Escuela de Negocios
Tipo de documento: Tesis de maestría

Master in Management + Analytics

Predicción y distribución de demanda eficiente a través de la optimización logística

Autoría: Tirone, Pablo Gabriel
Año: 2025

UNIVERSIDAD TORCUATO DI TELLA

MÁSTER IN MANAGEMENT + ANALYTICS

PREDICCIÓN Y DISTRIBUCIÓN DE DEMANDA EFICIENTE A TRAVÉS DE LA OPTIMIZACIÓN LOGÍSTICA

TESIS

Pablo Gabriel Tirone

Mayo 2025

Tutor: Javier Marenco

RESUMEN

En el contexto actual, la logística ha dejado de ser una función meramente operativa para convertirse en un componente estratégico dentro de las organizaciones. Su correcto funcionamiento permite no solo garantizar la disponibilidad de productos en tiempo y forma, sino también optimizar los recursos de la empresa en un entorno cada vez más competitivo y complejo. En particular, la capacidad de anticipar la demanda y de diseñar un sistema de distribución eficiente se vuelve esencial para responder con agilidad a las necesidades del mercado.

En Argentina, el costo logístico representa en promedio entre un 12% y un 18% de la facturación de una empresa, aunque en ciertos sectores puede incluso superar el 20%. Este valor se ve impactado por múltiples factores estructurales: la infraestructura vial, la distancia entre los centros de producción y consumo, la concentración geográfica de las economías regionales y las ineficiencias en los procesos internos de planificación. En este sentido, cada punto porcentual que se logre optimizar en logística representa una mejora significativa en la rentabilidad de la operación.

En este trabajo se propone un modelo de predicción de demanda y un modelo de optimización de transporte que busca resolver el problema de distribución desde un centro logístico hacia múltiples puntos de venta, minimizando el costo total y atendiendo la demanda de manera eficiente. El modelo considera no solo el costo logístico asociado a cada movimiento de mercadería, sino también el costo de insatisfacción derivado de no poder abastecer la totalidad de la demanda proyectada. Para esto, se parte de una predicción de demanda por producto y sucursal, lo cual permite anticipar necesidades y definir decisiones de asignación de recursos con mayor fundamento. La salida del modelo incluye tanto el costo logístico como el costo de insatisfacción, lo que permite evaluar el trade-off entre eficiencia operativa y nivel de servicio.

El análisis está circunscrito a la provincia de Córdoba que opera con un centro de distribución central que termina abasteciendo a diferentes sucursales dentro de la provincia. En este proyecto no se buscó únicamente ayudar a la empresa en cuestión a optimizar la forma en la que se planifica para reducir costos logísticos y tomar mejores decisiones, sino que también a poder reducir la cantidad de personas que trabajan en el área de planificación estratégica (20 personas) que diariamente necesitan hacer ajustes dado el modelo actual.

ABSTRACT

In today’s context, logistics has evolved from being a purely operational function to becoming a strategic component within organizations. Its proper execution not only ensures the availability of products in the right place and at the right time but also enables companies to optimize resources in an increasingly competitive and complex environment. In particular, the ability to anticipate demand and design an efficient distribution system is essential to respond with agility to market needs.

In Argentina, logistics costs represent on average between 12% and 18% of a company's revenue, and in certain sectors, this figure can even exceed 20%. This value is influenced by various structural factors such as road infrastructure, the distance between production and consumption centers, the geographic concentration of regional economies, and inefficiencies in internal planning processes. In this sense, each percentage point optimized in logistics can lead to a significant improvement in operational profitability.

This project proposes a transportation optimization model aimed at solving the distribution problem from a central warehouse to multiple retail branches, minimizing total cost while efficiently meeting demand. The model considers not only the logistical cost associated with each merchandise movement but also the dissatisfaction cost resulting from not being able to fulfill the projected demand. To this end, it begins with a demand forecast by product and branch, allowing for better anticipation of needs and more informed resource allocation decisions. The model's output includes both logistical and dissatisfaction costs, which enables the evaluation of trade-offs between operational efficiency and service level.

The analysis focuses on the province of Córdoba, which operates with a central distribution center supplying several branches within the province. Compared to the current operational setup, this project not only aims to assist the company that kindly provided and allowed the use of its data in making better decisions, but also to reduce the workload of the strategic planning team—currently composed of 20 people—who must make daily manual adjustments under the existing model.

Índice

1. INTRODUCCIÓN

1.1. Contexto

La industria del consumo masivo y los desafíos de planificación logística

La industria del consumo masivo, tal como la conocemos hoy, es el resultado de más de dos siglos de evolución industrial, tecnológica y comercial. Desde la Revolución Industrial hasta la actualidad, la forma en que se producen, distribuye y consumen bienes ha cambiado drásticamente, impulsada por una demanda cada vez más dinámica y una competencia global cada vez más feroz. En este escenario, la eficiencia dentro de la cadena de suministro —o *supply chain*— se vuelve un factor crítico para la rentabilidad y la supervivencia de las empresas.

A lo largo del tiempo, la cadena de sumistro se ha ido profesionalizando y sofisticando, integrando múltiples actores: proveedores, plantas, distribuidores, puntos de venta, operadores logísticos, y especialmente, áreas de planificación que cumplen un rol fundamental a la hora de anticipar la demanda y definir cómo satisfacerla operativamente. Ya no se trata únicamente de producir y entregar, sino de **anticipar cuánto y cuándo se va a necesitar cada producto**, y planificar su distribución de la forma más eficiente posible. **La planificación logística eficiente exige tomar decisiones interdependientes en un contexto de alta incertidumbre**, lo que convierte esta tarea en un desafío complejo pero abordable con las herramientas adecuadas.

El rol de la logística y la planificación y la necesidad de adaptarse modelos inteligentes

Dentro de esta red interconectada, dos actores emergen como claves para garantizar que los productos lleguen en tiempo y forma a los puntos de venta: **el distribuidor**, responsable de mover físicamente la mercadería a lo largo del territorio, y **el planificador**, encargado de anticipar la demanda y tomar decisiones estratégicas sobre cómo asignar los recursos disponibles. Ambos roles están íntimamente vinculados, ya que una planificación poco precisa puede generar rutas ineficientes, quiebres de stock o sobreacumulación de inventario, mientras que una logística sin flexibilidad puede limitar incluso la mejor planificación.

El trabajo de estos actores se ve tensionado por múltiples decisiones operativas que deben resolverse a diario: ¿cuánto y cuándo abastecer cada sucursal?, ¿cómo distribuir el stock limitado entre múltiples destinos?, ¿qué productos priorizar cuando no se puede cubrir toda la demanda?, ¿cómo organizar el reparto en función de las restricciones de capacidad o volumen? En la mayoría de las organizaciones, estas decisiones se toman de forma manual o apoyadas en herramientas básicas, lo cual puede generar ineficiencias significativas, especialmente cuando el volumen de datos, productos y destinos crece de forma considerable.

En este contexto, resulta indispensable avanzar hacia soluciones que incorporen **modelos inteligentes** de apoyo a la decisión, capaces de combinar predicciones de demanda con algoritmos de optimización que permitan simular múltiples escenarios, priorizar criterios estratégicos y encontrar asignaciones eficientes con restricciones reales. Modelos como los que

se proponen en este trabajo —basados en programación lineal y aplicados a la distribución de productos en una región específica— no solo permiten reducir costos logísticos, sino también **medir y minimizar el impacto económico de no satisfacer completamente la demanda**, algo que en la práctica se traduce en pérdida de ventas y disminución del nivel de servicio.

La implementación de estos modelos, si bien no reemplaza la experiencia y criterio de los planificadores, **ofrece un marco estructurado y repetible para tomar mejores decisiones**, optimizando el uso de los recursos disponibles. A su vez, estos enfoques permiten escalar el proceso de planificación sin depender exclusivamente del análisis manual de grandes volúmenes de datos, facilitando la automatización de tareas y reduciendo la carga operativa de los equipos.

En definitiva, el objetivo de este trabajo es **explorar cómo la combinación de modelos de predicción de demanda y algoritmos de optimización logística puede contribuir a resolver problemas reales en empresas del sector**, donde las decisiones operativas están fuertemente influenciadas por la incertidumbre, la disponibilidad limitada de recursos y la presión constante por mejorar la eficiencia.

1.2. Literatura Previa

El desarrollo de modelos de optimización en logística se inscribe dentro del marco de la **Investigación Operativa** (Operational Research, OR), una disciplina que aplica herramientas analíticas para mejorar la toma de decisiones en contextos complejos. La programación lineal, uno de sus pilares, tiene su origen en los trabajos de **George Dantzig**, quien en 1947 desarrolló el método *simplex*, una herramienta que revolucionó la capacidad de resolver problemas con múltiples variables y restricciones. Aunque su propuesta original no estaba orientada exclusivamente al ámbito logístico, sentó las bases para abordar situaciones reales como la distribución de recursos escasos, planificación de transporte y asignación de inventarios, todas problemáticas presentes en los sistemas de abastecimiento actuales (Dantzig, 1963).

En el marco de esta tesis, uno de los aportes más significativos proviene del artículo **"A Robust Optimization Approach to Supply Chain Management"**, desarrollado por **Dimitris Bertsimas y Aurélie Thiele** (2004). Este trabajo plantea una metodología alternativa para enfrentar la incertidumbre en variables clave como la demanda, sin requerir que dicha incertidumbre se modele con distribuciones probabilísticas complejas. La propuesta consiste en definir **conjuntos de incertidumbre acotados**, sobre los cuales se buscan soluciones que permanezcan viables bajo todos los posibles escenarios dentro de ese conjunto. A diferencia de enfoques estocásticos que requieren simulaciones o datos históricos amplios, este enfoque robusto es más flexible y menos dependiente de la calidad de los datos. En nuestro caso, si bien no se incorpora aún un componente explícito de riesgo o variación de la demanda por limitaciones de alcance y tiempo, **el marco teórico planteado por Bertsimas constituye una referencia directa** y abre una posible línea futura de mejora para el modelo aquí desarrollado (Bertsimas & Thiele, 2004).

Por otro lado, el estudio de **Tsao, Mangotra, Lu y Dong (2012)** se enfoca en los desafíos logísticos que enfrentan las empresas con **redes de distribución amplias y descentralizadas**, donde la asignación de inventario no puede basarse únicamente en decisiones locales. En este tipo de entornos, caracterizados por una diversidad de centros de distribución, zonas geográficas con

diferentes patrones de consumo y limitaciones logísticas propias, se vuelve esencial diseñar modelos que consideren el sistema en su conjunto. Este es precisamente uno de los objetivos centrales de nuestro modelo, que busca distribuir recursos de forma eficiente desde un centro logístico a múltiples sucursales, cada una con restricciones de capacidad y requerimientos mínimos. El trabajo de Tsao y colaboradores aporta herramientas conceptuales para entender esta complejidad sistémica y justificar el enfoque adoptado en nuestra tesis (Tsao, Mangotra, Lu & Dong, 2012).

La relevancia del **costo logístico** dentro del modelo se ve reforzada por el estudio de **Juhász y Bányai (2018)**, quienes analizan los desafíos operativos y económicos vinculados a la **logística de última milla**, es decir, la etapa final del proceso de distribución desde el centro de almacenamiento hasta los puntos de venta o consumo. Esta etapa es reconocida como una de las más costosas dentro de la cadena de suministro, dado que implica mayor fragmentación en los envíos, aumento del uso de recursos (como vehículos y personal) y menor eficiencia operativa. En el contexto argentino, este aspecto cobra aún más importancia si consideramos que los costos logísticos pueden representar entre el 12% y el 18% de la facturación de una empresa, o incluso más en sectores específicos. Incluir explícitamente este costo dentro de la función objetivo del modelo permite no solo obtener una solución económicamente óptima, sino también visibilizar el impacto que las decisiones logísticas tienen en la rentabilidad (Juhász & Bányai, 2018).

En resumen, el modelo desarrollado en este trabajo encuentra su respaldo conceptual en múltiples líneas de investigación: desde la formulación clásica de la programación lineal, pasando por enfoques modernos de optimización robusta para enfrentar incertidumbre en la demanda, hasta modelos aplicados a la asignación eficiente de inventario en redes logísticas complejas. El enfoque adoptado permite balancear la eficiencia económica (a través del costo logístico) y el nivel de servicio (a través del costo de insatisfacción), generando un modelo que, si bien es determinístico en esta versión, se encuentra preparado para futuras extensiones más sofisticadas que integren elementos de estacionalidad, incertidumbre o decisiones dinámicas multiperiodo.

1.3. Problema

En función de lo mencionado anteriormente, el presente análisis se centra en una empresa que comercializa productos a través de dos canales principales: la venta online y la venta física en sucursales.

El problema por abordar se centra en la mejora de la solución logística actual del retailer, buscando principalmente reducir los costos y el retrabajo manual en el proceso de entrega al cliente.

En el próximo apartado se describe el proceso actual de planificación y distribución que se está utilizando y a partir de dicho diagnóstico, se planteará un objetivo concreto y se desarrollará una metodología de trabajo orientada a proponer mejoras que contribuyan a optimizar la situación actual.

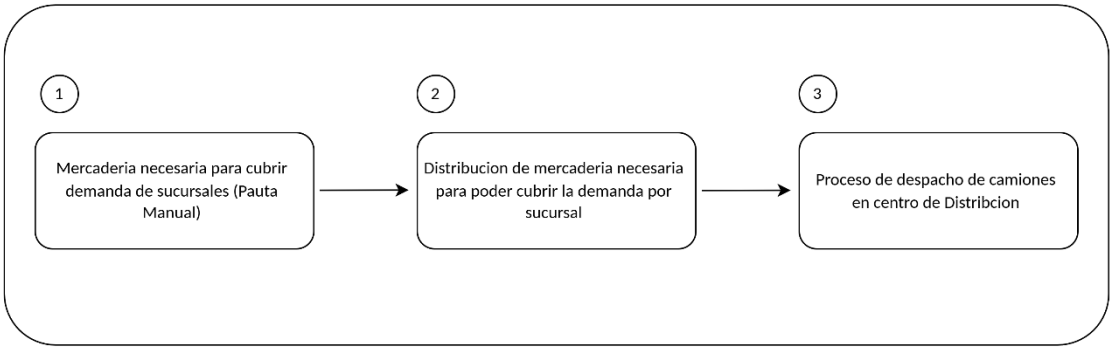

Figura 1: Proceso actual de planeación de envío de productos a sucursales en retail

En la f**igura 1** se plantea el proceso actual de planeamiento logístico del retailer bajo análisis que presenta tres etapas principales: determinación de demanda por sucursal, asignación de mercadería desde el origen, y despacho de productos hacia cada destino.

Etapa 1 – Determinación de la demanda por sucursal

El primer paso del proceso consiste en estimar la cantidad de mercadería necesaria para abastecer a cada sucursal. Este cálculo se realiza a partir de la corrida de un algoritmo propio de la compañía que se ajusta en función de variables básicas como el historial de ventas o estacionalidades conocidas. Sin embargo, este algoritmo fue originalmente desarrollado para un contexto empresarial distinto al actual, con un foco principal en evitar quiebres en los puntos de exhibición de las sucursales (el algoritmo es de finales de la década del 90, las tendencias de consumo desde ya cambiaron muchísimo ya que se introdujeron otros canales de consumo como venta web, todo esto sumado a que la velocidad de rotación ya no es la misma). Con el paso del tiempo, este proceso se expandió hacia depósitos y centros logísticos, manteniendo la misma lógica operativa, lo que llevó a una sobreestimación sistemática de la demanda.

El algoritmo corre con una frecuencia semanal, mientras que las entregas a sucursales son diarias, lo que genera una brecha temporal que compromete la precisión de las decisiones. El resultado de este desfase es un volumen de mercadería que, en promedio, excede lo realmente necesario, lo que se traduce en costos adicionales por sobre stock y dificultades en la gestión operativa de las entregas. Para el mes de diciembre 2024 el proceso arrojó un error porcentual medio superior al 200%.

Etapa 2 – Asignación de origen y destino

La segunda etapa consiste en la distribución de los productos desde el centro de distribución a cada sucursal. Este proceso se realiza de manera mayoritariamente manual por parte de los planificadores, quienes, apoyados en hojas de cálculo, cruzan la información obtenida del paso anterior con el stock actual de cada punto de venta y generan una propuesta a cada sucursal de la mercadería que debe enviarse día a dia.

La salida de este proceso es un calendario mensual de entregas diarias, construido de forma empírica que no refleja lo que pasa realmente y generalmente debe ser ajustados manualmente. Estas planillas son recibidas por cada sucursal las cuales revisan que lo que se haya propuesto como envio sea correcto y en el caso de que se necesiten ajustes se realizan de manera manual

y se reenvían a los planificadores para que coordinen correctamente las cantidades a enviar con el equipo de logística. Este proceso diario se conoce como proceso de **"pauta manual".**

El principal inconveniente que se detecta en esta etapa es el sobredimensionamiento de las necesidades logísticas. Esto se manifiesta especialmente en la contratación de servicios de transporte, donde se suelen reservar más vehículos que los efectivamente necesarios para ciertas sucursales y suelen faltar para otras.

Adicionalmente, el hecho de que tanto la generación de datos como su procesamiento se realicen de forma manual dificulta la posibilidad de escalar el modelo o automatizar decisiones. Esto representa una carga significativa de trabajo para el equipo de planificación y limita su capacidad de respuesta ante cambios súbitos en la demanda. En esta parte del proceso es donde se consume la mayor carga horaria del personal ya que tienen que revisar el informe de la totalidad de sucursales diariamente. Por otro lado, dependiendo del tipo de productos que haya finalmente que enviar puede necesitarse la contratación de ultimo momento de ciertos camiones para cubrir la demanda de sucursales específicas.

Etapa 3 – Distribución de productos en vehículos y despacho

La última etapa del proceso involucra el despacho de mercadería desde el centro de distribución hacia cada sucursal. Actualmente, no existe un sistema que permita automatizar el armado de camiones ni la distribución interna del volumen transportado. Esta operación se ejecuta de forma completamente manual, lo que genera múltiples reprocesos por errores de carga de información, un sistema ayudaría a que las personas puedan enfocarse en la validación de datos para que se pueda optimizar la carga y el ocupamiento de espacio en los vehículos.

En muchas ocasiones, la mercadería planificada para un camión no puede ser efectivamente estibada, lo que da lugar a la necesidad de reconfigurar la carga sobre la marcha, con pérdidas de eficiencia y potenciales errores de despacho. Además, se identifican casos en los que productos terminan siendo cargados "a la fuerza" lo que provoca daños no intencionales evitables.

Conclusión del análisis del proceso actual

Este proceso de planificación y distribución presenta limitaciones estructurales que afectan directamente a la eficiencia operativa del retailer. Entre los principales problemas se destacan:

- La falta de actualización del algoritmo de demanda, que genera sobreasignaciones sistemáticas.

- La dependencia de planillas y operaciones manuales, que impide escalar o automatizar decisiones clave.

- La ausencia de un sistema de armado logístico inteligente, que impacta en la precisión y efectividad del despacho de mercadería.

La existencia de estas limitaciones abre una oportunidad concreta para el desarrollo de herramientas de apoyo a la toma de decisiones que integren modelos predictivos de demanda y algoritmos de optimización logística. La incorporación de estas herramientas no solo permitiría reducir costos y mejorar el nivel de servicio, sino también liberar al equipo de planificación de tareas operativas repetitivas para poder enfocarse en decisiones estratégicas de mayor valor.

1.4. Objetivo

Plantear una función de abastecimiento logístico integral que incluya predicción de la demanda más función óptima de distribución entre orígenes y destinos por SKU de manera de reducir costos asociados al envío de mercadería hacia sucursales disminuyendo perdidas por costos de incumplimiento de demanda. El output del proceso es diario y se va a hacer foco el la fecha específica del 08 de diciembre de 2024. Abarcará únicamente la zona geográfica en la que trabaja el retailer que cuenta con 1 depósito central (deposito 111) y 5 sucursales (la sucursal 81, sucursal 82, sucursal 169,sucursal 173 y sucursal 176).
En la **figura 2** se muestra el alcance del objetivo. Si bien el proceso de planeación de retailer incluye 3 etapas,únicamente se tendrán en cuenta dentro del objetivo las primeras dos etapas: predicción de demanda y alineación de orígenes y destinos.

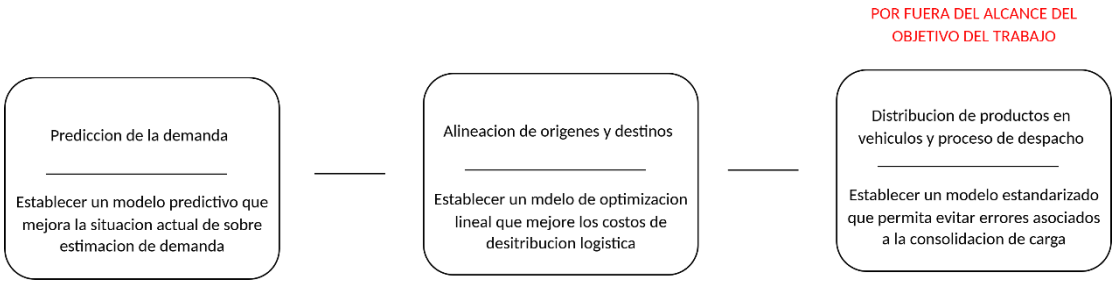

Figura 2: Alcance de objetivo del trabajo de predicción y distribución de demanda eficiente a través de la optimización logística.

2. DATOS

El retailer en análisis gestiona un volumen de datos considerable, para tener una magnitud por mes procesa más de cincuenta mil ordenes proveniente de sus operaciones tanto de venta online como en sucursales físicas. Esta información es central para la toma de decisiones comerciales y logísticas, y constituye la base para el desarrollo de los modelos predictivos y de optimización planteados en este trabajo.

Toda la información relevante se almacena en una **base de datos centralizada**, donde los usuarios de diferentes áreas tienen acceso para consultar, analizar y extraer datos según sus necesidades. Para la obtención de la información, se utilizan consultas en **lenguaje SQL**, a partir de las cuales se generan reportes que luego son exportados en **formatos CSV**. Estos archivos planos son el insumo principal para alimentar los algoritmos que permiten tanto predecir la demanda como planificar la distribución de productos.

El retailer administra una gran cantidad de productos, agrupados en distintas categorías, marcas y tipos de presentación, lo que genera una diversidad de SKUs que supera solo para el mundo retail más de 4000 SKUs distintos por año. Básicamente esta complejidad requiere un control constante sobre el inventario, los niveles de stock en cada sucursal y la evolución de las ventas por canal y por punto de venta. A su vez, debido a la dinámica propia del consumo de productos que vende el retailer—donde se lanzan, rotan o discontinuan constantemente—, los datos de catálogo también presentan cambios frecuentes que deben ser gestionados en los reportes operativos.

