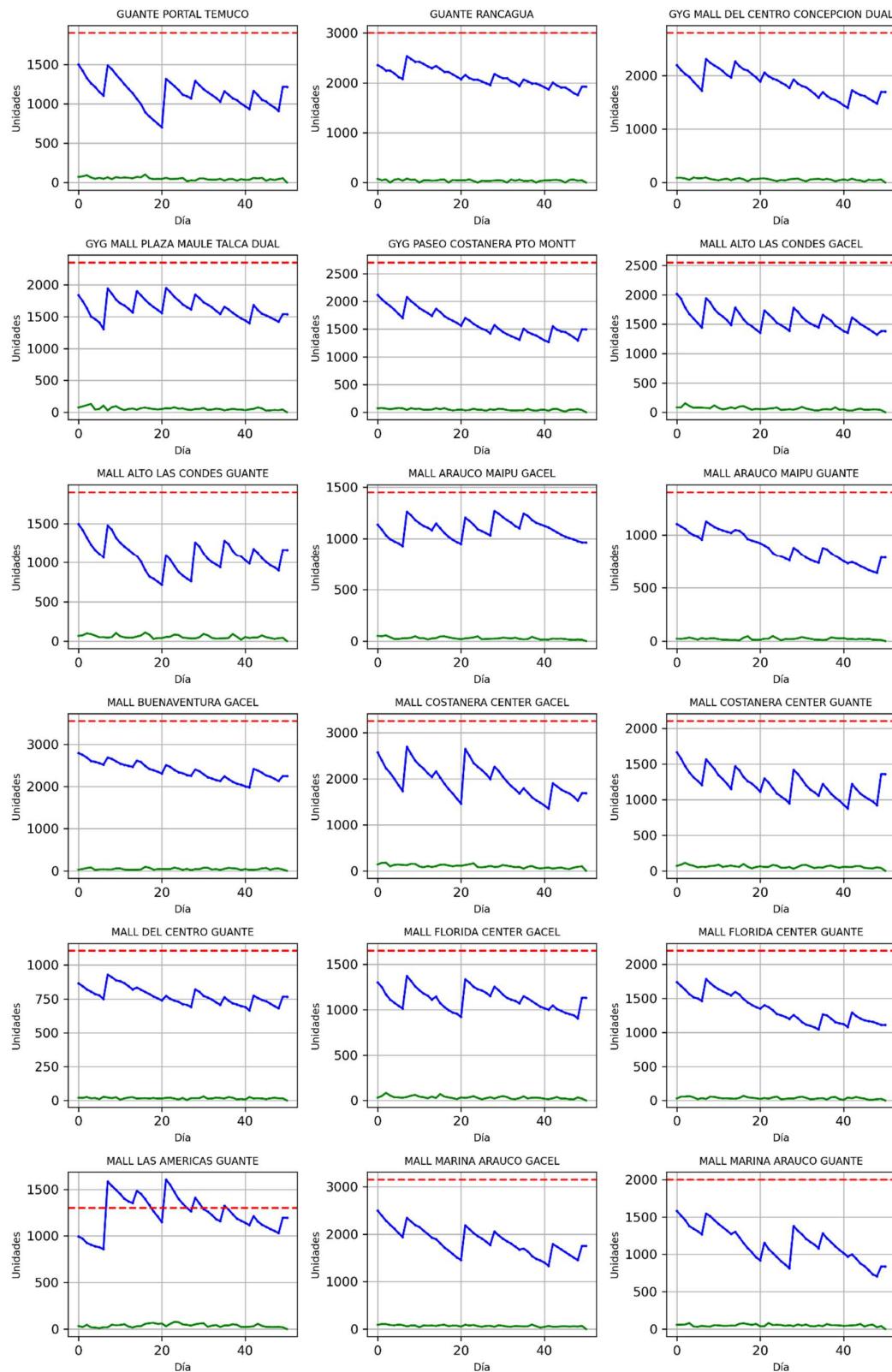
```

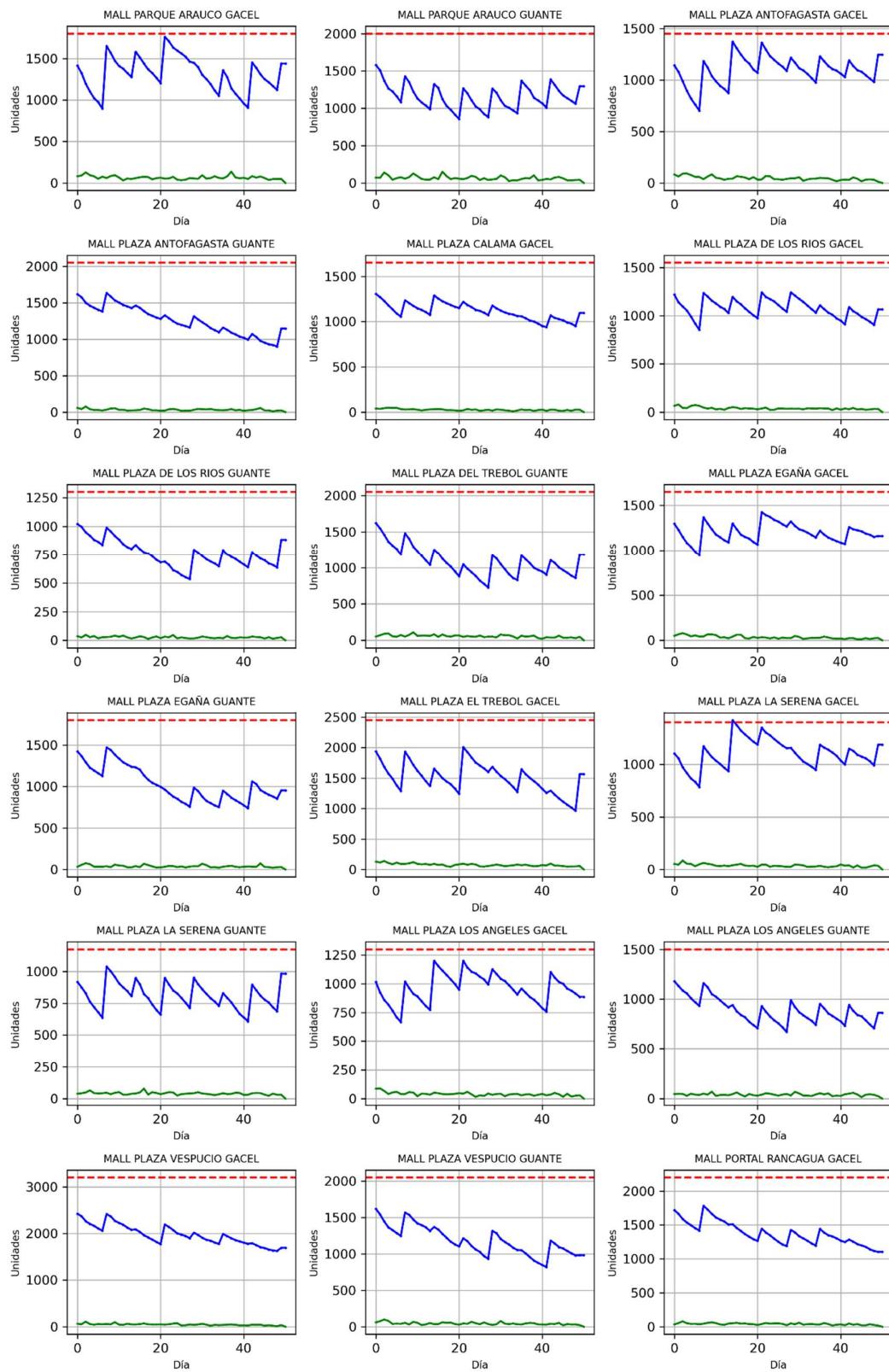
```
    return [x[1] for x in poblacion_ordenada][:int(N*elitismo)] +  
random.sample([x[1] for x in poblacion_ordenada][int(N*elitismo):],  
int(N*(1-elitismo)) )  
  