La periodicidad de actualización de los datos es diaria, permitiendo tener una visibilidad detallada de las ventas, el stock y los movimientos logísticos. Sin embargo, la calidad y disponibilidad de la información dependen en gran medida de los procesos de extracción manuales que cada usuario realiza, lo cual introduce potenciales riesgos de inconsistencias o desfases si no se controlan adecuadamente.

A pesar de contar con una infraestructura de base de datos robusta, en la práctica diaria **Excel sigue siendo una herramienta muy utilizada** para el procesamiento intermedio de los datos, tanto en reportes de control de stock como en seguimiento de ventas y proyecciones comerciales. Esta modalidad refleja una necesidad de contar con herramientas que reduzcan la intervención manual y permitan escalar el procesamiento de datos para soportar decisiones estratégicas más complejas.

A continuación, se comparten las diferentes tablas que se utilizaron para el proyecto y también se mencionan las transformaciones que se necesitaron para lograr el input inicial al modelo.

MODELO DE TABLAS UTILIZADAS EN EL PROYECTO:

1) Stock_Cordoba: La tabla muestra el detalle del stock que hay para las sucursales 81,82,169,173,176 y el depósito 111 de Córdoba. Las columnas que integran la tabla son las siguientes:

- cod_tit: sucursal o deposito.
- cod_articulo: código del SKU.
- cod_familia_art: código de la familia del artículo.
- nom_familia_art: nombre de la familia del artículo vinculado al anterior código.
- cantidad: cantidad de ese artículo.
- cod_estado: estado de la mercadería, solo importa cuando es normal o exhibición.
- fecha_mod: fecha de última modificación de cantidad o estado del artículo.
- nom_articulo: Nombre del artículo.

2) Ventas_sucursales: La tabla muestra todas las ventas realizadas para las sucursales 81,82,169,173 y 176 de enero a diciembre de 2024. Las columnas que integran la tabla son las siguientes:

- serie_factura: lugar donde se ejecutó la venta.
- Stock: lugar de donde sale o se entrega el stock para cumplir con la venta.
- cod_articulo: código del SKU.
- cantidad: cantidad entregada al cliente.
- fecha: fecha de la venta.
- Nombre archivo origen: archivo origen donde se trajo el dato de la venta

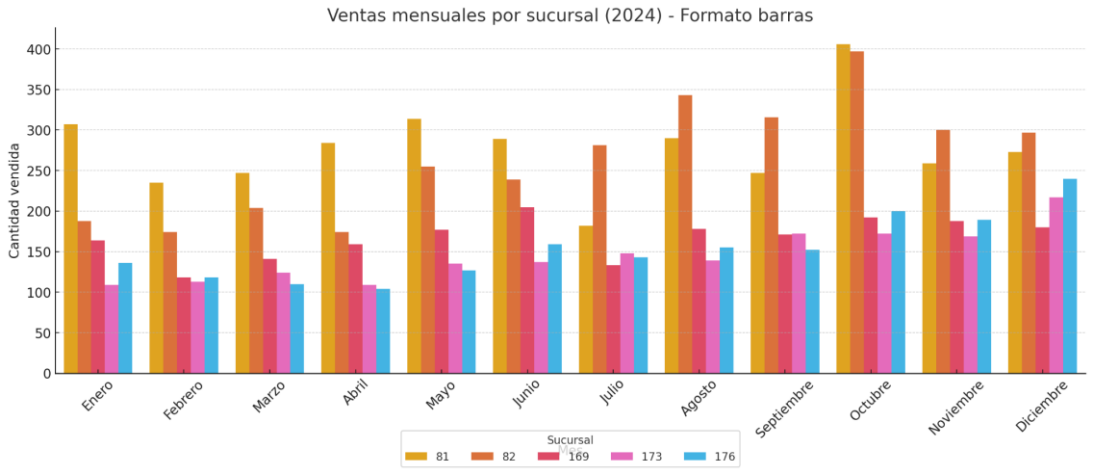

Figura 3: Distribución anual de ventas en sucursales de Córdoba.

En **la figura 3** puede verse la distribución anual de ventas mensuales generales de todos los productos para las diferentes sucursales de Córdoba. Claramente la sucursal 81 en conjunto con la sucursal 82 son las sucursales que mayor cantidad de ventas concentran teniendo una estacionalidad muy marcada en el mes de octubre debido a que en esa fecha se suele dar un evento promocional de productos electrónicos que traccionan la venta de productos generales en la compañía. El mes de diciembre si bien no es el mes que concentra la mayor cantidad de ventas es el mes que menos dispersión registra entre las ventas de las diferentes sedes comerciales.

3) **Relacion_sucursales:** tabla que muestra la relación de las sucursales y del depósito 111. Si una sucursal (aparece en la primera columna Sucursal) tiene en la columna hermanado el valor de otra sucursal quiere decir que la sucursal de la primera columna puede enviar mercadería a esta última sucursal. Ejemplo: Si en la columna llamada Sucursal aparece 111 y en la columna

llamada hermanado aparece 81 esto quiere decir que el depósito 111 puede enviar mercadería a 81. Las columnas que integran la tabla son las siguientes:

- Sucursal: Sucursal o depósito de origen de la mercadería.
- Nombre: Nombre de la sucursal o depósito de origen.
- Hermanado: Sucursal o depósito que puede recibir mercadería de la sucursal o depósito que aparece en la primera columna.
- Nombre: Nombre de la sucursal o depósito hermanada con la sucursal o depósito de la primera columna.

4) Costos_de_transporte: Tabla que muestra el costo por kilómetro para el traslado de una sucursal o depósito a otra sucursal o depósito. Las columnas que integran la tabla son las siguientes:

- Origen: Punto de Partida (puede ser una sucursal o un depósito).
- Destino: Punto de destino (puede ser una sucursal o un depósito).
- Costo por Km (ARS): Costo por kilómetro en pesos argentinos.

5) volumetria_final: tabla que muestra el lugar en m3 que ocupa cada uno de los artículos. Las columnas que integran la tabla son las siguientes:

- cod_articulo: el código del sku.
- volumen_art: el volumen en metros cúbicos (m3) que ocupa el artículo.

6) modelo_camiones: Tabla que muestra por modelo de camión que capacidad puede llevar cada uno de los camiones en metros cúbicos (m3). Las columnas que integran la tabla son las siguientes:

- Marca: Marca del camión en cuestión.
- Modelo: Modelo del camión en cuestión.
- Volumen (m3): Volumen en metros cúbicos que puede transportar.
- Tipo de camión: Marca si es camión rígido o de otro tipo.

7) volumetria_sucursales: Tabla que muestra el volumen de una sucursal específica en metros cúbicos. Las columnas que integran la tabla son las siguientes:

- Sucursal: Contiene el valor de la sucursal o depósito en cuestión.
- Espacio en M3: Marca el valor de almacenamiento de metros cúbicos asociado a una sucursal.

8) Pesos_Sucursales: Tabla que muestra el peso específico asociado a la importancia que tiene una sucursal por sobre otras en términos de que ante una situación en la cual solo hay cantidades para un solo sku la sucursal con mayor peso sea la abastecida.

- Sucursal: Contiene el valor de la sucursal o depósito en cuestión.

- Peso: Marca la importancia en la ponderación de las sucursales. Es un valor que va de 1 a 1000 siendo 1000 el valor más importante.

9) Sku_Killers: Contiene el valor de todos los Sku que tienen un tratamiento específico debido a que suelen tener una alta demanda en un periodo específico. Para estos Sku se trata de tener una cantidad específica de unidades siempre disponible en ciertas locaciones, es una tabla de actualización diaria. Las columnas que contiene la tabla son las siguientes:

- Sucursal: Contiene el valor de la sucursal o depósito en cuestión.
- SKU: Contiene el código identificador único del ítem (Stock Keeping Unit).
- Descripción: Contiene la información asociada a la descripción del ítem.
- Categoría: Categoría a la que pertenece el ítem.
- Cantidad: Cantidad mínima que tiene que tener una sucursal específica asociada a ese SKU.

11) Distancias: Tabla que muestra las distancias entre sucursales y depósitos. Las columnas que integran la tabla son las siguientes:

- Sucursal/Depósito Origen: Marca cual es el depósito o sucursal de origen.
- Sucursal/Depósito Destino: Marca cual es la sucursal o depósito de destino.
- Distancia: Distancia en kilómetros entre una sucursal/depósito y entre sucursales.

.

En las **figuras 4 y 5** se muestra como es la distribución de distancias entre el depósito 111, las sucursales y entre las sucursales.

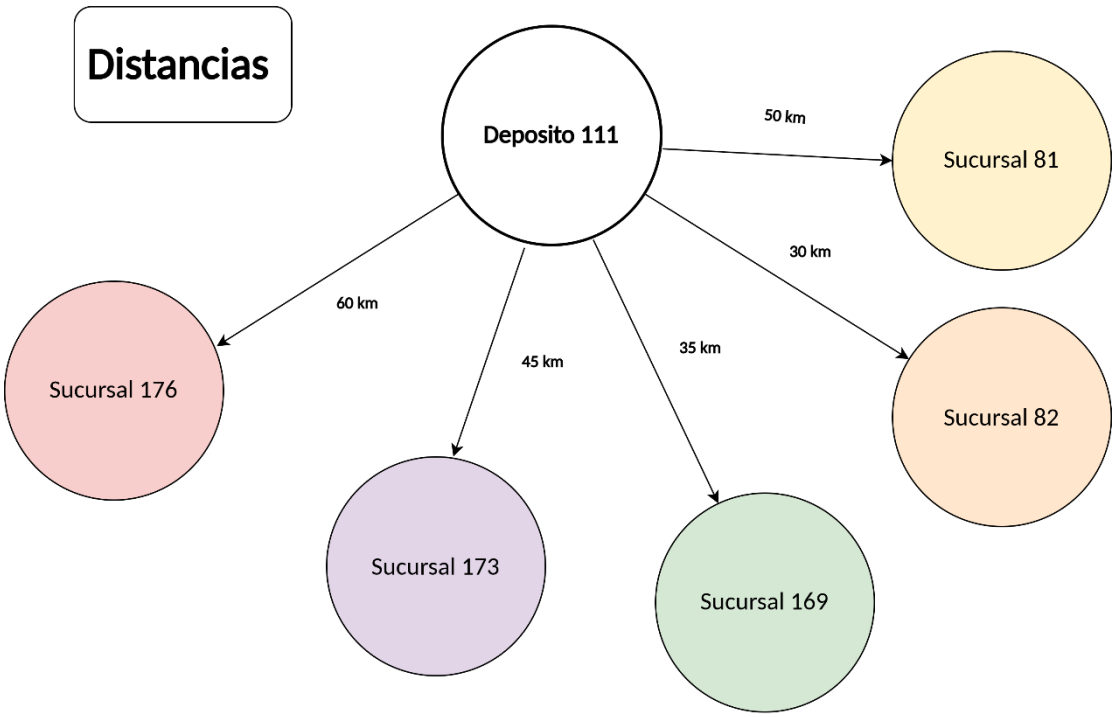

Figura 4: Distancia entre depósitos y sucursales.

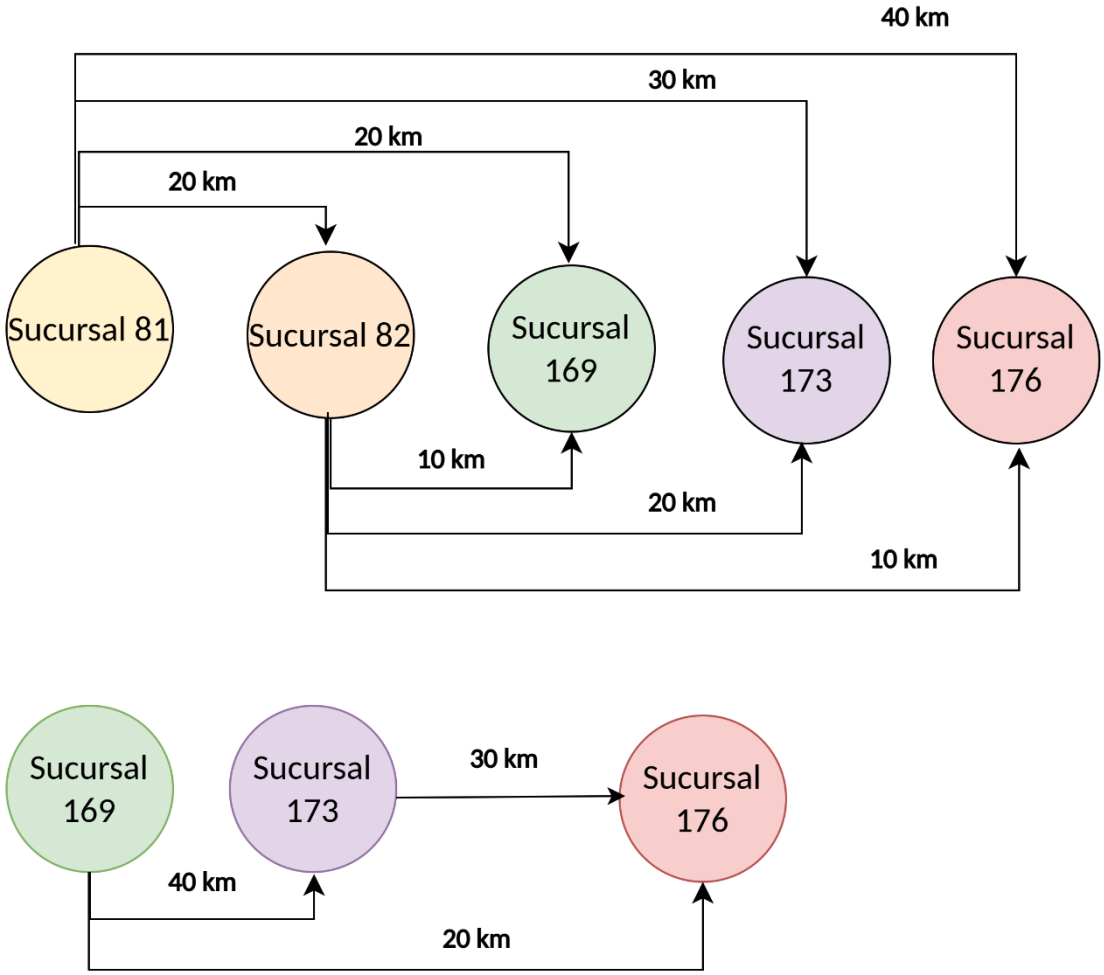

Figura 5: Distancias entre sucursales.

En función a la extracción de información de todas estas tablas se termina consolidando, tal cual se muestra en la tabla que está representada en la **tabla 1**, el archivo input para poder correr el modelo predictivo que es el primer paso del proceso de generación de información. Desde ese punto en adelante el modelo de optimización logística ya no necesita nueva información consolidada de manera manual. Uno de los puntos interesantes también a explorar es una mejora en los procesos de extracción y transformación de información para que los usuarios finales no necesiten trabajar consultas en lenguaje SQL de manera manual y de esa forma evitar errores involuntarios.

Fecha de venta	cod_articulo	Numero_sucursal	Sucursal	cantidad	mes	Categoria	Descripcion	Volumen	Killer
2/1/2024	13497	81	CÓRDOBA	1	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
9/1/2024	13497	81	CÓRDOBA	1	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
10/1/2024	13497	81	CÓRDOBA	3	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
11/1/2024	13497	81	CÓRDOBA	2	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
12/1/2024	13497	81	CÓRDOBA	3	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
13/1/2024	13497	81	CÓRDOBA	3	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
15/1/2024	13497	81	CÓRDOBA	3	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
16/1/2024	13497	81	CÓRDOBA	1	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer
18/1/2024	13497	81	CÓRDOBA	1	1	CAFÉ E INFUSIONES	PAVAPHILLIPS HD9368/00	0,018	Killer

Tabla 1: Tabla que muestra de manera consolidada la información listada más arriba.

A continuación, se detallan los campos del archivo consolidado:

- Fecha de venta: Fecha en la cual se produjo la venta de productos. No contiene información de ventas horarias, se muestra venta específica acumulada del día.
- Cod_articulo: Stock Key Unit asociada al producto en cuestión.
- Numero_sucursal: Número de la sucursal de venta.
- Cantidad: Cantidad vendida en ese día particular en la sucursal de ese SKU.
- Mes: Mes en el que se ejecutó la venta específica.
- Categoría: Categoría del producto.
- Descripción: Descripción del producto.
- Volumen: Volumen unitario del producto.

3. METODOLOGÍA

RESUMEN Y CONSOLIDACIÓN GENERAL DEL MODELO A IMPLEMENTAR:

Tal cual se plantea en la **figura 6**, el proceso de planificación logística planteado en este trabajo integra dos etapas principales: la predicción de la demanda futura y la posterior optimización de la distribución de mercadería hacia las sucursales. Ambas etapas se apoyan en modelos que procesan datos históricos y actuales extraídos de los sistemas del retailer.

El primer paso consiste en la preparación de la información de ventas históricas. A partir de la base de datos central de la empresa, los usuarios extraen mediante consultas SQL los registros de ventas semanales por SKU y sucursal. Estos datos son consolidados en un archivo plano, generalmente en formato CSV o XLS, que sirve como input para el modelo de predicción. La preparación incluye el agregado de feriados especiales y la organización de las fechas para facilitar la interpretación semanal de las ventas.

Una vez organizado el dataset, se procede a la ejecución del modelo de predicción de demanda, basado en la herramienta Prophet. Este modelo predictivo aditivo toma como insumo las series de ventas históricas y en conjunto con la configuración de ciertos parámetros que el usuario puede modificar de manera autónoma tales como intervalo de confianza, elección o no de capturar patrones diarios o semanales o incluir una lista de días feriados genera una proyección de ventas para un período futuro determinado, en este caso enfocado en semanas específicas de alta relevancia comercial. El modelo predice tanto valores puntuales como intervalos de confianza, permitiendo además generar versiones redondeadas de las predicciones para facilitar su uso logístico posterior. Una vez obtenidas las proyecciones, se exporta un nuevo archivo que contiene las cantidades esperadas por SKU y sucursal para el período proyectado.

Con la información de demanda proyectada disponible, se inicia la segunda etapa del proceso: la optimización logística de la distribución. El modelo de optimización recibe como input varios conjuntos de datos: las proyecciones de demanda, el stock inicial disponible en el centro de distribución y en las sucursales, las capacidades de almacenamiento de cada sucursal, las distancias entre ubicaciones, las categorías de los productos, y los volúmenes que ocupa cada SKU. Además, se consideran parámetros logísticos como la capacidad de carga de los camiones, el costo fijo y variable de los transportes, y las restricciones de mínimos de unidades por categoría que cada sucursal debe recibir.

La optimización se formula como un problema de programación lineal de minimización, donde la función objetivo busca reducir el costo total de operación. Este costo se compone de dos componentes principales: el costo logístico (transporte) y el costo de insatisfacción de demanda, asociado a las ventas no realizadas por falta de stock en destino. El modelo contempla también restricciones operativas críticas, como la conservación del stock (no enviar más unidades de las disponibles), el cumplimiento de capacidades de almacenamiento, y la obligación de cumplir mínimos operativos en función de las categorías de productos.

La ejecución del modelo genera una solución óptima que indica: Cuántas unidades de cada SKU deben ser enviadas desde el centro de distribución a cada sucursal; Qué movimientos de stock pueden hacerse entre sucursales, respetando siempre las transferencias permitidas y Cuántos camiones son necesarios para cubrir cada ruta de distribución, cumpliendo condiciones mínimas de carga para habilitar su uso.

El resultado del proceso es exportado nuevamente en archivos de salida que permiten analizar la cantidad de unidades enviadas, el volumen ocupado, el costo total de operación y el grado de cumplimiento de la demanda proyectada. Se incluye también un análisis detallado del cumplimiento de mínimos por categoría, la eficiencia de los camiones utilizados y la existencia de eventuales conflictos de envíos cruzados.

El modelo fue diseñado para que la ejecución de ambas etapas sea independiente pero encadenada, es decir, primero debe completarse la proyección de demanda y luego puede correrse la optimización logística con los nuevos datos proyectados. Esta estructura modular permite que el proceso sea replicable en distintos períodos de tiempo o ante cambios en los parámetros operativos, adaptándose a las necesidades del retailer de manera flexible y escalable.

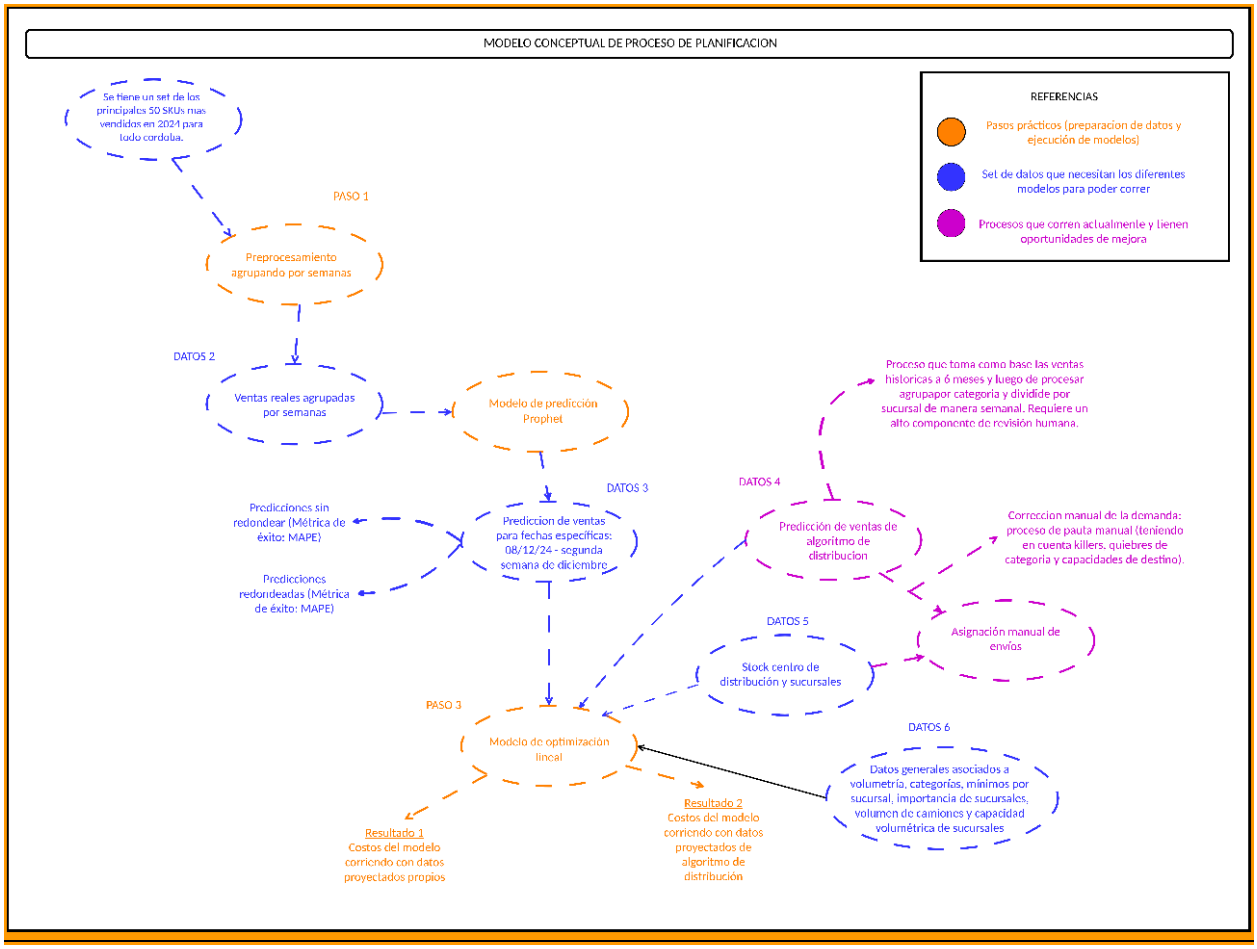

Figura 6: Explicación consolidada de la metodología aplicada al proceso de planeación sugerido.

3.1 Resumen metodológico modelo proyectivo: Modelado de demanda semanal para consumo masivo mediante Prophet.

1. Contexto general y enfoque geográfico

El objetivo específico de este apartado es la estimación de la demanda futura a nivel diario, que será posteriormente utilizada como input en un modelo de optimización orientado a la asignación óptima de productos desde centros de distribución hacia sucursales.

Teniendo en cuenta que el dataset cuenta con una estructura temporal clara, que se desea un modelo por combinación SKU-sucursal, independiente, de fácil mantenimiento y que existen patrones estacionales y eventos externos (como promociones especiales), se optó por implementar **Prophet**, un modelo desarrollado por Facebook, que combina una estructura aditiva flexible, con facilidad de uso, y capacidades integradas para feriados, crecimiento no lineal, y componentes estacionales customizables. Como mencionamos anteriormente el análisis se circunscribe a una selección de cinco sucursales localizadas en la región de Córdoba, identificadas con los números 81, 82, 169, 173 y 176. Esta selección responde a la necesidad de trabajar inicialmente con un subconjunto representativo que permita desarrollar, validar y ajustar la metodología de proyección antes de su escalamiento a otras regiones o unidades comerciales.

2. Exploración inicial y entrenamiento preliminar

Antes de construir el modelo final de proyección sobre las últimas semanas del histórico, específicamente basado en diciembre de 2024, se desarrolló un modelo preliminar de proyección para todo el año 2025. Este primer modelo tuvo un carácter exploratorio y tuvo como objetivo familiarizarse con la estructura de datos, identificar patrones de estacionalidad, definir las variables clave, y evaluar el comportamiento de Prophet como herramienta de modelado de series temporales en el contexto de consumo masivo.

3. Justificación de la elección del modelo Prophet

Dado el objetivo de predecir la demanda futura semanal de productos de consumo masivo en sucursales del interior de Argentina, se evaluaron distintos enfoques posibles para abordar la problemática. Entre ellos se consideraron:

- **Promedios móviles ajustados por estacionalidad**: se trata de una técnica clásica que permite suavizar la serie temporal, eliminando ruido y destacando patrones de estacionalidad y tendencia. Sin embargo, su capacidad predictiva es limitada ante datos con irregularidades, quiebres estructurales o feriados promocionales.
- **Modelos ARIMA/SARIMA**: son modelos estadísticos tradicionales que permiten capturar componentes de autocorrelación y estacionalidad de forma parsimoniosa. Si bien son robustos en series limpias y regulares, presentan dificultades al incorporar efectos externos como feriados, y no escalan bien cuando se necesita entrenar modelos independientes para cientos de combinaciones SKU-sucursal.
- **Modelos de machine learning como XGBoost, LightGBM o redes neuronales (NeuralProphet)**: estos modelos tienen gran capacidad de predicción en contextos con

muchas variables explicativas. Sin embargo, requieren una cantidad importante de ingeniería de variables, son menos transparentes para interpretación directa, y no aprovechan de manera nativa la estructura temporal de las series.

Dado que Prophet está especialmente diseñado para escenarios empresariales con registros temporales irregulares, alta frecuencia y necesidad de interpretabilidad se decidió avanzar por este enfoque.

4. Explicación del concepto del funcionamiento de Prophet

Prophet es un modelo de series temporales desarrollado por el equipo de investigación de Facebook en 2017, orientado principalmente a usuarios de perfil no estadístico que necesitan generar pronósticos robustos, interpretables y ajustables con bajo esfuerzo de parametrización. Su diseño está pensado para capturar estructuras comunes en datos de demanda como tendencias, estacionalidades múltiples y efectos de feriados o eventos especiales. A diferencia de los modelos tradicionales que requieren una alta especialización técnica, Prophet permite la implementación modular de estos componentes mediante una estructura aditiva intuitiva. Esto lo convierte en una herramienta altamente accesible para contextos de negocio, particularmente útil cuando se trabaja con múltiples series independientes (como la demanda por SKU y sucursal), donde se necesita escalabilidad y capacidad de intervención manual.

Cuando decimos que **Prophet es un "modelo aditivo"**, nos referimos a cómo se construye matemáticamente la predicción. Básicamente, **la idea central es que la demanda en una fecha determinada es la suma de varios componentes**:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon(t) \quad (1)$$

Donde:

- **g(t)** = Tendencia → es el crecimiento o decrecimiento general a lo largo del tiempo (puede ser lineal o logístico).
- **s(t)** = Estacionalidad → captura patrones que se repiten en ciclos (por ejemplo, semanal o anual).
- **h(t)** = Efectos de feriados → suma el impacto de días especiales (Hot Sale, Cyber Monday, etc.).
- **ε(t)** = Error aleatorio o ruido → la parte que el modelo no puede explicar.

Es un modelo aditivo dado que todos esos componentes se "suman" entre sí para formar la predicción final. No hay interacción multiplicativa entre ellos.

5. Preparación del modelo: separación en training y test

Una vez concluida la etapa exploratoria, en la que se utilizó información del año 2024 para proyectar la demanda del año 2025, se implementó una nueva estrategia de validación temporal. Esta consistió en identificar, para cada combinación SKU–sucursal, las últimas cuatro semanas con ventas registradas y reservarlas como conjunto de prueba (**test set**), asegurando

así una evaluación más representativa del desempeño del modelo ante escenarios recientes. El resto del histórico fue utilizado como **training set**.

Esto permitió realizar una comparación directa entre valores reales y proyectados sobre semanas recientes que el modelo no vio durante el entrenamiento. La validación se centró en evaluar el rendimiento del modelo en la predicción de semanas específicas, en particular, la segunda semana de diciembre de 2024, dado que de esta manera se lograba tener mayor cantidad de datos históricos de Enero a Noviembre para alimentar el modelo.

6. Transformación semanal y configuración del modelo Prophet

Para poder entrenar Prophet, fue necesario transformar la base de datos original, que contenía registros a nivel de fecha de entrega, en una serie temporal semanal. Tal cual se muestra en **la figura 7** Esto se logró agrupando la información por SKU, número de sucursal y semana calendario, utilizando como fecha representativa el inicio de cada semana (lunes).

```
# AGRUPAR POR SKU + SUCURSAL + SEMANA
ventas_semanales = df.groupby(['cod_articulo', 'Numero_Sucursal', 'Semana'])['cantidad'].sum().reset_index()
ventas_semanales.rename(columns={'Semana': 'ds', 'cantidad': 'y'}, inplace=True)
```

Figura 7: Script que muestra la implementación en Python de la transformación de registros por fecha de entrega en series temporales.

Luego de preparar los datos, el modelo Prophet fue configurado tal cual se muestra en la **figura 8** con los siguientes parámetros clave:

```
        modelo = Prophet(interval_width=0.95,
                         weekly_seasonality=True,
                         daily_seasonality=False,
                         holidays=feriados)
        modelo.fit(grupo)
```

Figura 8: Parámetros asociados a la implementación de Prophet.

- interval_width: parámetro que controla el nivel de confianza para los intervalos de prediccion configurado al 95% para obtener una banda de confianza amplia.
- weekly_seasonality: activado para capturar patrones semanales de comportamiento.
- daily_seasonality: desactivado por no haber evidencia de patrones diarios consistentes.
- holidays: se incluyó un DataFrame de feriados comerciales relevantes como Hot Sale (13 al 15 de mayo) , Electro Fans (15 al 19 de Octubre de 2024) y Cyber Monday (4 al 6 de noviembre de 2024).

La predicción se hizo para las 4 semanas posteriores a las últimas fechas del conjunto de entrenamiento, para coincidir con las semanas separadas en el conjunto de test.

7. Métricas de evaluación del modelo

Para evaluar la precisión del modelo se utilizó como métrica principal el **MAPE** (Mean Absolute Percentage Error). Esta métrica permite analizar la precisión relativa del modelo con respecto a la demanda real observada.

El código del Modelo expuesto en el anexo 1 incluye un apartado específico para el cálculo del MAPE (Mean Absolute Percentage Error) y el error medio, utilizados como métricas principales de evaluación (puede encontrarse en el apartado “#Calculo de Mape y de error medio”). Para esto, se compara el valor real observado en las últimas 4 semanas de cada combinación SKU–

Sucursal con las predicciones generadas por Prophet. Se calcula el error absoluto relativo y luego su promedio multiplicado por 100. Además, se realiza una comparación contra los valores redondeados hacia arriba y contra los límites superior e inferior del intervalo de predicción que entrega Prophet. Estos valores corresponden a los percentiles definidos por el parámetro interval_width, que por defecto es 0.80 o puede configurarse manualmente (por ejemplo, al 95%). El límite inferior (yhat_lower) representa el valor por debajo del cual se espera que se encuentren aproximadamente el 2.5% de las observaciones futuras, mientras que el límite superior (yhat_upper) indica el valor por encima del cual se espera que se encuentren también solo el 2.5% restante. En conjunto, estos límites definen una banda de confianza alrededor de la predicción central (yhat), reflejando la incertidumbre del modelo. Esta sección del código fue la utilizada para exponer en la tesis el desempeño del modelo predictivo.

Se consideraron:

- **MAPE global**: promedio general sobre todas las combinaciones.
- **MAPE por sucursal**: para detectar posibles patrones sistemáticos de subestimación o sobreestimación en determinadas ubicaciones.
- **MAPE contra predicción redondeada**: dado que la optimización posterior se realizará con unidades enteras, se evaluó también el rendimiento del modelo redondeando hacia arriba las predicciones.

Adicionalmente, como ejercicio extra, se excluyeron los outliers con error relativo superior al 100%, dado que pueden distorsionar el promedio global y llevarnos a pensar que el desempeño de todo el modelo es incorrecto (se contabilizó la cantidad de combinaciones excluidas y se analizó el impacto de dicha exclusión sobre el MAPE final)

8. Elección de la semana de predicción

Luego del análisis de volumen y cobertura de SKU por semana, se seleccionó dentro de las ventas semanales del mes de diciembre de 2024 que se muestra en la **Figura 9** la segunda semana de diciembre de 2024 como base de proyección. Esta semana no presentó la mayor cantidad de ventas registradas para ese mes y no esta propensa a estacionalidades propias (por ejemplo, como navidad) que pueden llegar a desvirtuar el análisis.

El conjunto final de datos proyectados, compuesto por SKU, sucursal, cantidad estimada, y bandas de confianza, servirá como input para el modelo de optimización, donde se determinarán las cantidades a distribuir desde los centros de distribución hacia cada destino, considerando restricciones de capacidad, volumen y prioridades de abastecimiento.

Este enfoque integral permite alinear la predicción de demanda con la planificación operativa, garantizando coherencia y escalabilidad en el proceso de toma de decisiones logísticas.

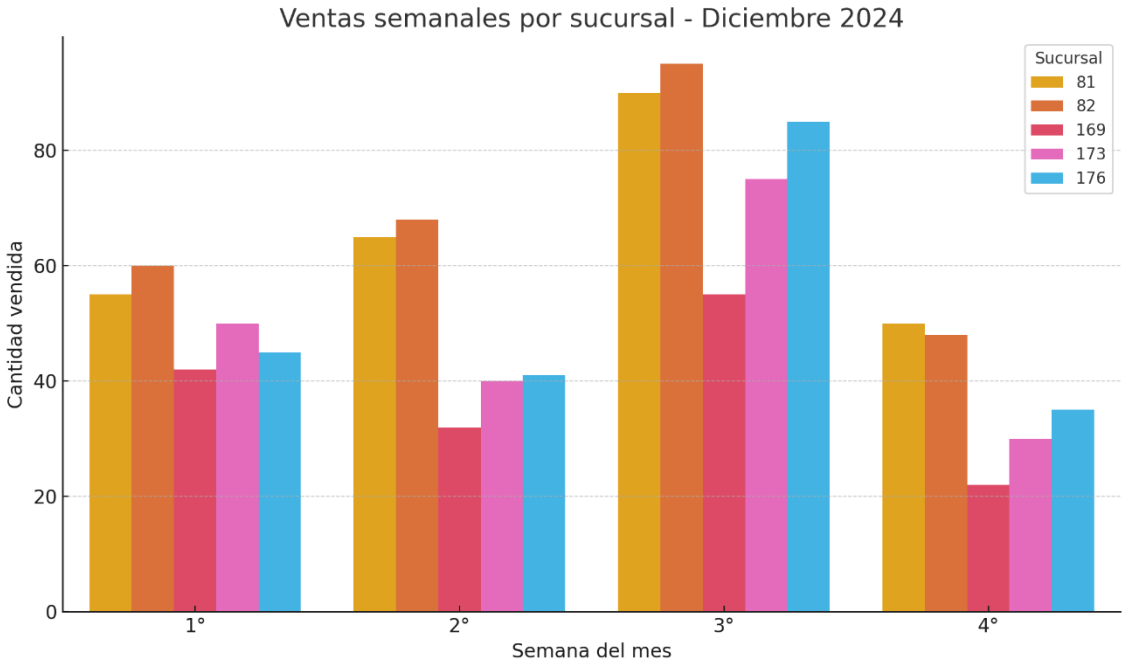

Figura 9: Distribución semanal de ventas por sucursal para diciembre de 2024.

3.2 Resumen metodológico modelo optimización: Implementación de un modelo de programación lineal entera.

Para la segunda parte del trabajo la cual se orienta específicamente en la construcción de un modelo de optimización lineal que permita definir cuáles son las cantidades, orígenes, destinos y costos óptimos para resolver el problema que se planteó al principio del documento se utilizó la librería PuLP de Python. El modelo se formula como un problema de programación lineal entera mixta (MILP), caracterizado por una función objetivo lineal, restricciones lineales, y variables que pueden ser continuas o enteras. Para su resolución, se utilizó el solver CBC (Coin-or Branch and Cut), integrado por defecto en la librería PuLP. CBC es un solucionador open source diseñado específicamente para este tipo de problemas, y aplica técnicas como *branch-and-bound* y *cutting planes* para encontrar soluciones óptimas. PuLP permite definir problemas de optimización en términos matemáticos mediante estructuras legibles y controlables desde código Python, facilitando así la integración con otras librerías y la lectura de datos desde múltiples fuentes (como Excel, bases de datos, APIs, etc.).

El enfoque de trabajo estuvo centrado en entender la lógica de distribución logística en escenarios reales, comenzando con modelos simples y luego incorporando progresivamente restricciones y complejidades relevantes. Este proceso de desarrollo iterativo permitió testear, validar y escalar la función objetivo junto con las restricciones del sistema de forma controlada. Inicialmente, se construyeron modelos con pocas variables (por ejemplo, solo un centro de distribución y dos sucursales) y sin restricciones de capacidad ni prioridades. Luego, se avanzó paso a paso en la inclusión de variables adicionales, como volumen, distancia, capacidad de transporte, insatisfacción de demanda, penalizaciones, transferencias entre sucursales, restricciones de categorías mínimas y contribución marginal.

Esta estrategia permitió aislar errores, entender las interacciones entre variables, y evaluar de manera clara el impacto de cada restricción en el comportamiento del modelo. Finalmente, se llegó a una versión escalable que contempla múltiples sucursales, múltiples SKUs, múltiples orígenes y destinos de stock, y múltiples criterios de optimización.

A continuación, se lista el detalle de las diferentes versiones de código iterativo que se fueron trabajando de manera de poder escalar hasta llegar a la versión 12, la mas completa previa a la corrida de datos productivos:

Evolución del modelo paso a paso

- **Modelo 0 – Optimización básica sin discriminación por sucursal:**

  Este primer modelo funcionó como un MVP. Se planteó una única función objetivo que minimiza el costo total de envío desde el centro de distribución (depósito 111) hacía dos sucursales (81 y 82), sin discriminar ni el stock por sucursal ni las limitaciones logísticas específicas. La demanda se trató de forma agregada por sucursal, sin capacidad de diferenciar cómo se satisface ni desde dónde. La función objetivo se centró en el costo de transporte por volumen enviado, sin restricciones adicionales. Las variables de entrada eran: cantidad demandada por sucursal, volumen de cada SKU y distancia. Fue clave para probar que la sintaxis de PuLP se interpretaba correctamente y que se generaba una solución óptima inicial.

- **Modelo 1 – Inclusión de stock por sucursal y detección de errores en duplicación:**

  En esta versión se incorporó la lógica de stock por sucursal, diferenciando la cantidad disponible en 111 de la ya presente en 81 y 82. Si bien se lograba una mejor aproximación al problema real, el modelo duplicaba ciertas cantidades enviadas, mostrando errores en cómo se modelaban las variables de decisión. La función objetivo se mantenía similar (costos logísticos), pero el foco pasó a estar en corregir la forma en que se calculaba lo que efectivamente se debía enviar. La entrada al modelo incluyó por primera vez el stock inicial por ubicación, y esto abrió paso al desarrollo de restricciones más robustas.

- **Modelo 2 – Corrección de duplicación y cálculo correcto de unidades enviadas:**

  Este modelo resolvió el problema detectado en la versión anterior: se corrigió la duplicación en los envíos y se ajustaron las restricciones para asegurar que no se supere el stock disponible en cada ubicación. La función objetivo siguió siendo la minimización del costo logístico, pero ahora con una estructura más limpia que reflejaba fielmente lo que se estaba enviando desde depósito y lo que ya había en cada sucursal. Se incorporaron validaciones para mantener el stock no negativo y se mejoró la interpretación de los resultados. Como entrada, se incluyeron las variables de stock y demanda por SKU y por sucursal, lo cual permitió separar el modelo de una lógica puramente agregada a una más granular.

- **Modelo 3 – Incorporación del cálculo de volumen y cantidad de viajes:**

En esta iteración se incorporó el concepto de volumen como unidad clave del modelo logístico. La función objetivo se definió para considerar no solo el volumen total transportado, sino también la cantidad de viajes necesarios, calculados en función de la capacidad del camión. Esta fue la primera vez que se comenzó a pensar en términos operativos reales, como cuánto se puede enviar por viaje, y qué costo tiene cada traslado. Se agregaron variables de volumen por SKU y se introdujo la capacidad de carga como restricción, aunque todavía no se modelaba formalmente la cantidad de viajes como variable entera. Esta versión permitió identificar cuellos de botella físicos y ajustar el modelo a la realidad operativa.

- **Modelo 4 – Envíos entre sucursales y prohibición de envíos cruzados:**

  Aquí el modelo dio un salto cualitativo importante al habilitar transferencias entre sucursales (por ejemplo, de la 81 a la 82 o viceversa). Para evitar bucles logísticos ineficientes, se agregaron variables binarias que prohibían envíos cruzados simultáneos de un mismo SKU entre dos sucursales. Esto implicó una expansión tanto en la función objetivo como en las restricciones: ahora el costo logístico debía contemplar rutas múltiples y la penalización de decisiones contradictorias. Las variables de entrada se ampliaron con la posibilidad de mover stock entre nodos del sistema, aumentando la flexibilidad del modelo y preparando el camino para escenarios más complejos con redes logísticas más densas.

- **Modelo 5 – Incorporación de restricción de capacidad en sucursales:**

  Este modelo introdujo por primera vez restricciones de capacidad física de almacenamiento en cada sucursal, en base al volumen total que se esperaba recibir. La función objetivo siguió igual, pero las nuevas restricciones forzaron al modelo a respetar los límites de ocupación por tienda. Fue clave para emular el comportamiento real de centros con espacio limitado. Se calcularon los volúmenes ocupados en cada ubicación considerando el stock inicial, los envíos desde 111 y las transferencias entre sucursales. Esta versión también ayudó a visualizar mejor el uso eficiente del espacio y a generar reportes post-ejecución que ayudaban a validar si cada punto de la red estaba o no operando dentro de sus capacidades.

- **Modelo 6 – Penalización por insatisfacción de demanda y priorización por sucursal:**

  Este modelo introdujo una de las mejoras conceptuales más relevantes: la inclusión de penalizaciones por demanda no satisfecha. Se incorporaron variables de insatisfacción por SKU y sucursal, y su impacto fue ponderado por un coeficiente de penalización. Además, se definió un esquema de prioridad por sucursal, en el que la sucursal 81 era más importante que la 82, aplicando penalizaciones más severas si no se cumplía su demanda. La función objetivo pasó a minimizar no solo costos logísticos, sino también estas penalizaciones. El modelo comenzó a “decidir” entre satisfacer demanda o asumir penalizaciones si el costo de cumplirla era muy alto. Esta lógica introdujo una capa de realismo estratégico al modelo.

- **Modelo 7 – Control de mínimos por categoría para evitar quiebres de stock:**

En esta versión se integraron restricciones que aseguraban que ciertas categorías clave tuvieran cobertura mínima en cada sucursal. Se crearon funciones que agrupaban los SKUs por categoría y se agregaron restricciones que exigían, por ejemplo, que al menos 10 unidades de "Celulares" debían estar disponibles en cada punto de venta. La función objetivo se mantuvo igual, pero el conjunto de restricciones creció significativamente. Esto elevó la cantidad de variables de entrada, ya que ahora cada SKU debía asociarse con una categoría. Fue un paso importante para reflejar lógicas comerciales reales, donde no alcanzar un mix mínimo de productos en una tienda puede generar problemas operativos o de ventas.

- **Modelo 8 – Priorización por contribución marginal en contexto de baja capacidad:**

  Este modelo combinó varias mejoras anteriores, pero agregó una priorización basada en contribución marginal por SKU. Cada SKU fue ponderado no sólo por su categoría sino por el valor económico que aportaba, y eso se incorporó en la penalización por insatisfacción. A la vez, se mantuvieron restricciones de volumen y mínimos por categoría. El escenario simulado planteaba una limitación fuerte en la capacidad de las sucursales, por lo que el modelo debía "elegir" qué enviar y qué dejar afuera. La función objetivo optimizaba para cubrir la mayor cantidad de demanda rentable posible, dado un presupuesto de espacio. Esta lógica de trade-off económico entre producto y restricción fue clave para escalar.