#Funcion principal del algoritmo genetico  
def algoritmo_genetico(N, mutacion, elitismo, generaciones):  
    poblacion = generar_poblacion(N)  
    mejor_solucion = None  
    mejor_costo = float('inf')  
  
    for generacion in range(generaciones):  
        # Cruce y mutación  
        poblacion = Cruzar(poblacion, mutacion)  
  
        # Selección de la población  
        poblacion = Seleccionar(poblacion, N, elitismo)  
  
        # Evaluación  
        mejor_solucion, mejor_costo = EvaluarPoblacion(poblacion)  
  
        print(f"Generación {generacion} - Mejor Costo: {mejor_costo:.2f}")  
  
    return mejor_solucion  
  
solucion = algoritmo_genetico(N=100, mutacion=0.2, elitismo=0.1, generaciones=100)
```

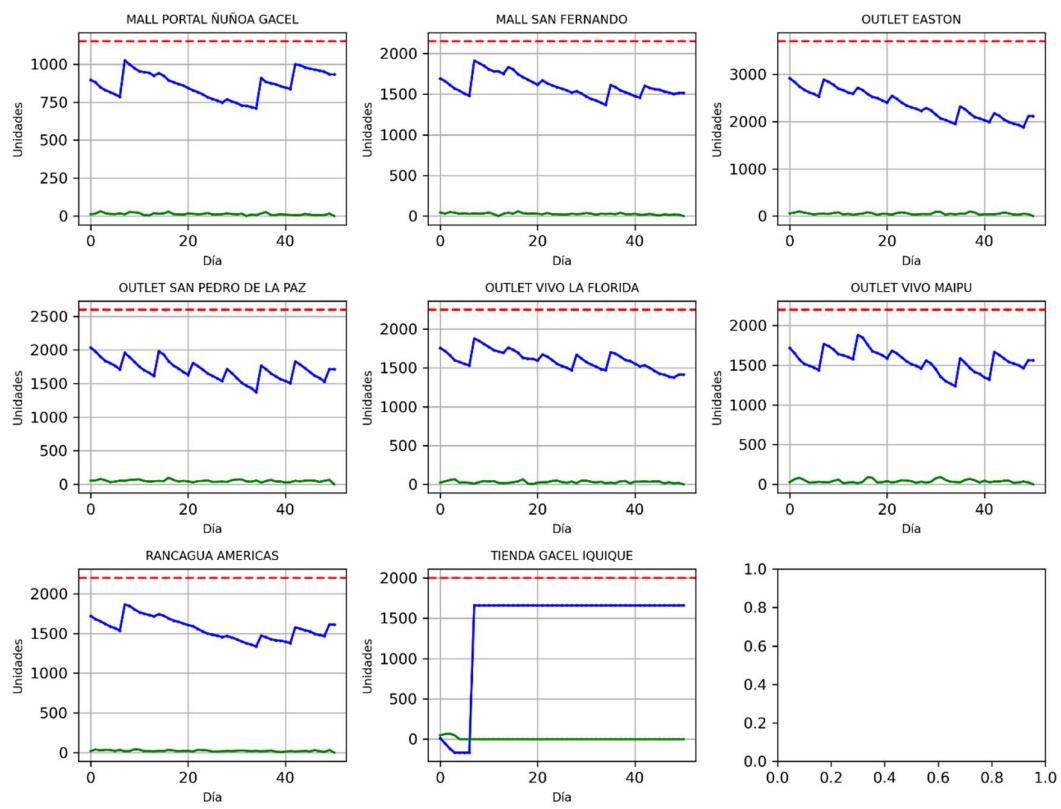
Apéndice VI

El Apéndice VI contiene gráficos que muestran la simulación de inventario diario, la venta diaria y la capacidad máxima por tienda en un horizonte de 50 días simulados usando el modelo EOQ planteado en este trabajo.









Apéndice VII

El Apéndice VII contiene gráficos que muestran la simulación de inventario diario, la venta diaria y la capacidad máxima por tienda en un horizonte de 25 días simulados usando el algoritmo genético planteado en este trabajo.