- **Modelo 9 – Escalamiento general con estructura modular y multipropósito:**

  Este modelo se implementó con estructura modular, capaz de escalar a múltiples sucursales y SKUs, y contemplar la mayoría de las restricciones desarrolladas anteriormente: volumen, capacidad, penalización por insatisfacción, prioridad por sucursal, transferencias, y mínimos por categoría. También se agregaron variables binarias para evitar envíos cruzados de un mismo SKU, y se diseñaron comprobaciones exhaustivas post-ejecución (viajes, ocupación, cumplimiento de mínimos, etc.). La función objetivo ya estaba completamente optimizada: logística + penalización estratégica.

- **Modelo 10 – Consola de escenarios, mínimos por SKU, rutas hermanadas y análisis de eficiencia logística:**

  Esta versión avanzó sobre el Modelo 10 con tres mejoras sustanciales. En primer lugar, incorporó una consola de selección de escenarios (input()) que permite elegir variantes predefinidas del modelo (por ejemplo, capacidad insuficiente, sin stock, o sin posibilidad de cobertura). Esto facilitó la exploración de distintos contextos logísticos sin modificar manualmente el código. En segundo lugar, se agregaron restricciones de mínimos por SKU en cada sucursal, además de los mínimos por categoría. Esto permitió garantizar la presencia de productos clave, especialmente SKUs killer, en ubicaciones estratégicas.

  La tercera gran mejora fue la implementación de un sistema de rutas "hermanadas", donde se validan explícitamente las combinaciones permitidas entre depósito y sucursales, y entre sucursales entre sí. Esto se gestionó mediante las estructuras

transferencias_permitidas y distancia, y afectó tanto la definición de variables como las restricciones.

Además, se añadió un bloque de análisis post-ejecución que compara la cantidad de viajes del modelo real versus los viajes teóricamente óptimos (consolidando volumen). Esto habilita evaluar la eficiencia en el uso de camiones y detectar subutilización. Por último, se enriqueció el análisis de causas de insatisfacción (por costos, disponibilidad o priorización de otras sucursales), y se detalló el cálculo de volumen ocupado por tienda con validación visual. La función objetivo mantuvo la estructura de minimizar costos logísticos más penalizaciones, pero el modelo creció en granularidad, robustez y explicabilidad.

En resumen, las mejoras que se implementaron con respecto al modelo 9 fueron:

- Consola de selección de escenarios (input()).
- Restricciones de mínimos por SKU (además de por categoría).
- Control de rutas permitidas mediante transferencias_permitidas
- Comparación entre viajes reales y óptimos por volumen total.
- Análisis post-modelo más detallado (stock final, causas de insatisfacción, camiones subutilizados).

- **Modelo 11 – Optimización logística con consolidación de carga, prioridades estratégicas y cobertura mínima de mix:**

Este modelo a diferencia de versiones anteriores incorpora una visión integral del sistema, considerando no solo el cumplimiento de la demanda, sino también las restricciones operativas y estratégicas que afectan a la red de distribución.

Entre sus mejoras más destacadas se encuentra la inclusión de una lógica de consolidación de envíos mediante la variable de decisión de cantidad de camiones por ruta, permitiendo agrupar entregas y controlar la eficiencia logística con criterios de carga mínima y capacidad máxima por camión. Es una mirada que soluciona o acerca el modelo definitivamente a una expresión más real del problema operativo dado que contempla una parte variable del costo logístico por kilómetro recorrido pero que también activa costos logísticos fijos cada vez que se activa la necesidad del uso de un camión. Además, se incorporan penalizaciones ponderadas por insatisfacción, ajustadas según la prioridad estratégica de cada sucursal. Todo el modelo es altamente parametrizable, lo que permite adaptarse a diferentes escenarios mediante una consola interactiva de selección de casos.

Este modelo sienta las bases para su posterior implementación en entornos productivos con datos reales y escalamiento a múltiples sucursales y regiones logísticas.

A continuación se listan las mejoras específicas de este modelo:

-Agrupación de envíos mediante variable de cantidad de camiones por ruta.
-Restricciones de carga mínima (80%) y carga máxima por camión.

-Penalización por insatisfacción de demanda ponderada por prioridad de sucursal.
-Chequeo post-ejecución de eficiencia de volumen transportado versus capacidad contratada.
-Verificación del cumplimiento de todas las restricciones clave (capacidad, stock, mínimos,cobertura).

Lamentablemente el modelo, si bien mucho más robusto que los anteriores, terminaba teniendo fallas en la implementación de datos productivos los que nos llevaron a la necesidad de tener que trabajar en una revisión de código extenso que vería como resultado el modelo 12. Estos errores se detallan en el **anexo 3** donde puede verse claramente qué resultados nos llevaron a la necesidad de hacer los cambios en el código que a su vez pueden verse en el **anexo 4**.

Modelo 12 –Planificación Logística Integral (PLI)

El Modelo PLI constituye la versión más avanzada y robusta desarrollada en el marco de este trabajo para la planificación logística de distribución de mercadería. A diferencia de sus predecesores, integra de forma sistemática tanto las restricciones operativas como las condiciones estratégicas de cobertura y eficiencia, contemplando los límites físicos del sistema y la lógica económica de los envíos. Su diseño prioriza el cumplimiento de la demanda minimizando los desvíos logísticos, a través de una correcta activación de camiones por ruta, cada uno realizando un único trayecto sin encadenamientos. Además, incorpora mecanismos para prevenir inconsistencias operativas, como envíos cruzados entre sucursales o distribución sin disponibilidad real de stock, lo que garantiza soluciones coherentes y aplicables a contextos reales.

El modelo es completamente parametrizable, lo que permite su adaptación a distintos escenarios logísticos sin necesidad de modificar su estructura base. Una vez ejecutado, se dispone de un módulo de validación exhaustivo que permite auditar el comportamiento del sistema desde múltiples dimensiones. Este módulo analiza los costos desagregados (fijos y variables), el cumplimiento de la demanda por SKU y sucursal, la eficiencia del uso de camiones, el cumplimiento de mínimos por categoría, las restricciones de capacidad y la integridad del stock en cada nodo. También se verifica la existencia de inconsistencias logísticas, como envíos cruzados innecesarios. Todos los resultados son exportados a un archivo Excel con reportes organizados por pestañas temáticas.

Las mejoras clave incluyen: activación controlada de camiones por volumen, validación de carga mínima, conservación estricta del stock, verificación cruzada de restricciones operativas y exportación automatizada de resultados. A continuación, se presenta la formulación matemática completa que sustenta este modelo:

1. Variables de Decisión

Se definen las siguientes variables de decisión para modelar la distribución de productos:

- $x^{(CD)}_{(sku,suc)} \in \mathbb{Z} \geq 0$: Cantidad de unidades del SKU enviadas desde el CD (depósito) hacia la sucursal.
- $x_{(sku,i,j)} \in \mathbb{Z} \geq 0$: Cantidad de unidades del SKU transferidas entre sucursales desde la sucursal i a la j.
- $I_{\{sku,suc\}} \in \mathbb{Z} \geq 0$: Unidades de demanda insatisfecha del SKU en la sucursal.
- $B_{(sku,i,j)} \in \{0,1\}$: Variable binaria que indica si hubo transferencia del SKU entre la sucursal i y j (sirve para evitar cruces redundantes).
- $C_{(i,j)} \in \mathbb{Z} \geq 0$: Cantidad de camiones activados para transportar stock desde el origen i al destino j.
- $V^{(i,j)}_{(enviado)} \in \mathbb{R} \geq 0$: Volumen Enviado en una ruta especifica desde el origen i al destino j.

2. Parámetros del Modelo

Los siguientes parámetros alimentan el modelo:

- $D_{(sku,suc)} \in \mathbb{Z} \geq 0$: Demanda del SKU en la sucursal.
- $V_{(sku)} \in \mathbb{R} \geq 0$: Volumen del SKU.
- $volumen_{sku}.x_{(sku,i,j)}$: Volumen del SKU en un trayecto i,j.
- $S_{(sku,suc)} \in \mathbb{Z} \geq 0$: Stock inicial disponible del SKU en la sucursal.
- $CM_{(sku)} \in \mathbb{Z} \geq 0$: Contribución marginal del SKU al negocio (prioridad de satisfacción).
- $W_{(suc)} \in \mathbb{Z} \geq 0$: Peso o importancia relativa de la sucursal.
- $K \in \mathbb{R} \geq 0$: Capacidad de cada camión (en volumen).
- $A \in \mathbb{R} \geq 0$: Porcentaje mínimo de ocupación requerida para que se active un camión.
- $d_{(i,j)} \in \mathbb{R} \geq 0$: Distancia entre el punto de origen i y el destino j.

- $c_f \in \mathbb{R} \geq 0$: Costo fijo asociado a cada movimiento activado.

- $c_v \in \mathbb{R} \geq 0$: Costo variable por kilometro recorrido.

3. Función Objetivo

El objetivo del modelo es **minimizar el costo total de distribución**, considerando tanto los costos logísticos como las penalizaciones por demanda insatisfecha. En este contexto, el parámetro $\mathrm{CM}_{(sku)}$, que representa la **contribución marginal del SKU al negocio**, no está actualmente parametrizado por la empresa. Su estimación requiere un análisis específico del área de costos, ya que no solo depende de la diferencia entre el precio de venta y el costo de adquisición o fabricación, sino también del impacto que genera no contar con el producto en el momento en que el cliente lo necesita. Este impacto puede traducirse en la pérdida de futuras ventas o en el abandono del canal, afectando directamente la rentabilidad de la compañía.

Dado que no se dispone de una métrica formal para esta variable, y con el fin de orientar al modelo hacia la satisfacción de la demanda, se optó por asignar un **valor arbitrariamente elevado a** $\mathrm{CM}_{(sku)}$ Este valor está detallado en los anexos y actúa como un mecanismo que penaliza fuertemente la insatisfacción de demanda, forzando al modelo a priorizar su cobertura siempre que sea posible.

En cuanto al parámetro $\mathrm{W}_{(suc)}$, que representa el **peso relativo o importancia estratégica de cada sucursal**, si bien es completamente parametrizable, en este trabajo se lo considera constante para todas las sucursales. Esto se debe a que su cálculo dependería de múltiples factores, como el volumen de ventas, el posicionamiento geográfico o el impacto que puede tener la falta de disponibilidad de un producto en términos de pérdida de clientes o imagen comercial. Al igual que $\mathrm{CM}_{(sku)}$, su definición precisa requeriría una evaluación complementaria desde áreas como Comercial o Inteligencia Empresarial.

A continuación, se presenta la **función objetivo**, compuesta por dos términos: el primero, correspondiente al **costo logístico asociado al uso de camiones**, y el segundo, al **costo de insatisfacción de demanda** o **lucro cesante**. Este segundo componente está ponderado por la contribución marginal del SKU y el peso relativo de la sucursal, reflejando así la prioridad que debe asignarse a la cobertura de ciertos productos en determinados puntos de venta.

$$\min \left[\sum_{(i,j) \in T} \mathrm{C}_{\{i,j\}} . \left(c_f + d_{(i,j)} \cdot c_v \right) + \sum_{(SKU,s) \in D} I_{(\mathrm{SKU,s})} . \mathrm{CM}_{(sku)} . \mathrm{W}_{(suc)} \right] (2)$$

- $d_{(i,j)}$: Distancia entre el punto de origen i y el destino j.
- c_v: Costo variable por kilometro recorrido.
- c_f: Costo fijo asociado a cada movimiento activado.
- $\mathrm{CM}_{(sku)}$: Contribución marginal del SKU al negocio (prioridad de satisfacción).
- $\mathrm{W}_{(\mathrm{suc})}$: Peso o importancia relativa de la sucursal.
- $\mathrm{C}_{(i,j)}$: Cantidad de camiones activados para transportar stock desde el origen i al destino j.

- T: Conjunto de rutas posibles (origen, destino).
- D: Conjunto de combinaciones SKU-suc.

Como punto extra a tratar teniendo en cuenta la definición de la función objetivo y particularmente la demanda insatisfecha, resulta relevante considerar que la insatisfacción de demanda no necesariamente implica una pérdida definitiva. Existen alternativas logísticas y comerciales que pueden mitigar parcialmente su impacto. Una de ellas es la sustitución de productos, es decir, ofrecer al cliente un artículo similar —en características o funcionalidad— que cumpla el mismo propósito, minimizando así la pérdida de venta. Otra posibilidad es la postergación de la compra, ya sea mediante mecanismos como listas de espera, reservas con fecha futura o entrega diferida. Asimismo, algunos clientes pueden optar por satisfacer su necesidad a través de otros canales propios de la empresa (como la tienda online o puntos de venta cercanos), lo que implica una reubicación interna de la demanda antes que una pérdida externa. Incluir estas opciones en el modelo requeriría parametrizar elasticidades entre productos, niveles de tolerancia a la espera por parte del consumidor y capacidades de redireccionamiento entre sucursales, elementos que podrían abordarse como extensiones futuras del sistema desarrollado.

4. Restricciones del Modelo

4.1. Capacidad de Camiones

El modelo contempla que el volumen total de productos enviados en una ruta determinada no puede superar la capacidad total de los camiones activados para esa ruta. Esta condición se impone para cada combinación de origen y destino permitida.

$$\sum_{sku \in SKUs} volumen_{sku} . \mathrm{x}_{(sku,i,j)} \leq K \, . C_{(i,j)} \quad \forall (i,j) \in Transferencias\ permitidas \quad (3)$$

- $\mathrm{C}_{(i,j)}$: Cantidad de camiones activados entre i y j.
- K: Capacidad de cada camión (en volumen).
- $volumen_{sku}$: volumen unitario del SKU.
- $. x_{(sku,i,j)}$: Cantidad de unidades enviadas desde i hacia j.

Adicionalmente, si un camión es activado, debe transportar al menos un valor mínimo A (entre 0 y 1) de su capacidad, salvo que se haya definido A=0.

$$\text{Si A} > 0 \Rightarrow volumen_{sku} . x_{(sku,i,j)} \geq \sum_{(i,j)} \mathrm{C}_{(\mathrm{i,j})} \cdot \mathrm{K\,A} \quad \forall (i,j)\ Transferencias\ permitidas \qquad (4)$$

- A $\in (0,1)$: fracción mínimo de ocupación requerida para que se active un camión.

4.2. Activación de Camiones

Si existe volumen en una ruta, se debe activar al menos un camión, la expresión plantea que :

$$\mathrm{C}_{(\mathrm{i,j})} \geq \frac{\mathrm{V}^{(\mathrm{i,j})}_{(enviado)}}{(\mathrm{K})} \qquad (5)$$

- K: Capacidad de cada camión (en volumen).
- $C_{(i,j)}$: Cantidad de camiones activados para transportar stock desde el origen i al destino j.
- $V_{(\text{enviado})}^{(i,j)}$: Volumen Enviado en una ruta especifica desde el origen i al destino j.

4.3. Cruces Redundantes entre Sucursales

Evita envíos cruzados del mismo SKU entre dos sucursales. Si se envía de i a j, no se puede enviar simultáneamente de j a i:

$$B_{(\text{sku,i,j})} + B_{(sku,j,i)} \leq 1 \quad (6)$$

$$x_{(\text{sku,i,j})} \leq M \cdot B_{(sku,i,j)} \quad (7)$$

$$x_{(sku,i,j)} \leq M \cdot B_{(sku,j,i)} \quad (8)$$

- $x_{(sku,i,j)}$: Cantidad de unidades del SKU transferidas entre sucursales desde la sucursal i a la j.
- $B_{(sku,i,j)}$: Variable binaria que indica si hubo transferencia del SKU entre la sucursal i y j (sirve para evitar cruces redundantes).
- M: Valor extremadamente grande para que se cumpla la condición.

4.4. Cumplimiento de Demanda

Asegura que la demanda sea cubierta con stock inicial, entradas y permite insatisfacción si no se llega a cubrir:

$$S_{(\text{sku,suc})} + x_{(\text{sku,suc})}^{(\text{CD})} + \sum_{(\text{i}\neq\text{suc})} x_{(\text{sku,i,suc})} - \sum_{(\text{j}\neq\text{suc})} x_{(sku,suc,j)} + I_{(\text{sku,suc})} \geq D_{(sku,suc)} \quad (9)$$

- $S_{(sku,suc)}$: Stock inicial disponible del SKU en la sucursal.
- $x_{(\text{sku,suc})}^{(\text{CD})}$: Cantidad de unidades del SKU enviadas desde el cd a la sucursal.
- $x_{(sku,i,suc)}$: Cantidad de unidades del SKU transferidas desde la sucursal i a la j.
- $I_{(sku,suc)}$: Unidades de demanda instatisfecha del SKU en la sucursal.
- $D_{(\text{sku,suc})}$: Demanda del SKU en la sucursal.

4.5. Capacidad de Almacenamiento en Sucursales

El volumen total almacenado en cada sucursal (stock inicial + entradas - salidas) no debe exceder la capacidad física:

$$\sum_{(sku)} V_{(sku)} \cdot \left[S_{(sku,suc)} + x_{(\text{sku,suc})}^{(\text{CD})} + x_{(sku,i,suc)} - x_{(sku,j,suc)}\right] \leq \text{cap}_{(suc)} \quad (10)$$

- $S_{(sku,suc)}$: Stock inicial disponible del SKU en la sucursal.
- $x^{(CD)}_{(sku,suc)}$: Cantidad de unidades transferidas desde el CD a la sucursal.
- $x_{(sku,i,suc)}$: Cantidad de unidades transferidas desde la sucursal i a la sucursal j (representando entradas a la sucursal J).
- $x_{(sku,j,suc)}$: Cantidad de unidades transferidas de la sucursal j a la sucursal i (representando salidas de la sucursal j).

4.6. Mínimos por SKU en Sucursales

Si se exige un mínimo de unidades por SKU en una sucursal, se debe cumplir con esa cantidad:

$$S_{(sku,suc)} + x^{(CD)}_{(sku,suc)} + x_{(sku,i,suc)} - x_{(sku,j,suc)} \geq M_{(sku,suc)} \qquad (11)$$

- $M_{(sku,suc)}$: Parámetro que indica la **cantidad mínima requerida** del SKU en la sucursal para considerar cumplida la restricción.

4.7. Mínimos por Categoría en Sucursales

La suma de todos los SKUs de una categoría debe alcanzar el mínimo estipulado para cada sucursal:

$$\sum_{(sku \in c)} \left[S_{(sku,suc)} + x^{(CD)}_{(sku,suc)} + x_{(sku,i,suc)} - x_{(sku,j,suc)} \right] \geq J_{(cat,suc)} \qquad (12)$$

- $J_{(cat,suc)}$: Cantidad exigida mínima de unidades por categoría en cada sucursal.
- $x^{(CD)}_{(sku,suc)}$: Cantidad de unidades transferidas desde el CD a la sucursal.
- $x_{(sku,i,suc)}$: Cantidad de unidades transferidas desde la sucursal i a la sucursal j (representando entradas a la sucursal J).
- $x_{(sku,j,suc)}$: Cantidad de unidades transferidas de la sucursal j a la sucursal i (representando salidas de la sucursal j)

4.8. Conservación de Stock

El sistema no puede crear unidades. El total inicial (CD + sucursales) debe ser mayor o igual a lo enviado:

$$S_{(total\ inicial)} \geq \sum_{(i,j)} x_{(sku,i,j)} + \sum_{(cd,suc)} x^{(CD)}_{(sku,suc)} \qquad (13)$$

- $x_{(sku,i,j)}$: Cantidad de unidades del SKU transferidas entre sucursales desde la sucursal i a la j.
- $x_{(\text{sku,suc})}^{(\text{CD})}$: Cantidad de unidades del SKU transferidas desde el cd a la sucursal.
- $S_{(total\ inicial)}$: Stock inicial del SKU a nivel general sistema.

4. RESULTADOS

4.1 PRIMERA PARTE: COMPARACIÓN ENTRE EL OUTPUT DEL MODELO PREDICTIVO PROPHET Y EL ALGORITMO DE DISTRIBUCIÓN QUE UTILIZA ACTUALMENTE LA EMPRESA.

El modelo de predicción de demanda semanal desarrollado mediante Prophet fue evaluado a través de un conjunto de validación compuesto por las últimas cuatro semanas disponibles de datos históricos. Esta validación se llevó adelante tanto para evaluar la capacidad de predicción respecto a los valores reales observados, como para comparar la performance del modelo frente al sistema de predicción de demanda actualmente utilizado (el algoritmo de distribución). Se cuenta desde ya con el detalle de los valores reales que se enviaron a las sucursales 81,82,169,173 y 176 los cuales fueron separados antes de entrenar el modelo.

La estructura del modelo incluyó el ajuste de una serie semanal por combinación SKU-sucursal, incorporando estacionalidad semanal, efectos de feriados relevantes (Hot Sale, Electro Fans y Cyber Monday), y estableciendo bandas de confianza al 95%. El input de ventas fue agregado a nivel semanal y el target fue la cantidad de productos entregados por semana en cada sucursal objetivo de Córdoba.

El primer eje de validación fue el cálculo del **MAPE (Mean Absolute Percentage Error)** de la predicción respecto a los valores reales. Para ello se construyeron dos variantes: una sobre la predicción cruda (**tabla 4**) y otra sobre la predicción redondeada (**tabla 3**) hacia arriba en valores enteros, considerando que la distribución logística opera en unidades enteras. Además, se realizo otro análisis paralelo en donde se eliminaron outliers con errores superiores al 100% para entender cuál es el comportamiento de la predicción solamente dejando por fuera del análisis estos resultados específicos, aunque es importante aclarar que estos valores en el modelo final se encuentran contemplados. Estos fueron los resultados de la comparación:

Mape por Sucursal - Prediccion redondeada				
Numero de Sucursal	MAPE (%) - Redondeado (Incluye Outliers)	MAPE (%) - Redondeado (Outliers filtrados)	Canditidad de registrosExluidos (Redondeados)	% Registros excluidos (Redondeados)
81	105,11	71,92	16	13,68
82	85,38	63,71	11	9,73
169	73,1	60,56	8	9,41
173	71,8	61,29	5	5,88
176	74,72	67,18	3	2,83

Tabla 2: Resultados filtrados para predicciones redondeadas – MAPE.

Mape por Sucursal - Prediccion original				
Numero de Sucursal	MAPE (%) - Redondeado (Incluye Outliers)	MAPE (%) - Redondeado (Outliers filtrados)	Canditidad de registrosExluidos (Redondeados)	% Registros excluidos (Redondeados)
81	106,4	81,98	16	13,68
82	86,11	70,21	11	9,73
169	86,4	79,06	8	9,41
173	78,88	71,47	5	5,88
176	85,83	80,59	3	2,83

Tabla 3: Resultados filtrados para predicciones Prophet – MAPE.

Las tablas generadas mostraron los valores de MAPE promedio por sucursal, tanto considerando todos los datos como filtrando aquellos con errores extremos. Se observó que el MAPE promedio mejoraba al excluir outliers, reflejando una predicción más precisa en la mayoría de las sucursales siendo la cantidad de registros filtrados como se puede ver a nivel de tabla 2 registros para sucursal 81, 11 registros para sucursal 82, 8 registro para sucursal 169, 5 registros para sucursal 173 y 3 registros para sucursal 176. Asimismo, el redondeo de las predicciones

resultó, en la mayoría de los casos, en un leve descenso adicional del error relativo, hecho que refuerza la pertinencia de ajustar las proyecciones a números enteros.

En paralelo, se comparó el desempeño del modelo Prophet contra el sistema actual de proyección utilizado para alimentar el modelo de optimización logística para una fecha específica del mes de diciembre de 2024 (la fecha exacta es el 08/12/2024). Si bien el sistema vigente se basa en reglas fijas o promedios históricos simples, Prophet demostró una mejor capacidad de capturar dinámicas estacionales y responder a eventos excepcionales como los feriados comerciales, aspectos que el modelo anterior no contemplaba explícitamente.

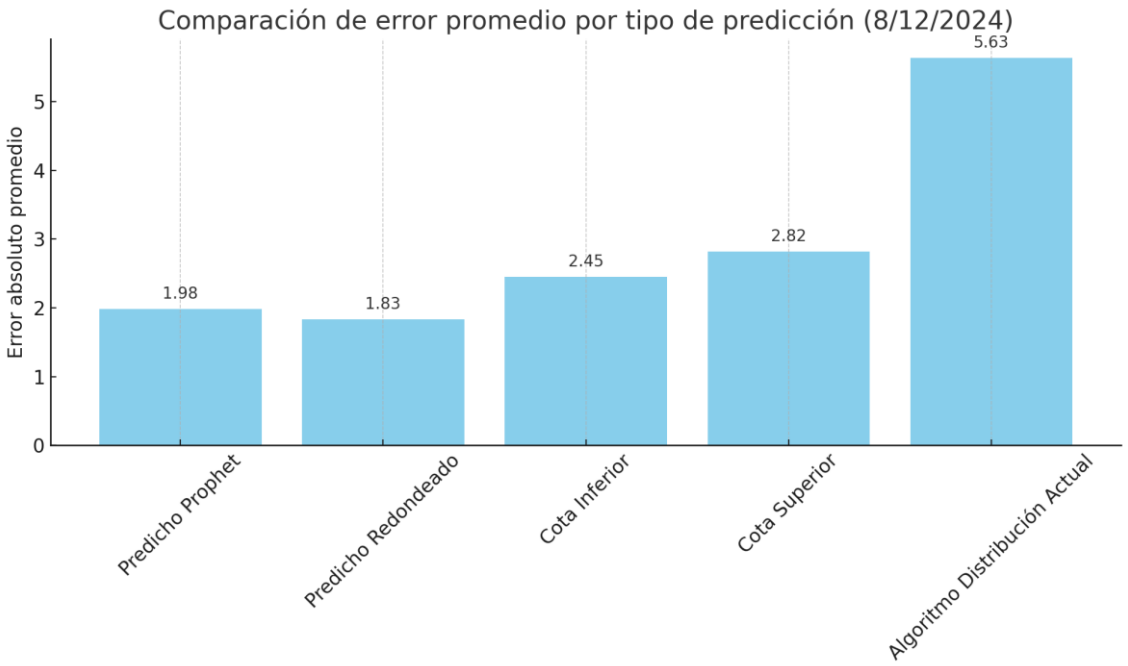

Figura 10: Error promedio por tipo de predicción.

En **la figura 10** se compara el error absoluto promedio de distintos métodos de predicción frente a los valores reales observados. Cada barra representa el error promedio para un modelo: Prophet, Prophet redondeado a números enteros, cota inferior, cota superior y el algoritmo de distribución vigente. El error absoluto promedio se calcula como la diferencia entre el valor real y el valor predicho, en términos absolutos, promediado sobre todas las combinaciones SKU-sucursal. Cuanto menor es el valor, mayor es la precisión del modelo. El análisis permite visualizar cuál de los enfoques logró una mejor aproximación a la demanda real. En conclusión, se puede ver claramente que de todos los modelos, Prophet redondeado fue el que mejor se adaptó a los datos de producción.

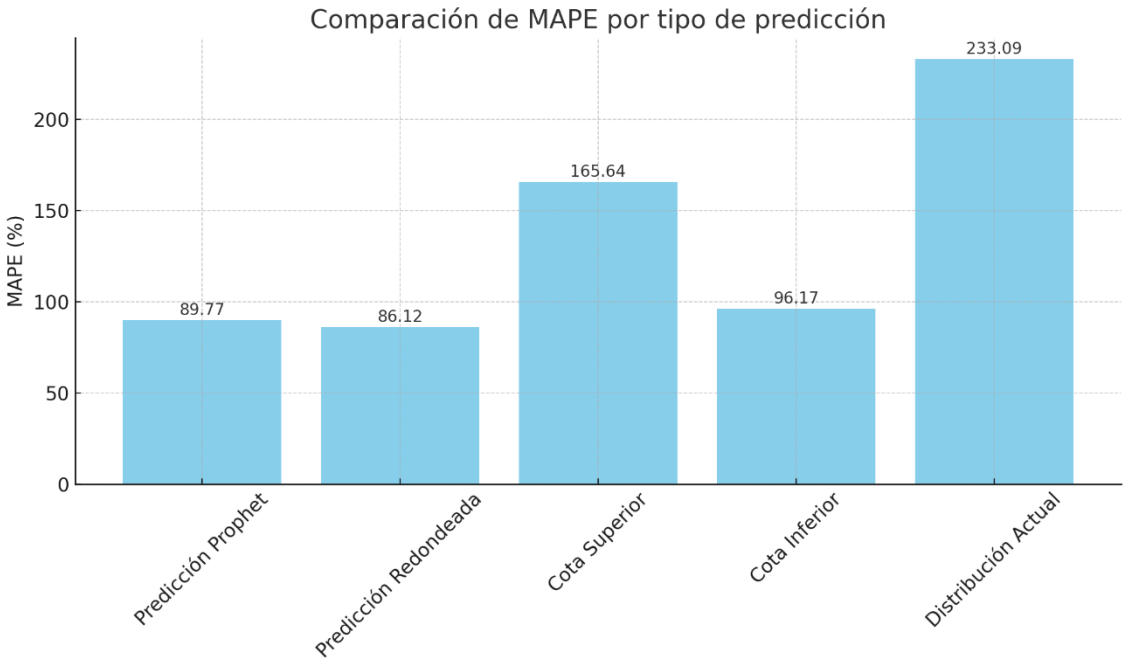

Figura 11: MAPE por tipo de predicción.

Por otro lado, si volvemos a analizar los modelos basándonos en la métrica original MAPE, se observa que la predicción redondeada del modelo Prophet presenta el menor error porcentual promedio frente a los valores reales, superando tanto a la predicción directa como a las cotas superior e inferior, y al algoritmo de distribución actual. Esta diferencia sostenida en el desempeño respalda la decisión de adoptar, para fines operativos, el valor predicho por Prophet redondeado hacia arriba como estimación principal de demanda, combinando precisión relativa con viabilidad práctica en su aplicación logística.

Si bien el modelo Prophet redondeado demostró ser el más eficaz en términos de aproximación a los valores reales observados, es necesario advertir que una mejora en la precisión del pronóstico no implica automáticamente una reducción del riesgo operacional. Al reducir el margen de sobreestimación de la demanda que caracterizaba al algoritmo anterior, el sistema tiende a trabajar con niveles de inventario más ajustados. Esta mejora, aunque deseable desde la eficiencia logística, incrementa el riesgo de quiebre de stock ante cualquier desvío no anticipado. En efecto, al disminuir los colchones de seguridad implícitos en los excesos de predicción previos, la operación se vuelve más sensible a eventos no modelizados o a errores residuales. Si bien este riesgo aún no fue cuantificado en términos financieros dentro del presente trabajo, se considera como línea futura de análisis evaluar el trade-off entre eficiencia predictiva y robustez operativa, de modo de poder ajustar el modelo en función del nivel de servicio deseado por la organización.

Estos resultados constituyen un paso fundamental hacia un esquema de proyección de demanda más robusto, dinámico y escalable, que sirva como input sólido para la optimización de la red logística que tiene actualmente el retailer y más específicamente evite tanta interacción manual que puede llevar a errores involuntarios o a soluciones alejadas de lo óptimo.

4.2 Segunda parte: Resultados asociados a Modelo de Optimización

El script asociado al modelo 12 viene integrado directamente con la exportación de un archivo CSV que contiene la información correspondiente a diferentes aspectos que son super importante de abordar a nivel de output del proceso de distribución logística. A continuación, se lista lo que compone a cada diferente solapa y también la información que contiene:

1. Resumen por SKU y Sucursal:

Esta solapa tal cual se detalla en la **tabla 4**, detalla para cada combinación SKU–Sucursal:

- **Stock inicial**: unidades disponibles antes de los envíos.
- **Entradas desde CD y sucursales**: volumen recibido.
- **Salidas a otras sucursales**: si esa sucursal funciona también como redistribuidora.
- **Total disponible final**: unidades efectivas que quedan disponibles.
- **Demanda esperada y mínima**: valores de entrada definidos en el modelo.
- **Checks de cumplimiento**: columnas que indican si se cumple o no con los mínimos por SKU.

Esta comprobación busca validar si las unidades asignadas cubren la demanda esperada y las condiciones mínimas impuestas.

Sucursal	SKU	Stock inicial	Entradas desde CD	Entradas desde sucursales	Salidas hacia sucursales	Total disponible final	Demanda	Mínimo requerido	¿Cumple mínimo?	¿Demanda satisfecha?	Unidades no satisfechas
81	12982	0	0	2	0	2	0	2	☑	☑	0
81	13040	0	0	4	0	4	4	0	☑	☑	0
81	13257	0	0	3	0	3	3	2	☑	☑	0
81	13469	0	0	1	0	1	1	1	☑	☑	0
81	160879	0	0	6	0	6	1	6	☑	☑	0
81	250093	1	0	2	0	3	1	3	☑	☑	0
81	502327	2	0	0	0	2	1	2	☑	☑	0
81	781934	0	0	4	0	4	2	4	☑	☑	0
81	782297	0	0	16	0	16	5	16	☑	☑	0
81	782311	1	0	7	0	8	8	2	☑	☑	0
81	782327	2	0	12	0	14	3	14	☑	☑	0
81	20632	0	0	0	0	0	0	0	☑	☑	0
81	502347	0	0	3	0	3	0	3	☑	☑	0
81	502579	0	0	0	0	0	0	0	☑	☑	0
81	782349	0	0	3	0	3	0	3	☑	☑	0
81	13696	0	0	0	0	0	0	0	☑	☑	0
81	94742	0	0	0	0	0	0	0	☑	☑	0
81	502530	0	0	0	0	0	0	0	☑	☑	0
81	782335	0	0	5	0	5	0	5	☑	☑	0
81	13629	0	0	0	0	0	0	0	☑	☑	0
81	502532	0	0	0	0	0	0	0	☑	☑	0

Tabla 4: Resumen por SKU y sucursal.

2. Resumen de Camiones y Costos por Ruta:

Muestra, para cada ruta origen–destino, tal cual se puede ver en la **tabla 5**:

- Cantidad de camiones utilizados.
- Costo fijo: en función de los camiones contratados.
- Costo variable: proporcional a los kilómetros recorridos y volumen transportado.
- Costo total por ruta.

Esta comprobación permite identificar rutas logísticamente costosas o subutilizadas.

ejemplo:

Ruta	Camiones utilizados	Volumen total transportado (m³)	Capacidad total contratada (m³)	Eficiencia de llenado (%)	¿Cumple mínimo requerido?
111 → 169	1	9,2878	10	92,88	☑
111 → 173	1	9,9838	10	99,84	☑
111 → 176	1	9,9688	10	99,69	☑
169 → 81	1	7,0918	10	70,92	☑
176 → 82	1	7,9744	10	79,74	☑

Tabla 5 : Resumen camiones y costos por ruta.

3. Verificación de Mínimos por Categoría:

Para cada sucursal y categoría de producto, tal cual se muestra en la **tabla 6**:

- Cantidad total recibida en esa categoría.
- Mínimo requerido (definido en la parametrización).
- Check de cumplimiento.

Esta comprobación es útil para verificar que las categorías clave estén representadas en cada punto de venta.

Sucursal	Categoría	Total recibido/utilizable	Mínimo requerido	¿Cumple mínimo?
81	R-ACONDICIONADORES(R2)	0	0	☑
81	CELULARES SIN LINEA	65	0	☑
81	CAFE E INFUSIONES	1	0	☑
81	PEQUEÑOS CUIDADO PERSONAL Mujer	9	0	☑
81	PREPARACION DE ALIMENTOS	0	0	☑
81	R-AGUA CALIENTE(R9)	0	0	☑
81	TV	5	0	☑
81	R-HELADERAS(R16)	6	0	☑
81	PEQUEÑOS HOGAR	7	0	☑
82	R-ACONDICIONADORES(R2)	5	0	☑
82	CELULARES SIN LINEA	27	0	☑
82	CAFE E INFUSIONES	10	0	☑
82	PEQUEÑOS CUIDADO PERSONAL Mujer	22	0	☑
82	PREPARACION DE ALIMENTOS	0	0	☑
82	R-AGUA CALIENTE(R9)	0	0	☑
82	TV	29	0	☑

Tabla 6: Verificación de mínimos por categoría.

4. Verificación de Volumen y Capacidad

Como puede verse en detalle en la **tabla 7**, la tabla contiene:

- Volumen inicial: lo disponible antes de la distribución.
- Volumen final: lo que queda después de la redistribución.
- Capacidad instalada de cada sucursal y CD.
- Validación de exceso o negativo: si se supera la capacidad o si el volumen termina en negativo (lo cual es un error del modelo).

Esta comprobación es clave para verificar consistencia. A continuación, en la tabla 18 (tabla) puede verse la información referida. Como puede verse en la tabla debajo ninguna sucursal

termina con un volumen final que supere su capacidad, incluido el centro de distribución (comparativa entre tercer columna y cuarta columna que se manifiesta en la quinta columna con una tilde positiva).

Ubicación	Volumen inicial	Volumen final	Capacidad	¿Excede capacidad?	¿Volumen final positivo?
81	0,0682	7,16	500	☑	☑
82	1,9716	9,946	500	☑	☑
169	0,1596	2,3556	500	☑	☑
173	0,1948	10,1786	500	☑	☑
176	0,0168	2,0112	500	☑	☑
111	446,1862	416,9458	10000000	☑	☑

Tabla 7: Verificación de volumen y capacidad.

5.Eficiencia de Camiones:

En la **tabla 8** se puede ver la comprobación asociada a la eficiencia de camiones. Las columnas que se muestran son las siguientes:

- Volumen total transportado.
- Capacidad total contratada (camiones × capacidad).
- Porcentaje de uso efectivo.
- Check de cumplimiento de mínimo de carga por camión.

Esta comprobación ayuda a entender el grado de eficiencia logística del modelo y sale del cociente entre el volumen total transportado en metros cúbicos y la capacidad total contratada para una ruta.

Ruta	Camiones utilizados	Volumen total transportado (m³)	Capacidad total contratada (m³)	Eficiencia de llenado (%)	¿Cumple mínimo requerido?
111 → 169	1	9,2878	10	92,88	☑
111 → 173	1	9,9838	10	99,84	☑
111 → 176	1	9,9688	10	99,69	☑
169 → 81	1	7,0918	10	70,92	☑
176 → 82	1	7,9744	10	79,74	☑

Tabla 8: Verificación de volumen y capacidad.

6. Envíos Cruzados Detectados (No presente en el informe a menos que se den ejemplos).

Identifica si, para un mismo SKU, hubo:

- Envíos en ambas direcciones entre un par de sucursales (ej. 81→82 y 82→81).
- Esto no debería ocurrir si se activaron correctamente las restricciones de no redundancia.

Esta comprobación ayuda a hacer un diagnóstico para identificar redundancias logísticas y posibles ajustes en restricciones.

En función al proceso de análisis de resultados se realizaron diferentes corridas del modelo 12 en donde se buscó comparar cual es la diferencia a nivel costos logísticos de correr el modelo con los datos proyectados por el algoritmo de distribución (datos output reales que obtuvo el retailer y que luego fueron ajustados por el proceso de pauta manual, con necesidad de

intervención humana) y los datos proyectados por el modelo Prophet, totalmente automatizado (únicamente toma como input fechas especiales y demanda histórica).Esto se puede ver en la **figura 12**. En el marco de ese análisis llamaremos resultados "A" al modelo que corre con los datos predicción de Prophet y resultados "B" al modelo que corre con los datos basado en el output del algoritmo de distribución. Vamos a revisar los costos comparativos corriendo los modelos con SKU killers y sin SKU killers, es decir con un mínimo de SKU por sucursal establecido por el negocio como input del modelo.

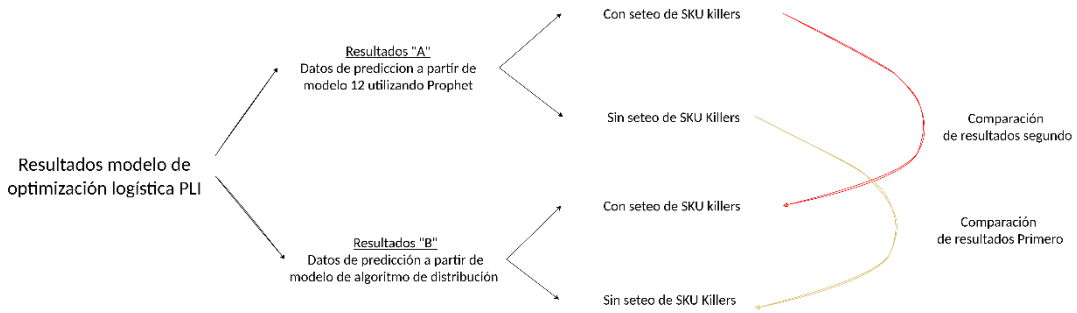

Figura 12: Verificación de volumen y capacidad.

Comparativo primero: Resultados "A" y Resultados "B" (sin seteo de SKU Killers):

El costo total de los resultados "A" fue de $ 66.500 con un costo variable de $16.500 para el envío de 479 unidades. Fueron necesarios 5 camiones que tuvieron un promedio de llenado de 74,56%. Las rutas que se recorrieron fueron principalmente entre sucursales (se registra únicamente la necesidad de enviar mercadería desde el centro de distribución). Es importante destacar que el output del script asociado a este proceso fue Óptimo.

El costo total de los resultados "B" fue de $ 75.500 con un costo variable de $ 25.500 para el envio de 689 unidades. Fueron necesarios 5 camiones que tuvieron un promedio de llenado de 94,34%, bastante más alto que el que se dio para el set de resultados "A", aunque esto se debe a que hubo un aumento de envío de mercadería ociosa. Las rutas que se recorrieron, a diferencia del modelo 24, no fueron principalmente rutas entras las sucursales, sino que esta vez la sucursal 169,173 y 176 fueron abastecidas desde el centro de distribución mientras que las sucursales 81 y 82 fueron abastecidas desde las sucursales 169 y 176 respectivamente. Los viajes desde el centro de distribución terminaron generando una diferencia a nivel de costo logístico variable. Es importante destacar que el output del modelo fue óptimo en función a las restricciones que se implementaron.

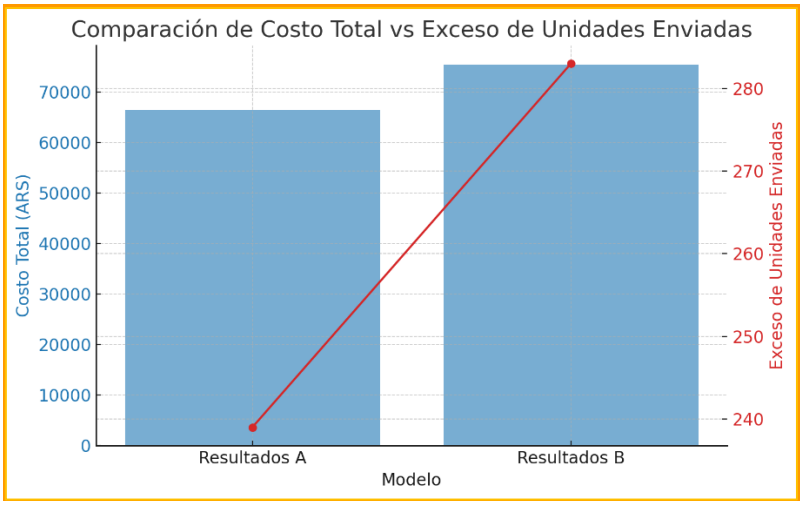

Figura 13: Comparativo de resultados por costo total y por exceso de unidades enviadas.

En la **figura 13** se puede ver la comparación del costo logístico total con el exceso de unidades enviadas por cada set de resultados. El **eje izquierdo (barras azules)** muestra el costo total en ARS mientras que el eje derecho **(línea roja)** indica cuántas unidades más se enviaron respecto a las ventas reales como medida de performance contra lo que realmente se envió ese día en la operación.

En conclusión, el set de resultados "A" fue más eficiente: tuvo menor costo total y menor exceso de unidades enviadas; mientras que el set de resultados "B", envió 284 unidades extra y gastó más (90 unidades más y $9.000 adicionales) para enviar mercadería que realmente no fue necesaria con respecto a las ventas que se terminaron dando en la realidad.

Esto demuestra que, para este escenario, la propuesta de predicción enviada por Prophet y corrida con el modelo PLI logró un mejor balance entre ajuste a la demanda real y control de costos logísticos.

Comparativo segundo: Resultados "A" y Resultados "B" (con seteo de Sku Killers):

En términos de abordar el problema con el seteo de cantidades predefinidas para SKU Killers, se adoptó una configuración en el modelo para que al momento de realizar el input de datos se incluyan 6 SKUs que tengan un mínimo de cantidades por sucursal. Esta configuración fue impuesta realmente para la operación para ese día dado que se trataba de un producto de telefonía celular que debía estar presente obligatoriamente a nivel tienda física.

A nivel modelo el cambio no fue radical pero sí a nivel costos. La suma de estas cantidades neteadas para ambos modelos que se resolvieron óptimamente equiparó los costos y la cantidad de camiones que eran necesarios activar.

El set de resultados "A" y "B" terminaron arrojando el mismo valor de costo logístico: $ 75.500. El set de resultados "B" nos marca que se transportaron 771 unidades para lo que se necesitaron 5 camiones siendo las rutas de esos camiones 111 -> 169, 111-> 173, 111-> 176, 169->81 y 176-> 82.La eficiencia de llenado de camiones fue de 91,23% en promedio.

Por otro lado para el set de resultados "A" se terminaron transportando 809 unidades para lo

que se necesitaron también 5 camiones. Para este caso se repitieron las mismas rutas que para el escenario “B” y se tuvo una eficiencia del 88,61% de utilización de espacio dentro de los camiones.

En términos de comparación contra la venta real para esa fecha, si bien el set de resultados “A” sobredimensionó a nivel general la cantidad total de productos a vender al revisar el mix de productos se puede notar que las 809 unidades se ajustan más a la realidad que las 771 unidades que muestra el escenario “B”, dado que en el escenario “B” se sobreestiman unidades que no se van a necesitar para ciertos productos específicos y sobredimensiona la cantidad general de unidades a entregar para los productos que están por fuera de SKU killers.

En conclusión, para este caso podemos decir que los resultados que se obtuvieron a partir de correr el modelo PLI volvieron a ser más eficientes dado que para un mayor número de unidades transportadas se logró ajustar mejor a la demanda real manteniendo el mismo costo que el otro modelo.

Por otro lado, también, en este apartado se avanzó con el ejercicio de comparar los resultados obtenidos a partir de los modelos de predicción y optimización desarrollados con lo que efectivamente se terminó disponiendo como rutas durante el día en cuestión por parte del equipo de operaciones logísticas. Es decir, se busca entender no solo qué habría sido óptimo proyectar o distribuir, sino también cómo se operó en la práctica ese día.

De acuerdo con los registros del área de planificación estratégica, el 8 de diciembre se ejecutaron 15 viajes con un costo logístico total superior a los $100.000. Lamentablemente, no se cuenta con el detalle desagregado de la asignación de productos por sucursal para esos envíos, pero sí con la lógica general que se aplicó de forma manual. Dicha lógica, históricamente utilizada por los planificadores en Córdoba, responde a cuatro criterios principales: (a) prioridad a las sucursales con mayor volumen histórico de demanda; (b) asignación proporcional en función del stock disponible en el centro de distribución (CD); (c) evitar transferencias si una sucursal ya cuenta con stock, priorizando solo envíos donde el inventario se encuentra en cero; y (d) completar camiones con al menos el 70% de su capacidad volumétrica estimada. En este caso, el orden de asignación fue el tradicional: sucursal 81 → 82 → 169 → 173 → 176.

Sobre la base de esa lógica y utilizando como insumo los valores de predicción del algoritmo de distribución tradicional, se consideraron como referencia los envíos que efectivamente se realizaron a las sucursales durante la jornada del 8 de diciembre, entendidos como el escenario real de operación. Esta asignación manual, basada en experiencia operativa, replicó una lógica de distribución históricamente aplicada en el territorio de Córdoba. Al contrastar este escenario real con los resultados obtenidos a partir del modelo de optimización lineal —escenarios “A” y “B”, construidos con insumos de demanda generados mediante Prophet (sin redondeo y con redondeo hacia arriba, respectivamente)— se observa que ambos modelos optimizados resultan ampliamente superadores en términos de eficiencia y costos logísticos. En particular, ambos permiten cumplir con la demanda proyectada utilizando menos cantidad de viajes y logrando una mejor utilización del volumen disponible en cada envío. Esta comparación evidencia el potencial de la combinación entre predicción automatizada y asignación optimizada frente a métodos manuales de planificación logística.

5. CONCLUSIONES

A lo largo de esta tesis se abordó un problema central para la operación de un retailer argentino dedicado a la comercialización de productos en tiendas físicas a partir de ventas por canales físicos como a través de canales digitales: la necesidad de mejorar su modelo de planificación logística y distribución de mercadería a sucursales. La problemática detectada se centra en un sistema de planeamiento que, si bien había sido útil en un contexto histórico diferente, hoy presenta limitaciones estructurales para enfrentar la dinámica de consumo actual. Entre los principales problemas del modelo en uso detectado se encontraron la variación sistemática de dar con una demanda adecuada, la alta dependencia de procesos manuales para la planificación diaria y la falta de automatización en la consolidación de cargas y optimización del transporte.

En función de este diagnóstico, se propuso como solución el desarrollo de un esquema metodológico que integra dos componentes principales: un modelo predictivo de demanda basado en series temporales (implementado con Prophet) y un modelo de optimización logística lineal (construido mediante PuLP en Python) que permite asignar productos de forma eficiente desde el centro de distribución hacia las sucursales. El objetivo general de esta propuesta fue reducir costos logísticos y minimizar el impacto de la insatisfacción de la demanda.

Desde el punto de vista predictivo, la implementación de Prophet permitió pronosticar la demanda semanal de productos a nivel de combinación SKU-sucursal. Este modelo, especialmente diseñado para manejar series temporales con estacionalidad y efectos de feriados, ofreció una mejora sustancial frente a los métodos anteriores utilizados por el retailer, basados en promedios históricos rígidos y reglas de negocio manuales. La capacidad de Prophet de modelar dinámicamente tendencias y estacionalidades resultó en una predicción más ajustada, como quedó evidenciado en la comparación de resultados. El modelo logró un MAPE global competitivo con respecto al MAPE del proceso que se corre actualmente en la empresa (algoritmo de distribución).

Por el lado de la optimización logística, el modelo desarrollado permitió considerar simultáneamente múltiples restricciones críticas para el negocio: capacidad de camiones, stock inicial disponible, volúmenes de productos, prioridades de abastecimiento entre sucursales, costos fijos y variables de transporte, mínimos por categoría y mínimos específicos para SKUs estratégicos (los llamados SKU killers). A diferencia del proceso manual previo, el modelo asegura que cada movimiento de mercadería respete la disponibilidad real de stock, la capacidad física de las sucursales y las prioridades estratégicas establecidas, minimizando además la posibilidad de errores humanos o de sobrecostos logísticos por falta de planificación anticipada.

El proceso de validación de resultados, en el que se comparó la performance del modelo de optimización utilizando como input los datos proyectados de Prophet contra los datos proyectados por el algoritmo tradicional, permitió concluir que la combinación de ambos modelos ofrece beneficios concretos al retailer. En escenarios sin imposición de mínimos por SKU killer, el modelo basado en Prophet mostró un menor costo logístico total y una menor cantidad de unidades enviadas en exceso, demostrando mayor ajuste a la demanda real y, en consecuencia, mayor eficiencia operativa. En escenarios donde sí se impusieron mínimos

estratégicos por producto, ambos modelos lograron costos logísticos similares, pero el modelo basado en Prophet presentó un mejor ajuste en el mix de productos entregados, alineándose más adecuadamente con las necesidades comerciales del negocio.

Más allá de los resultados numéricos, la implementación de este modelo integral representa una oportunidad significativa para transformar la forma en que el retailer aborda su planificación logística. La posibilidad de automatizar la generación de proyecciones de demanda y la distribución óptima de productos implica una reducción sustancial en la carga operativa diaria del equipo de planificación. En lugar de dedicar varias horas al día a la corrección manual de pautas de entrega y al ajuste de camiones, los planificadores podrán focalizarse en actividades de mayor valor agregado, como el análisis estratégico de tendencias de consumo, la evaluación de nuevas oportunidades comerciales o la gestión de relaciones con proveedores y operadores logísticos.

Adicionalmente, la adopción de un modelo de planificación basado en datos reales y optimización matemática contribuye a una mayor transparencia en la toma de decisiones, facilitando auditorías internas, simulaciones de escenarios alternativos y negociaciones más informadas con transportistas y proveedores de servicios logísticos. Esta transición hacia una cultura de decisiones basadas en datos es un paso fundamental para que el retailer pueda sostener su competitividad en un mercado cada vez más exigente, caracterizado por consumidores que esperan disponibilidad inmediata de productos, diversidad de canales de entrega y experiencias de compra consistentes.

En síntesis, el trabajo desarrollado en esta tesis demuestra que la utilización de modelos predictivos de demanda combinados con algoritmos de optimización logística no solo es técnicamente viable en el contexto de un retailer argentino, sino que resulta en beneficios tangibles tanto a nivel de costos como de eficiencia operativa. La solución propuesta permite abordar de manera estructurada los desafíos de planificación que hoy son resueltos de manera manual y empírica, abriendo un camino hacia una operación más inteligente, ágil y sustentable.

El caso analizado confirma que incluso en empresas que cuentan con restricciones presupuestarias, infraestructuras tecnológicas anticuadas y procesos de extracción de datos parcialmente manuales, es posible dar un salto cualitativo significativo mediante el uso estratégico de herramientas de análisis y optimización basadas en datos.

A pesar de los resultados positivos obtenidos, es importante destacar que toda solución basada en modelización matemática está sujeta a ciertos límites operativos, organizacionales y contextuales que deben ser reconocidos. Por ejemplo, el desempeño del modelo de predicción depende directamente de la calidad y granularidad de los datos históricos disponibles, y puede verse afectado por eventos atípicos no capturados en los registros previos (como promociones inesperadas o cambios drásticos en la estrategia comercial). Asimismo, la implementación del modelo de optimización en un entorno real puede requerir adaptaciones frente a restricciones informales no modeladas, como decisiones políticas internas, particularidades del layout físico en tiendas, o disponibilidad operativa de transportistas. Más allá de evaluar mejoras exclusivamente desde el punto de vista técnico, una evolución futura del sistema debería también contemplar mecanismos de validación colaborativa con usuarios operativos, incorporación de feedback humano en tiempo real, y revisión continua del modelo con una lógica de mejora incremental, integrando no solo nuevas herramientas tecnológicas, sino también capacidades organizacionales que favorezcan su adopción, interpretación y adaptación.

Finalmente, es importante remarcar que este proyecto se centra en resolver una porción específica de la problemática logística general del retailer, y que su éxito abre la puerta a futuras extensiones. Entre las principales oportunidades de evolución se encuentran la incorporación de modelos de predicción basados en técnicas más sofisticadas como XGBoost, la optimización de rutas de transporte considerando ventanas horarias y restricciones dinámicas de tráfico, y la automatización completa del pipeline de datos para eliminar la necesidad de intervenciones manuales. En conclusión, los resultados obtenidos validan el enfoque metodológico adoptado y confirman que los modelos de predicción y optimización desarrollados en esta tesis constituyen un aporte concreto para mejorar la eficiencia logística de la empresa, sentando las bases para futuras innovaciones en su operación comercial.

6. REFERENCIAS

Bertsimas, D., & Thiele, A. (2004). A robust optimization approach to supply chain management. *Lecture Notes in Computer Science*, *3064*,86/100.https://www.researchgate.net/publication/37595220

Dantzig,G.B.(1963).*Linear Programming and Extensions*.Princeton University Press.

Juhász, I., & Bányai, T. (2018). Cost analysis and optimization of last mile logistics. *Logistics Journal*, *448*(1),1–11.

Tsao, Y. C., Mangotra, M., Lu, J. C., & Dong, M. (2012). Inventory policies and network design in a supply chain with multiple distribution centers. *Journal of the Operational Research Society*, *63*(10), 1353–1367.

7. APÉNDICE A: PROPUESTA DE MEJORA MODELO PREDICTIVO: XGBOOST

El modelo de predicción actual, basado en Prophet, permitió establecer una primera aproximación sólida para la estimación de la demanda semanal a nivel SKU y sucursal. Sin embargo, el análisis detallado de los resultados muestra que aún existe margen de mejora en términos de ajuste fino de la predicción y reducción de errores relativos medios y del MAPE. En este contexto, se plantea la necesidad de explorar la implementación de modelos de machine learning más flexibles y potentes, siendo XGBoost una de las alternativas más relevantes.

Conceptualmente, Prophet modela la demanda como una combinación aditiva de componentes de tendencia, estacionalidad y eventos especiales, utilizando internamente técnicas de descomposición de series temporales. Si bien es altamente interpretativo y sencillo de implementar, Prophet parte de supuestos predefinidos sobre la estructura temporal de los datos, lo cual puede ser una limitación cuando existen interacciones complejas entre múltiples factores que afectan la demanda. Además, Prophet requiere una transformación explícita de la serie temporal a frecuencia semanal, lo cual puede perder información intersemanal o patrones de variabilidad más fina.

XGBoost, en cambio, plantea un enfoque completamente diferente. Se basa en el entrenamiento de conjuntos de árboles de decisión potenciados mediante boosting, permitiendo capturar relaciones no lineales y complejas entre múltiples variables explicativas.

Estructura del modelo:

$$y(t) \approx f(X) = m = 1\sum MT(X) \qquad (14)$$

Donde:

- **X** = vector de variables explicativas (mes, día, lag, feriado, sucursal, etc.)
- **T(X)** = árboles de decisión individuales

Al no asumir una estructura fija sobre la serie, XGBoost puede modelar de manera más natural la interacción entre factores como el mes del año, la semana, el día del mes, la condición de feriado, el historial de ventas anteriores (lags), el SKU y la sucursal. Este enfoque basado en variables enriquecidas tiene el potencial de mejorar la capacidad predictiva del modelo, reducir el error medio y acercar el MAPE a valores más acordes con los estándares de la industria de consumo masivo (20-30%).

Para adaptar el modelo de predicción de Prophet hacia XGBoost, los principales cambios conceptuales serían los siguientes. Primero, se requiere una reformulación del dataset: ya no se trabaja exclusivamente con una estructura de tiempo y valor, sino que es necesario construir un conjunto de variables explicativas o features. Estas variables incluirían información temporal

(mes, semana, día), información categórica (SKU, sucursal), indicadores binarios (feriados) y variables de memoria como lags de ventas pasadas.

Segundo, el modelo dejaría de trabajar en una frecuencia semanal obligatoria. En su lugar, cada fila representaría una observación diaria enriquecida con atributos, permitiendo capturar patrones de corto y largo plazo sin necesidad de imponer un esquema aditivo preestablecido.

Tercero, la validación del modelo seguiría el mismo esquema adoptado con Prophet: una separación temporal de los datos en conjuntos de entrenamiento y prueba, utilizando las últimas cuatro semanas como horizonte de evaluación. Se mantendría la métrica de MAPE como indicador principal de desempeño, pero ahora calculado sobre las predicciones generadas a partir del modelo XGBoost.

Finalmente, la implementación de XGBoost requeriría prestar especial atención a la ingeniería de features y a la optimización de hiperparámetros. Factores como la profundidad máxima de los árboles, la tasa de aprendizaje y el número de estimadores podrían tener un impacto significativo en la capacidad predictiva. La posibilidad de incluir información externa adicional (como precios, promociones o variables económicas) también sería una ventaja comparativa respecto a Prophet.

En conclusión, el pasaje conceptual de Prophet a XGBoost implica abandonar una estructura rígida de serie temporal en favor de un enfoque más flexible y multivariable, donde el modelo aprende directamente de las relaciones presentes en los datos. Este cambio está motivado no solo por la posibilidad de capturar patrones más complejos, sino también por el objetivo explícito de reducir el MAPE y el error medio, optimizando así la calidad del input que alimentará el modelo de optimización logística en las etapas posteriores.

8. APÉNDICE B: PROPUESTA DE MEJORA EN MODELO DE CONSOLIDACIÓN DE CARGA Y DESPACHO

La necesidad de maximizar la utilización del espacio disponible en los vehículos de transporte representa un desafío crítico en la planificación logística de cualquier empresa que busque mejorar su eficiencia operativa y reducir costos. Particularmente en el caso analizado en esta tesis, donde se planifica la distribución de productos desde un centro de distribución hacia múltiples sucursales, resulta evidente que una mejora adicional en la operación puede lograrse abordando específicamente la problemática del **acomodamiento eficiente de la carga en camiones**.

Actualmente, el modelo de optimización desarrollado considera restricciones de volumen total transportado, limitaciones de capacidad de camiones y mínimos de carga necesarios para activar un viaje. Sin embargo, no se modela en forma detallada cómo deberían ser ubicados los productos dentro del espacio físico disponible. Incorporar esta dimensión permitiría optimizar no solo las cantidades enviadas, sino también la configuración física del transporte, maximizando la utilización real de la bodega y, por ende, disminuyendo la cantidad total de viajes requeridos.

Para adaptar este concepto al contexto de esta tesis, se plantea la construcción de una **función de optimización lineal** cuyo objetivo sea maximizar el volumen de carga efectiva transportada en cada camión activado, respetando las capacidades volumétricas disponibles.

Conceptualmente, el modelo de optimización debería definirse en torno a las siguientes premisas:

El universo de ítems a distribuir ya ha sido determinado en la etapa previa de optimización de asignación de productos a sucursales. Por tanto, el problema de acomodamiento debe trabajar sobre las cantidades resultantes para cada ruta origen–destino.

Cada ítem posee un volumen unitario conocido. La suma de los volúmenes de los ítems asignados a un mismo camión no puede superar la capacidad total disponible en metros cúbicos de dicho vehículo.

La función objetivo debe estar orientada a maximizar el volumen total cargado en el camión, optimizando el espacio ocupado.

Dado que ciertos productos pueden tener restricciones físicas adicionales (por ejemplo, apilabilidad limitada, necesidad de mantener orientación, entre otros), en una versión futura más avanzada del modelo podrían incorporarse también restricciones de compatibilidad entre ítems. Sin embargo, en una primera instancia conceptual se asume que todos los ítems son compatibles entre sí.

Bajo este marco conceptual, se puede describir el planteo del modelo de la siguiente manera.

Se definen variables de decisión binarias que indican si un ítem específico es cargado o no en un determinado camión en una determinada ruta. Alternativamente, si se considera que ya existe una asignación de cantidades a nivel origen-destino, puede definirse una variable continua que represente la cantidad de unidades de cada SKU cargadas en cada camión específico.

La función objetivo busca maximizar la suma total del volumen de todos los ítems efectivamente cargados en el camión. Esto equivale a maximizar el porcentaje de ocupación efectiva de la bodega disponible.

Se impone como restricción que el volumen total cargado no supere la capacidad volumétrica del camión.

Se debe respetar que no se carguen más unidades de cada ítem que las que han sido asignadas a la ruta correspondiente en la etapa previa de optimización de asignación.

Si se plantea un esquema con múltiples camiones disponibles para una misma ruta, se podría también optimizar el número de camiones utilizados minimizando la cantidad total de vehículos activos, aunque este objetivo secundario puede ser modelado por separado o mediante una penalización en la función objetivo.

Formalmente, el modelo lineal básico podría expresarse del siguiente modo:

Declaración de variables:

- $X_i \in \mathbb{Z} \geq 0$: Cantidad de unidades del ítem i cargadas en el camión.
- $v_i \in \mathbb{R} \geq 0$: Volumen unitario del ítem i.
- $V_{\{camión\}} \in \mathbb{R} \geq 0$: Volumen máximo disponible en el camión.
- $d_i \in \mathbb{Z} \geq 0$: Cantidad total de unidades del ítem i disponibles para enviar en la ruta.
- $d_i \in \mathbb{Z} \geq 0$: Cantidad total de unidades del ítem i disponibles para enviar en la ruta.

Función Objetivo:

Maximizar el volumen unitario por ítem por la cantidad de ítems para el camión en cuestión.

$$\sum_i (v_i \times x_i) \qquad (15)$$

Restricciones:

$$\sum_i (v_i \times x_i) \leq V_{\{camion\}} \qquad (16)$$

El volumen total que se planea mandar no puede ser mayor que el volumen del depósito del camión.

$$0 \leq x_i \leq d_i \forall\, i \qquad (17)$$

La cantidad total de items a enviar en el camión no puede ser mayor que la cantidad de items a enviar en la ruta.

En otras palabras, el modelo busca decidir cuántas unidades de cada producto cargar para llenar el camión lo máximo posible, sin exceder ni el volumen disponible ni las unidades asignadas previamente.

Es importante remarcar que la inclusión de este modelo de optimización de carga no implica la sustitución del modelo de asignación de productos a sucursales, sino que constituye una **segunda capa de optimización complementaria**. Primero se decide qué productos deben ser enviados a cada sucursal en función de la demanda proyectada, stock disponible, capacidades de almacenamiento y costos logísticos. Luego, sobre esa asignación determinada, se optimiza la forma en que los productos se distribuyen físicamente dentro de los camiones para maximizar el aprovechamiento del espacio y reducir costos asociados al transporte.

La implementación de este modelo permitiría al retailer no solo reducir el costo por viaje en términos de mejor utilización de los recursos, sino también mejorar la planificación de flotas, reducir tiempos de carga y descarga, y aumentar la capacidad de respuesta ante aumentos súbitos de la demanda o cambios en las rutas de distribución.

En resumen, la propuesta de desarrollar un modelo de optimización lineal específico para el problema de acomodamiento de carga en camiones representa una extensión natural del trabajo realizado en esta tesis. Su incorporación permitiría capturar un nuevo nivel de eficiencia en la red logística del retailer, alineándose con las mejores prácticas internacionales en la gestión de transporte y distribución.

9. ANEXO 1: SCRIPT DE PREDICCIÓN BASADO EN LIBRERÍA PROPHET

```
# IMPORTS
import pandas as pd
from prophet import Prophet
from tqdm import tqdm
import warnings
import math
warnings.filterwarnings("ignore")

# CARGAR ARCHIVO
archivo = "Archivo final modelo prophet.xlsx"  # asegurate que esté en la misma carpeta
df = pd.read_excel(archivo)

# Crear DataFrame con feriados especiales
feriados = pd.DataFrame({
    'holiday': ['hot_sale'] * 3 + ['cyber_monday'] * 3,
    'ds': pd.to_datetime([
        '2024-05-13', '2024-05-14', '2024-05-15',
        '2024-11-04', '2024-11-05', '2024-11-06'
    ]),
    'lower_window': 0,
    'upper_window': 0
})


# FORMATO DE FECHA
df['Fecha Entrega'] = pd.to_datetime(df['Fecha Entrega'])
df['Semana'] = df['Fecha Entrega'].dt.to_period('W').apply(lambda r: r.start_time)
```

```
# AGRUPAR POR SKU + SUCURSAL + SEMANA
ventas_semanales = df.groupby(['cod_articulo', 'Numero_Sucursal', 'Semana'])['cantidad'].sum().reset_index()
ventas_semanales.rename(columns={'Semana': 'ds', 'cantidad': 'y'}, inplace=True)

################################################################################
#####
###### CREAR DATAFRAME TEST CON LAS ÚLTIMAS 4 SEMANAS DE CADA COMBINACIÓN########
################################################################################
####

ventas_semanales_test = (
   ventas_semanales
   .sort_values(['cod_articulo', 'Numero_Sucursal', 'ds'])
   .groupby(['cod_articulo', 'Numero_Sucursal'])
   .tail(4)  # toma las 4 últimas semanas por grupo
   .reset_index(drop=True)
)

# FUNCION PARA CALCULAR SEMANA DEL MES DINÁMICA
def semana_del_mes(fecha):
   semana = (fecha.day - 1) // 7 + 1
   mes = fecha.strftime("%B").capitalize()
   return f"{semana}° semana de {mes}"

# ASEGURARSE QUE LA COLUMNA 'ds' SEA DATETIME
ventas_semanales_test['ds'] = pd.to_datetime(ventas_semanales_test['ds'])

# AGREGAR COLUMNA
ventas_semanales_test['Semana del mes'] = ventas_semanales_test['ds'].apply(semana_del_mes)
```

```
# Mostrar ejemplo de resultado
print(" Muestra del DataFrame de test:")
print(ventas_semanales_test.head())

df_ventas_semanales_test = pd.DataFrame(ventas_semanales_test)

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

# CREAR DATAFRAME TRAIN EXCLUYENDO LAS ÚLTIMAS 4 SEMANAS DE CADA COMBINACIÓN
ventas_semanales_train = (
   ventas_semanales
   .sort_values(['cod_articulo', 'Numero_Sucursal', 'ds'])
   .groupby(['cod_articulo', 'Numero_Sucursal'])
   .apply(lambda g: g.iloc[:-4])  # todo menos las últimas 4 semanas
   .reset_index(drop=True)
)

# Mostrar ejemplo de resultado
print(" Muestra del DataFrame de entrenamiento:")
print(ventas_semanales_train.head())

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

# AGRUPAR COMBINACIONES
combinaciones = ventas_semanales_train.groupby(['cod_articulo', 'Numero_Sucursal'])
```

```
# LISTA DE RESULTADOS
resultados = []

# FUNCIÓN PARA CALCULAR SEMANA DEL MES DINÁMICA
def semana_del_mes(fecha):
    semana = (fecha.day - 1) // 7 + 1
    mes = fecha.strftime("%B").capitalize()
    return f"{semana}° semana de {mes}"

# LOOP POR CADA COMBINACIÓN
for (SKU, sucursal), grupo in tqdm(combinaciones, desc="Proyectando últimas 4 semanas"):
    if len(grupo) >= 6:
        grupo = grupo.sort_values('ds')

        modelo = Prophet(interval_width=0.95, weekly_seasonality=True, daily_seasonality=False,
holidays=feriados)
        modelo.fit(grupo)

        future = modelo.make_future_dataframe(periods=4, freq='W')
        forecast = modelo.predict(future)

        forecast_ultimas_4 = forecast.tail(4).copy()

        for _, fila in forecast_ultimas_4.iterrows():
            fecha_entrega = fila['ds'].date()

            yhat = max(fila['yhat'], 0)
            yhat_lower = max(fila['yhat_lower'], 0)
            yhat_upper = max(fila['yhat_upper'], 0)
```

```
        yhat_redondeado = 0 if yhat <= 0 else math.ceil(yhat)

        resultados.append({
            'Fecha entrega': fecha_entrega,
            'Semana del mes': semana_del_mes(fecha_entrega),
            'cod_articulo': SKU,
            'Numero_Sucursal': sucursal,
            'Cantidad (predicción)': round(yhat, 2),
            'Cantidad (predicción redondeada)': yhat_redondeado,
            'Cantidad (rango inferior)': round(yhat_lower, 2),
            'Cantidad (rango superior)': round(yhat_upper, 2)
        })

# CONVERTIR A DATAFRAME
df_resultados = pd.DataFrame(resultados)

######################################################################
######## Calculo de MAPE y de error medio ####################
######################################################################

df_comparacion = df_ventas_semanales_test.merge(
    df_resultados,
    how='inner',
    left_on=['Semana del mes', 'cod_articulo', 'Numero_Sucursal'],
    right_on=['Semana del mes', 'cod_articulo', 'Numero_Sucursal']
)

df_comparacion = df_comparacion.rename(columns={
    'y': 'real',
    'Cantidad (predicción)': 'predicho'
})
```

```
# Cálculo tradicional
df_comparacion['error_abs'] = abs(df_comparacion['real'] - df_comparacion['predicho'])
df_comparacion['error_rel'] = df_comparacion['error_abs'] / (df_comparacion['real'] + 1e-5)

# Cálculo contra redondeado
df_comparacion['error_absoluto_vs_valores_redondeados'] = abs(df_comparacion['real'] - df_comparacion['Cantidad (predicción redondeada)'])
df_comparacion['error_relativo_vs_valores_redondeados'] = df_comparacion['error_absoluto_vs_valores_redondeados'] / (df_comparacion['real'] + 1e-5)

# Mejora con redondeo
df_comparacion['¿Mejora con redondeo?'] = df_comparacion['error_relativo_vs_valores_redondeados'] < df_comparacion['error_rel']

# MAPEs
mape = df_comparacion['error_rel'].mean() * 100
mape_redondeado = df_comparacion['error_relativo_vs_valores_redondeados'].mean() * 100

# MOSTRAR
print(" MAPE global entre test y predicción:", round(mape, 2), "%")
print(" MAPE global con redondeo:", round(mape_redondeado, 2), "%")
print("\n Muestra de comparaciones:")
print(df_comparacion[['Semana del mes', 'cod_articulo', 'Numero_Sucursal', 'real', 'predicho',
                      'Cantidad (predicción redondeada)', 'error_rel',
                      'error_relativo_vs_valores_redondeados', '¿Mejora con redondeo?']].head())

# EXPORTAR
df_comparacion.to_excel("comparacion_test_vs_prediccion.xlsx", index=False)
print("Archivo exportado como 'comparacion_test_vs_prediccion.xlsx'")
```

```
################################################################################
#####
# Analisis sacando aquellos valores extremos de MAPE que distorsionan el modelo
################################################################################
####

# ANALIZAR DISTRIBUCIÓN DE ERRORES ALTOS
print(" Casos con error relativo muy alto:")
print("Errores > 100%:", (df_comparacion['error_rel'] > 1).sum())
print("Errores > 200%:", (df_comparacion['error_rel'] > 2).sum())
print("Errores > 300%:", (df_comparacion['error_rel'] > 3).sum())
print("Total filas:", len(df_comparacion))

# COPIA PARA NO ALTERAR EL ORIGINAL
df_comparacion_clip = df_comparacion.copy()

# CORTAR ERRORES > 300% (opcional, podés ajustar)
df_comparacion_clip['error_rel'] = df_comparacion_clip['error_rel'].clip(upper=3)

# NUEVO MAPE DESPUÉS DEL CORTE
mape_clipeado = df_comparacion_clip['error_rel'].mean() * 100

print("\n MAPE con errores acotados a máx. 300%:", round(mape_clipeado, 2), "%")

# AGRUPAR POR SKU
mape_por_SKU = (
    df_comparacion_clip
    .groupby('cod_articulo')['error_rel']
    .mean()
    .sort_values()
    .reset_index()
)
```

```
mape_por_sku['MAPE (%)'] = (mape_por_sku['error_rel'] * 100).round(2)
mape_por_sku.drop(columns='error_rel', inplace=True)

print("\n MAPE promedio por SKU:")
print(mape_por_sku.head(10))  # top 10 mejores SKUs



########################################
######C   COMPROBACION DE RESULTADOS
########################################

# Total de filas en la comparación
total_filas = len(df_comparacion)

# Filas excluidas por error relativo > 1 (100%)
excluidas_original = (df_comparacion['error_rel'] > 1).sum()
excluidas_redondeado = (df_comparacion['error_relativo_vs_valores_redondeados'] > 1).sum()

# Mostrar info
print(" DISTRIBUCIÓN DE ERRORES RELATIVOS ALTOS:")
print(f"Total de combinaciones evaluadas: {total_filas}")
print(f" Excluidas por error_rel > 100% (original): {excluidas_original} ({round(100 * excluidas_original / total_filas, 2)}%)")
print(f" Excluidas por error_rel_redondeado > 100%: {excluidas_redondeado} ({round(100 * excluidas_redondeado / total_filas, 2)}%)")

# Filtrado: se conservan solo las que tienen ambos errores <= 100%
df_comparacion_clip = df_comparacion[
   (df_comparacion['error_rel'] <= 1) &
   (df_comparacion['error_relativo_vs_valores_redondeados'] <= 1)
].copy()
```

```
filas_incluidas = len(df_comparacion_clip)

print(f" Filas utilizadas para cálculo de MAPE (ambos errores <= 100%): {filas_incluidas} ({round(100 * filas_incluidas / total_filas, 2)}%)")

# Calcular MAPEs sobre el conjunto limpio
mape_clipeado = df_comparacion_clip['error_rel'].mean() * 100
mape_clipeado_redondeado = df_comparacion_clip['error_relativo_vs_valores_redondeados'].mean() * 100

print("\n MAPE con errores <= 100% (original):", round(mape_clipeado, 2), "%")
print("MAPE con errores <= 100% (redondeado):", round(mape_clipeado_redondeado, 2), "%")



#########################
# Agrupación por SKU sobre el conjunto filtrado
#########################

mape_por_sku = (
    df_comparacion_clip
    .groupby('cod_articulo')['error_rel']
    .mean()
    .sort_values()
    .reset_index()
)

mape_por_sku['MAPE (%)'] = (mape_por_sku['error_rel'] * 100).round(2)
mape_por_sku.drop(columns='error_rel', inplace=True)

print("\n MAPE promedio por SKU (con errores <= 100%):")
print(mape_por_sku.head(10))
```

```
########################################

######## MAPER POR SUCURSAL  #############

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

# --- CONFIGURACIÓN INICIAL ---

total_por_sucursal = df_comparacion.groupby('Numero_Sucursal').size().reset_index(name='total_combinaciones')


# --- PREDICCIÓN ORIGINAL ---


# MAPE completo por sucursal

mape_original_completo = (

   df_comparacion.groupby('Numero_Sucursal')['error_rel']

   .mean().reset_index(name='MAPE (%) - original (incluye outliers)')

)


# MAPE filtrado (<= 100%)

original_filtrado = df_comparacion[df_comparacion['error_rel'] <= 1]

mape_original_filtrado = (

   original_filtrado.groupby('Numero_Sucursal')['error_rel']

   .mean().reset_index(name='MAPE (%) - original (filtrado)')

)

mape_original_filtrado['MAPE (%) - original (filtrado)'] = (mape_original_filtrado['MAPE (%) - original (filtrado)'] * 100).round(2)


# Conteo de excluidos

excluidos_original = df_comparacion.groupby('Numero_Sucursal').apply(lambda x: (x['error_rel'] > 1).sum()).reset_index(name='Excluidos (original)')

excluidos_original['% excluidos (original)'] = (

   100 * excluidos_original['Excluidos (original)'] / total_por_sucursal['total_combinaciones']

).round(2)
```

```
# Unir todo
tabla_original = mape_original_completo.merge(mape_original_filtrado, on='Numero_Sucursal', how='left') \
                 .merge(excluidos_original, on='Numero_Sucursal', how='left') \
                 .merge(total_por_sucursal, on='Numero_Sucursal', how='left')

tabla_original['MAPE (%) - original (incluye outliers)'] = (tabla_original['MAPE (%) - original (incluye outliers)'] * 100).round(2)

# --- PREDICCIÓN REDONDEADA ---

# MAPE completo por sucursal
mape_redondeado_completo = (
   df_comparacion.groupby('Numero_Sucursal')['error_relativo_vs_valores_redondeados']
   .mean().reset_index(name='MAPE (%) - redondeado (incluye outliers)')
)

# MAPE filtrado (<= 100%)
redondeado_filtrado = df_comparacion[df_comparacion['error_relativo_vs_valores_redondeados'] <= 1]
mape_redondeado_filtrado = (
   redondeado_filtrado.groupby('Numero_Sucursal')['error_relativo_vs_valores_redondeados']
   .mean().reset_index(name='MAPE (%) - redondeado (filtrado)')
)
mape_redondeado_filtrado['MAPE (%) - redondeado (filtrado)'] = (mape_redondeado_filtrado['MAPE (%) - redondeado (filtrado)'] * 100).round(2)

# Conteo de excluidos
excluidos_redondeado = df_comparacion.groupby('Numero_Sucursal').apply(lambda x: (x['error_relativo_vs_valores_redondeados'] > 1).sum()).reset_index(name='Excluidos (redondeado)')
excluidos_redondeado['% excluidos (redondeado)'] = (
   100 * excluidos_redondeado['Excluidos (redondeado)'] / total_por_sucursal['total_combinaciones']
```

```
).round(2)

# Unir todo
tabla_redondeado = mape_redondeado_completo.merge(mape_redondeado_filtrado, on='Numero_Sucursal', how='left') \
                        .merge(excluidos_redondeado, on='Numero_Sucursal', how='left') \
                        .merge(total_por_sucursal, on='Numero_Sucursal', how='left')

tabla_redondeado['MAPE (%) - redondeado (incluye outliers)'] = (tabla_redondeado['MAPE (%) - redondeado (incluye outliers)'] * 100).round(2)

# --- MOSTRAR ---
print("\n TABLA 1: MAPE por sucursal - predicción original")
print(tabla_original.sort_values('MAPE (%) - original (filtrado)').head(10))

print("\n TABLA 2: MAPE por sucursal - predicción redondeada")
print(tabla_redondeado.sort_values('MAPE (%) - redondeado (filtrado)').head(10))



#######################################
# GRAFICOS
#######################################

# 1)

import matplotlib.pyplot as plt

def mostrar_tabla_original(tabla_original):
    fig, ax = plt.subplots(figsize=(12, 3))
    ax.axis('off')

    columnas = [
```

```
        'Numero_Sucursal',
        'MAPE (%) - original(incluye outliers)',
        'MAPE (%) - original(filtrado)',
        'Excluidos(original)',
        '% excluidos(original)'
    ]

    tabla_data = tabla_original[[
        'Numero_Sucursal',
        'MAPE (%) - original (incluye outliers)',
        'MAPE (%) - original (filtrado)',
        'Excluidos (original)',
        '% excluidos (original)'
    ]]

    table_plot = ax.table(cellText=tabla_data.values,
                          colLabels=columnas,
                          cellLoc='center',
                          loc='center')

    table_plot.auto_set_font_size(False)
    table_plot.set_fontsize(6)
    table_plot.scale(1.2, 1.2)
    plt.title(" 📊 MAPE por sucursal - Predicción original", fontsize=14, weight='bold')
    plt.tight_layout()
    plt.show()

def mostrar_tabla_redondeada(tabla_redondeado):
    fig, ax = plt.subplots(figsize=(12, 3))
    ax.axis('off')
```

```
    columnas = [
        'Numero_Sucursal',
        'MAPE (%) - redondeado(incluye outliers)',
        'MAPE (%) - redondeado(filtrado)',
        'Excluidos(redondeado)',
        '% excluidos(redondeado)'
    ]

    tabla_data = tabla_redondeado[[
        'Numero_Sucursal',
        'MAPE (%) - redondeado (incluye outliers)',
        'MAPE (%) - redondeado (filtrado)',
        'Excluidos (redondeado)',
        '% excluidos (redondeado)'
    ]]

    table_plot = ax.table(cellText=tabla_data.values,
                          colLabels=columnas,
                          cellLoc='center',
                          loc='center')

    table_plot.auto_set_font_size(False)
    table_plot.set_fontsize(6)
    table_plot.scale(1.2, 1.2)
    plt.title(" MAPE por sucursal - Predicción redondeada", fontsize=14, weight='bold')
    plt.tight_layout()
    plt.show()

mostrar_tabla_original(tabla_original)
mostrar_tabla_redondeada(tabla_redondeado)
```

```
# 2) Prediccion por Sku



def graficar_prediccion_vs_real(sku_objetivo, sucursal_objetivo, df_reales, df_predicciones):
    """
    Genera un gráfico comparativo entre valores reales y predicciones (con redondeo y márgenes).

    Parámetros:
        sku_objetivo (int): Código del artículo a graficar.
        sucursal_objetivo (int): Código de la sucursal.
        df_reales (DataFrame): DataFrame con las columnas 'cod_articulo', 'Numero_Sucursal', 'ds', 'y'
        df_predicciones (DataFrame): DataFrame con columnas:
            - 'cod_articulo', 'Numero_Sucursal', 'Fecha entrega',
            - 'Cantidad (predicción)', 'Cantidad (predicción redondeada)',
            - 'Cantidad (rango inferior)', 'Cantidad (rango superior)'
    """
    # Filtrar reales
    reales = df_reales[
        (df_reales['cod_articulo'] == sku_objetivo) &
        (df_reales['Numero_Sucursal'] == sucursal_objetivo)
    ].sort_values('ds')

    # Filtrar predicciones
    pred = df_predicciones[
        (df_predicciones['cod_articulo'] == sku_objetivo) &
        (df_predicciones['Numero_Sucursal'] == sucursal_objetivo)
    ].sort_values('Fecha entrega')

    if reales.empty or pred.empty:
        print(" No hay datos suficientes para graficar esta combinación.")
```

```
        return

    # Crear gráfico
    plt.figure(figsize=(12, 6))

    # Reales
    plt.plot(reales['ds'], reales['y'], color='black', marker='o', label='Real 2024')

    # Predicción original
    plt.plot(pred['Fecha entrega'], pred['Cantidad (predicción)'], color='red', marker='o', label='Predicción 2025')

    # Predicción redondeada
    plt.plot(pred['Fecha entrega'], pred['Cantidad (predicción redondeada)'], color='orange', marker='s', linestyle='--', label='Predicción redondeada')

    # Banda de margen de predicción
    plt.fill_between(
        pred['Fecha entrega'],
        pred['Cantidad (rango inferior)'],
        pred['Cantidad (rango superior)'],
        color='skyblue',
        alpha=0.4,
        label='Rango de predicción'
    )

    # Estética
    plt.title(f" SKU {sku_objetivo} -  Sucursal {sucursal_objetivo}")
    plt.xlabel("Fecha")
    plt.ylabel("Cantidad")
    plt.xticks(rotation=45)
    plt.legend()
```

```
    plt.grid(True)
    plt.tight_layout()
    plt.show()


graficar_prediccion_vs_real(
    sku_objetivo=12982,
    sucursal_objetivo=81,
    df_reales=ventas_semanales_test,
    df_predicciones=df_resultados)
```

ANEXO 2: SCRIPT DE OPTIMIZACIÓN LINEAL (MODELO 12 - PLI)

Se expone el código para el modelo 12 con Sku Killers (Resultados “B”), es importante para ver diferentes variaciones de output del modelo modificar la primera parte del Script a nivel de Inputs.

```
import pandas as pd

from pulp import LpProblem, LpMinimize, LpVariable, lpSum, LpStatus, value


# Crear el modelo

modelo = LpProblem("Distribución con prioridades y restricciones de categoría", LpMinimize)

# ==============================

# 1) PARÁMETROS DE ENTRADA SEGÚN ESCENARIO

# ==============================



   # Escenario de predicción 08/12/2024


sucursales = [81, 82, 169, 173, 176]


# ----------------------Bloque 1 - Sku ----------------#


skus = [12982, 13040, 13257, 13469, 160879, 250093, 502327, 781934, 782297,

        782311, 782327, 20632, 502347, 502579, 782349, 13696, 94742,

        502530, 782335, 13629, 502532, 782282,250001,782203,13662,90048,180304]


# skus = [12982, 13040, 13257]


# ----------------------Bloque 2 - Categorias ----------------#


categorias = {

   12982: "PEQUEÑOS CUIDADO PERSONAL  Mujer",

   13040: "PEQUEÑOS CUIDADO PERSONAL  Mujer",
```

```
    13257: "PEQUEÑOS CUIDADO PERSONAL  Mujer",
    13469: "CAFE E INFUSIONES",
    13629: "CAFE E INFUSIONES",
    13662: "CAFE E INFUSIONES",
    13696: "PREPARACION DE ALIMENTOS",
    20632: "R-ACONDICIONADORES(R2)",
    90048: "R-AGUA CALIENTE(R9)",
    94742: "R-AGUA CALIENTE(R9)",
    160879: "R-HELADERAS(R16)",
    180304: "PREPARACION DE ALIMENTOS",
    250001: "PEQUEÑOS HOGAR",
    250093: "PEQUEÑOS HOGAR",
    502327: "TV",
    502347: "TV",
    502530: "TV",
    502532: "TV",
    502579: "TV",
    781934: "CELULARES SIN LINEA",
    782203: "CELULARES SIN LINEA",
    782282: "CELULARES SIN LINEA",
    782297: "CELULARES SIN LINEA",
    782311: "CELULARES SIN LINEA",
    782327: "CELULARES SIN LINEA",
    782335: "CELULARES SIN LINEA",
    782349: "CELULARES SIN LINEA"
}
# ----------------------Bloque 3 - Volumen ----------------#

volumen_sku = {
    12982: 0.015, 13040: 0.015, 13257: 0.015, 13469: 0.036, 13629: 0.036,
    13662: 0.018, 13696: 0.075, 20632: 0.35, 90048: 0.18, 94742: 0.22,
```

```
    160879: 1.1, 180304: 0.015, 250001: 0.014, 250093: 0.021, 502327: 0.0048,
    502347: 0.0048, 502530: 0.12, 502532: 0.0012, 502579: 0.06, 781934: 0.006,
    782203: 0.0012, 782282: 0.0012, 782297: 0.006, 782311: 0.0024, 782327: 0.0036,
    782335: 0.0048, 782349: 0.0048
    }
# ---------- Bloque 4 - Stock ----------#

stock = {
    12982: {"111": 175, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    13040: {"111": 45, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    13257: {"111": 246, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    13469: {"111": 43, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    13629: {"111": 44, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    13662: {"111": 40, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    13696: {"111": 83, "81": 0, "82": 0, "169": 2, "173": 2, "176": 0},
    20632: {"111": 86, "81": 0, "82": 2, "169": 0, "173": 0, "176": 0},
    90048: {"111": 24, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    94742: {"111": 34, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    160879: {"111": 313, "81": 0, "82": 1, "169": 0, "173": 0, "176": 0},
    180304: {"111": 21, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    250001: {"111": 54, "81": 2, "82": 0, "169": 0, "173": 2, "176": 0},
    250093: {"111": 329, "81": 1, "82": 2, "169": 0, "173": 0, "176": 0},
    502327: {"111": 248, "81": 2, "82": 1, "169": 0, "173": 0, "176": 0},
    502347: {"111": 152, "81": 0, "82": 1, "169": 0, "173": 0, "176": 0},
    502530: {"111": 141, "81": 0, "82": 1, "169": 0, "173": 0, "176": 0},
    502532: {"111": 21, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    502579: {"111": 174, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    781934: {"111": 254, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    782203: {"111": 52, "81": 0, "82": 0, "169": 0, "173": 0, "176": 0},
    782282: {"111": 7, "81": 0, "82": 0, "169": 0, "173": 0, "176": 1},
    782297: {"111": 310, "81": 0, "82": 0, "169": 1, "173": 0, "176": 2},
```

```
    782311: {"111": 95, "81": 1, "82": 0, "169": 0, "173": 1, "176": 0},
    782327: {"111": 62, "81": 2, "82": 0, "169": 1, "173": 0, "176": 1},
    782335: {"111": 155, "81": 0, "82": 0, "169": 0, "173": 2, "176": 0},
    782349: {"111": 205, "81": 0, "82": 0, "169": 0, "173": 1, "176": 0}
}
# ----------------------Bloque 5 - Demanda ----------------#

demanda = {
    81: {
        12982: 6, 13040: 0, 13257: 2, 13469: 1, 13629: 0, 13662: 0, 13696: 0, 160879: 6,
        180304: 0, 20632: 0, 250001: 4, 250093: 3, 502327: 5, 502347: 3, 502530: 0,
        502532: 0, 502579: 0, 781934: 8, 782203: 15, 782282: 0, 782297: 16, 782311: 5,
        782327: 14, 782335: 5, 782349: 3, 90048: 0, 94742: 0
    },
    82: {
        12982: 0, 13040: 0, 13257: 18, 13469: 4, 13629: 0, 13662: 6, 13696: 0, 160879: 6,
        180304: 0, 20632: 5, 250001: 0, 250093: 0, 502327: 30, 502347: 12, 502530: 5,
        502532: 0, 502579: 0, 781934: 24, 782203: 0, 782282: 0, 782297: 6, 782311: 0,
        782327: 0, 782335: 8, 782349: 5, 90048: 0, 94742: 0
    },
    169: {
        12982: 0, 13040: 0, 13257: 6, 13469: 0, 13629: 0, 13662: 0, 13696: 0, 160879: 0,
        180304: 0, 20632: 0, 250001: 0, 250093: 0, 502327: 0, 502347: 6, 502530: 1,
        502532: 0, 502579: 0, 781934: 6, 782203: 0, 782282: 0, 782297: 15, 782311: 0,
        782327: 15, 782335: 4, 782349: 0, 90048: 0, 94742: 6
    },
    173: {
        12982: 0, 13040: 6, 13257: 0, 13469: 0, 13629: 8, 13662: 0, 13696: 12, 160879: 0,
        180304: 0, 20632: 0, 250001: 20, 250093: 0, 502327: 0, 502347: 0, 502530: 0,
        502532: 10, 502579: 0, 781934: 4, 782203: 0, 782282: 0, 782297: 6, 782311: 3,
        782327: 0, 782335: 9, 782349: 16, 90048: 0, 94742: 0
```

```
    },
    176: {
        12982: 5, 13040: 2, 13257: 0, 13469: 0, 13629: 0, 13662: 0, 13696: 5, 160879: 0,
        180304: 8, 20632: 0, 250001: 0, 250093: 10, 502327: 1, 502347: 0, 502530: 0,
        502532: 0, 502579: 0, 781934: 3, 782203: 0, 782282: 6, 782297: 18, 782311: 0,
        782327: 24, 782335: 0, 782349: 3, 90048: 0, 94742: 0
    }
}
# ----------Bloque 6 minimo de sku por sucursal----------#

minimos_sku_sucursal = {
    (12982, 81): 2, (13040, 81): 0, (13257, 81): 2, (13469, 81): 1, (160879, 81): 6,
    (250001, 81): 4, (250093, 81): 3, (502327, 81): 2, (502347, 81): 3, (781934, 81): 4,
    (782203, 81): 15, (782297, 81): 16, (782311, 81): 2, (782327, 81): 14, (782335, 81): 5,
    (782349, 81): 3, (13662, 81): 0, (20632, 81): 0, (502530, 81): 0, (502579, 81): 0,
    (94742, 81): 0, (13696, 81): 0, (13629, 81): 0, (502532, 81): 0, (90048, 81): 0,
    (180304, 81): 0, (782282, 81): 0,

    (12982, 82): 4, (13257, 82): 18, (13469, 82): 4, (13662, 82): 6, (20632, 82): 5,
    (160879, 82): 6, (250093, 82): 0, (502327, 82): 10, (502347, 82): 12, (502530, 82): 5,
    (502579, 82): 0, (781934, 82): 8, (782297, 82): 6, (782335, 82): 8, (782349, 82): 5,
    (13040, 82): 0, (13469, 82): 4, (250001, 82): 0, (782203, 82): 0, (782311, 82): 0,
    (782327, 82): 0, (94742, 82): 0, (13696, 82): 0, (13629, 82): 0, (502532, 82): 0,
    (90048, 82): 0, (180304, 82): 0, (782282, 82): 0,

    (13257, 169): 6, (13696, 169): 8, (94742, 169): 6, (502327, 169): 2, (502347, 169): 6,
    (502530, 169): 1, (781934, 169): 4, (782297, 169): 15, (782327, 169): 15,
    (782335, 169): 4, (12982, 169): 0, (13040, 169): 0, (13469, 169): 0, (13629, 169): 0,
    (20632, 169): 0, (250001, 169): 0, (250093, 169): 0, (502579, 169): 0, (782203, 169): 0,
    (782311, 169): 0, (782349, 169): 0, (502532, 169): 0, (90048, 169): 0,
    (180304, 169): 0, (782282, 169): 0,
```

```
    (13040, 173): 6, (13629, 173): 8, (13696, 173): 12, (250001, 173): 20, (502532, 173): 4,
    (781934, 173): 4, (782297, 173): 6, (782311, 173): 6, (782335, 173): 9,
    (782349, 173): 16, (12982, 173): 0, (13257, 173): 0, (13469, 173): 0, (160879, 173): 0,
    (250093, 173): 0, (502327, 173): 0, (502347, 173): 0, (502530, 173): 0,
    (502579, 173): 0, (782203, 173): 0, (782327, 173): 0, (94742, 173): 0,
    (90048, 173): 0, (180304, 173): 0, (782282, 173): 0, (20632, 173): 0, (13662, 173): 0,

    (12982, 176): 2, (13040, 176): 2, (13629, 176): 0, (13696, 176): 2, (90048, 176): 0,
    (180304, 176): 8, (250093, 176): 10, (502327, 176): 2, (502347, 176): 0,
    (781934, 176): 2, (782282, 176): 6, (782297, 176): 18, (782327, 176): 24,
    (782349, 176): 3, (13257, 176): 0, (13469, 176): 0, (13662, 176): 0, (160879, 176): 0,
    (250001, 176): 0, (502530, 176): 0, (502532, 176): 0, (502579, 176): 0,
    (782203, 176): 0, (782311, 176): 0, (782335, 176): 0, (94742, 176): 0
}
#-----------------Bloque 7 Distancias--------------#

distancia = {
        (81, 82): 20, (82, 81): 20, (81, 169): 20, (169, 81): 20,
        (81, 173): 30, (173, 81): 30, (81, 176): 40, (176, 81): 40,
        (82, 169): 10, (169, 82): 10, (82, 173): 20, (173, 82): 20,
        (82, 176): 10, (176, 82): 10, (169, 173): 40, (173, 169): 40,
        (169, 176): 20, (176, 169): 20, (173, 176): 30, (176, 173): 30,
        (111, 81): 50, (81, 111): 50, (111, 82): 30, (82, 111): 30,
        (111, 169): 35, (169, 111): 35, (111, 173): 45, (173, 111): 45,
        (111, 176): 60, (176, 111): 60
    }

#------------------Bloque 8 Transferencias Permitidas---------#

transferencias_permitidas = [
```

```
    # Desde CD (111) a sucursales
    (111, 81),
    (111, 82),
    (111, 169),
    (111, 173),
    (111, 176),

    # Entre sucursales (bidireccionales)
    (81, 82), (82, 81),
    (81, 169), (169, 81),
    (81, 173), (173, 81),
    (81, 176), (176, 81),
    (82, 169), (169, 82),
    (82, 173), (173, 82),
    (82, 176), (176, 82),
    (169, 173), (173, 169),
    (169, 176), (176, 169),
    (173, 176), (176, 173)
  ]

# ------------Bloque 9  Capacaidad en sucursales ----------------

capacidad_sucursales = {s: 500 for s in sucursales}
capacidad_111 = {111:10000000}

# ------------Bloque 10  Cantidad minima por sku por scursal----------------

cantidad_minima_por_categoria = {cat: 0 for cat in set(categorias.values())}


# -------------Bloque 10 - Contribucion marginal por Sku -----------------
```

```
contribucion_marginal = {sku: 1000000 for sku in skus}

# ----------Bloque 11 - Peso de Sucursal ----------

peso_sucursal = {s: 1.0 for s in sucursales}

#---------Inputs Logisticos/ Negocio ----------

capacidad_camion = 10.0

costo_por_km = 150

costo_fijo_camion = 10000

carga_minima_por_camion = 0.5

# ==========

# 2) VARIABLES DE DECISIÓN

# ==========

x_deposito = LpVariable.dicts(

   "Envio_de_CD",

   ((sku, j) for j in sucursales for sku in skus if (111, j) in transferencias_permitidas),

   lowBound=0,

   cat='Integer'

)

x_inter = LpVariable.dicts(

   "Envio_entre_sucursales",

   ((sku, i, j) for i in sucursales for j in sucursales if i != j and (i, j) in transferencias_permitidas for
sku in skus),

   lowBound=0,

   cat='Integer'

)
```

```
insatisfaccion = LpVariable.dicts(
    "Insat",
    ((sku, suc) for sku in skus for suc in sucursales),
    lowBound=0,
    cat='Integer'
)

camiones = LpVariable.dicts(
    "Camiones",
    ((origen, destino) for origen in [111] + sucursales for destino in sucursales if origen != destino and (origen, destino) in transferencias_permitidas),
    lowBound=0,
    cat='Integer'
)

envio_bin = LpVariable.dicts(
    "Envio_bin",
    ((sku, i, j) for sku in skus for i in sucursales for j in sucursales if i != j and (i, j) in transferencias_permitidas),
    cat='Binary'
)



# ==========================
# FUNCIÓN OBJETIVO
# ==========================

modelo += lpSum(
    camiones[(origen, destino)] * (costo_fijo_camion + distancia[(origen, destino)] * costo_por_km)
    for (origen, destino) in camiones
```

```
) + lpSum(
    insatisfaccion[(sku, suc)] * contribucion_marginal[sku] * peso_sucursal[suc]
    for sku in skus for suc in sucursales
), "Costo_Total"

# ==========================
# RESTRICCIONES
# ==========================

# ==========================
# 1) RESTRICCIONES RELACIONADAS A CAMIONES
# ==========================

for (origen, destino) in camiones:

    volumen_total = lpSum(
        (x_deposito[(sku, destino)] if origen == 111 else x_inter[(sku, origen, destino)]) * volumen_sku[sku]
        for sku in skus
        if (origen, destino) in transferencias_permitidas
    )

    # Siempre: el volumen no puede exceder la capacidad total de los camiones activados
    modelo += volumen_total <= camiones[(origen, destino)] * capacidad_camion, f"Max_Carga_Camion_{origen}_{destino}"

    if carga_minima_por_camion > 0:
        # El camión solo se activa si transporta al menos un % de su capacidad
        modelo += volumen_total >= camiones[(origen, destino)] * capacidad_camion * carga_minima_por_camion, f"Min_Carga_Camion_{origen}_{destino}"
        modelo += camiones[(origen, destino)] <= volumen_total / (capacidad_camion * carga_minima_por_camion), f"No_Exceder_Camion_{origen}_{destino}"
```

```
    else:
        # Si no hay mínimo, se activa con cualquier volumen, pero debe haber algo
        modelo += camiones[(origen, destino)] >= volumen_total / (capacidad_camion + 1e-6), f"Activa_Camion_Si_Hay_Volumen_{origen}_{destino}"

# ==========================
# 2)EVITAR ENVÍOS CRUZADOS DE UN MISMO SKU ENTRE DOS SUCURSALES
# ==========================

for sku in skus:
    for i in sucursales:
        for j in sucursales:
            if i < j and (j, i) in transferencias_permitidas and (i, j) in transferencias_permitidas:
                modelo += envio_bin[(sku, i, j)] + envio_bin[(sku, j, i)] <= 1, f"NoCruzar_{sku}_{i}_{j}"
                modelo += x_inter[(sku, i, j)] <= 100000 * envio_bin[(sku, i, j)], f"ActivarBin_{sku}_{i}_{j}"
                modelo += x_inter[(sku, j, i)] <= 100000 * envio_bin[(sku, j, i)], f"ActivarBin_{sku}_{j}_{i}"

# ==========================
# 3) RESTRICCIONES DE CUMPLIMIENTO DE LA DEMANDA
# ==========================

for sku in skus:
    for suc in sucursales:
        entradas = x_deposito.get((sku, suc), 0) + lpSum(
            x_inter[(sku, i, suc)] for i in sucursales if i != suc and (suc, i) in transferencias_permitidas
        )
        stock_inicial = stock[sku].get(str(suc), 0)
        salidas = lpSum(
            x_inter[(sku, suc, j)] for j in sucursales if j != suc and (j, suc) in transferencias_permitidas
        )
```

```
    if sku in demanda.get(suc, {}):  # ← Esta línea es la clave
        modelo += entradas + stock_inicial - salidas + insatisfaccion[(sku, suc)] >= demanda[suc][sku], f"Demanda_{sku}_{suc}"

# ==========================
# 4) RESTRICCIONES DE CAPACIDAD EN SUCURSALES
# ==========================

for suc in sucursales:
    volumen_total_sucursal = lpSum(
        (
            stock[sku].get(str(suc), 0) +
            x_deposito.get((sku, suc), 0) +
            lpSum(x_inter[(sku, i, suc)] for i in sucursales if i != suc and (suc, i) in transferencias_permitidas) -
            lpSum(x_inter[(sku, suc, j)] for j in sucursales if j != suc and (j, suc) in transferencias_permitidas)
        ) * volumen_sku[sku]
        for sku in skus
    )
    modelo += volumen_total_sucursal <= capacidad_sucursales[suc], f"Capacidad_Sucursal_{suc}"

# ==========================
# 5)RESTRICCIONES DE MÍNIMOS POR SKU EN SUCURSALES
# ==========================

for (sku, suc), minimo in minimos_sku_sucursal.items():

    stock_inicial_minimos = stock[sku].get(str(suc), 0)
```

```
    entradas_cd_minimos = x_deposito.get((sku, suc), 0)

    entradas_suc_minimos = lpSum(
        x_inter[(sku, i, suc)]
        for i in sucursales if i != suc and (i, suc) in transferencias_permitidas
    )

    salidas_suc_minimos = lpSum(
        x_inter[(sku, suc, j)]
        for j in sucursales if j != suc and (suc, j) in transferencias_permitidas
    )

    cantidad_total_utilizable_minimos = stock_inicial_minimos + entradas_cd_minimos + entradas_suc_minimos - salidas_suc_minimos

    modelo += cantidad_total_utilizable_minimos >= minimo, f"Min_Sku_{sku}_{suc}"

# ==========================
# 6) RESTRICCIONES DE MÍNIMOS POR CATEGORÍA EN SUCURSALES
# ==========================

for categoria, minimo in cantidad_minima_por_categoria.items():
    skus_cat = [sku for sku in skus if categorias[sku] == categoria]

    for suc in sucursales:

        cantidad_total_utilizable_sucursales = lpSum(
            stock[sku].get(str(suc), 0) +  # stock inicial
            x_deposito.get((sku, suc), 0) +  # entradas desde el CD
            lpSum(x_inter[(sku, i, suc)] for i in sucursales if i != suc and (i, suc) in transferencias_permitidas) -  # entradas desde otras sucursales
```

```
        lpSum(x_inter[(sku, suc, j)] for j in sucursales if j != suc and (suc, j) in transferencias_permitidas)  # salidas hacia otras sucursales
        for sku in skus_cat
    )

    modelo += cantidad_total_utilizable_sucursales >= minimo, f"Min_Categoria_{categoria}_{suc}"

# ==============================
# 7) RESTRICCIÓN DE CONSERVACIÓN DE STOCK
# ==============================

for sku in skus:

  # 1️⃣ STOCK TOTAL INICIAL EN TODO EL SISTEMA (CD + SUCURSALES)
  stock_total_inicial = stock[sku].get("111", 0) + sum(
    stock[sku].get(str(suc), 0) for suc in sucursales
  )

  # 2️⃣ MOVIMIENTOS TOTALES SALIENTES DEL SISTEMA (CD + ENTRE SUCURSALES)
  movimientos_totales = lpSum([
    x_deposito[(sku, j)]
    for j in sucursales
    if (111, j) in transferencias_permitidas and (sku, j) in x_deposito
  ]) + lpSum([
    x_inter[(sku, i, j)]
    for i in sucursales
    for j in sucursales
    if i != j and (i, j) in transferencias_permitidas and (sku, i, j) in x_inter
  ])
```

```
    # 3️⃣ RESTRICCIÓN GLOBAL → NO INVENTAR UNIDADES
    modelo += stock_total_inicial >= movimientos_totales, f"Conservacion_Global_Total_Stock_{sku}"

    # 4️⃣ RESTRICCIÓN LOCAL → CD NO ENVÍA MÁS DE LO QUE TIENE
    salidas_desde_cd = lpSum(
        x_deposito[(sku, j)]
        for j in sucursales
        if (111, j) in transferencias_permitidas and (sku, j) in x_deposito
    )
    modelo += salidas_desde_cd <= stock[sku].get("111", 0), f"Conservacion_CD_{sku}"

    # 5️⃣ RESTRICCIONES POR SUCURSAL → CADA UNA NO ENVÍA MÁS DE LO QUE TIENE + LE LLEGA
    for suc in sucursales:

        stock_real = stock[sku].get(str(suc), 0)

        entradas_cd = x_deposito[(sku, suc)] if (sku, suc) in x_deposito else 0

        entradas_sucursales = lpSum(
            x_inter[(sku, i, suc)]
            for i in sucursales
            if i != suc and (i, suc) in transferencias_permitidas and (sku, i, suc) in x_inter
        )

        salidas_sucursales = lpSum(
            x_inter[(sku, suc, j)]
            for j in sucursales
            if j != suc and (suc, j) in transferencias_permitidas and (sku, suc, j) in x_inter
        )
```

```
        modelo += (
            stock_real + entradas_cd + entradas_sucursales >= salidas_sucursales
        ), f"Conservacion_Sucursal_{sku}_{suc}"


# ==============================
#  BLOQUE DE RESOLUCIÓN DEL MODELO
# ==============================

modelo.solve()

print(f"Estado del modelo: {LpStatus[modelo.status]}")
print(f"Valor óptimo: {value(modelo.objective):,.2f}")
if LpStatus[modelo.status] != "Optimal":
    print(" Atención: el modelo no tiene solución factible. No se deben interpretar los resultados como válidos.")

# ==============================
#  COMPROBACIONES DE RESULTADOS
# ==============================

# 1) Costo total desagregado
costo_variable = sum(
    value(camiones[(origen, destino)]) * costo_por_km * distancia[(origen, destino)]
    for (origen, destino) in camiones
)
costo_fijo = sum(
    value(camiones[(origen, destino)]) * costo_fijo_camion
    for (origen, destino) in camiones
)
costo_logistico = costo_variable + costo_fijo
```

```
costo_penalizaciones = sum(
    value(insatisfaccion[(sku, suc)]) * contribucion_marginal[sku] * peso_sucursal[suc]
    for sku in skus for suc in sucursales
)
print("\n COSTOS TOTALES:")
print(f"  → Costo logístico total: ARS {costo_logistico:,.2f}")
print(f"     • Costo fijo camiones: ARS {costo_fijo:,.2f}")
print(f"     • Costo variable por km: ARS {costo_variable:,.2f}")
print(f"  → Penalización por insatisfacción: ARS {costo_penalizaciones:,.2f}")
print(f"  → Costo total: ARS {costo_logistico + costo_penalizaciones:,.2f}")



# 2) Resumen de sku por sucursal


resumen = []


for suc in sucursales:
    for sku in skus:
        stock_inicial = stock[sku].get(str(suc), 0)
        entradas_cd = value(x_deposito[(sku, suc)]) if (sku, suc) in x_deposito else 0
        entradas_suc = sum(value(x_inter[(sku, i, suc)]) for i in sucursales if i != suc and (i, suc) in
transferencias_permitidas and (sku, i, suc) in x_inter)
        salidas_suc = sum(value(x_inter[(sku, suc, j)]) for j in sucursales if j != suc and (suc, j) in
transferencias_permitidas and (sku, suc, j) in x_inter)
        total_disponible = stock_inicial + entradas_cd + entradas_suc - salidas_suc
        demanda_sku = demanda.get(suc, {}).get(sku, 0)


        minimo = minimos_sku_sucursal.get((sku, suc), 0)
        cumple_minimo = "ok" if total_disponible >= minimo else "not ok"


        unidades_no_satisfechas = value(insatisfaccion[(sku, suc)]) if (sku, suc) in insatisfaccion else
0
```

```
        demanda_satisfecha = "ok" if unidades_no_satisfechas == 0 else "not ok"

        resumen.append({
            "Sucursal": suc,
            "SKU": sku,
            "Stock inicial": stock_inicial,
            "Entradas desde CD": entradas_cd,
            "Entradas desde sucursales": entradas_suc,
            "Salidas hacia sucursales": salidas_suc,
            "Total disponible final": total_disponible,
            "Demanda": demanda_sku,
            "Mínimo requerido": minimo,
            "¿Cumple mínimo?": cumple_minimo,
            "¿Demanda satisfecha?": demanda_satisfecha,
            "Unidades no satisfechas": int(unidades_no_satisfechas)
        })

df_resumen = pd.DataFrame(resumen)
print(df_resumen)

# 3)
# a) Resumen Camiones utilizados y costos

# Crear resumen de camiones utilizados con desglose de costos
resumen_camiones = []

for (origen, destino), var in camiones.items():
    cantidad_usada = value(var)
    if cantidad_usada > 0:
        dist_km = distancia[(origen, destino)]
        costo_fijo_total = cantidad_usada * costo_fijo_camion
```

```
        costo_variable_total = cantidad_usada * dist_km * costo_por_km
        costo_total = costo_fijo_total + costo_variable_total

        resumen_camiones.append({
            "Ruta": f"{origen} → {destino}",
            "Camiones usados": int(cantidad_usada),
            "Costo fijo total": costo_fijo_total,
            "Costo variable total": costo_variable_total,
            "Costo total ruta": costo_total
        })

df_camiones = pd.DataFrame(resumen_camiones)
print(df_camiones)

# b) Resumen de capacidades de camiones utilizados

datos_camiones = []

for (origen, destino) in camiones:
    cantidad_camiones = value(camiones[(origen, destino)])
    if cantidad_camiones > 0:
        # Volumen transportado en esa ruta
        volumen_total = sum(
            (value(x_deposito[(sku, destino)]) if origen == 111 else value(x_inter[(sku, origen, destino)])) * volumen_sku[sku]
            for sku in skus
            if (origen, destino) in transferencias_permitidas
        )

        capacidad_total = cantidad_camiones * capacidad_camion
        eficiencia = volumen_total / capacidad_total if capacidad_total > 0 else 0
```

```
        cumple_minimo = "ok" if eficiencia >= carga_minima_por_camion else "not ok"

        datos_camiones.append({
            "Ruta": f"{origen} → {destino}",
            "Camiones utilizados": int(cantidad_camiones),
            "Volumen total transportado (m³)": volumen_total,
            "Capacidad total contratada (m³)": capacidad_total,
            "Eficiencia de llenado (%)": round(eficiencia * 100, 2),
            "¿Cumple mínimo requerido?": cumple_minimo
        })

df_camiones_eficiencia = pd.DataFrame(datos_camiones)
print(df_camiones_eficiencia)

# 4) Resumen de categorias

# Crear tabla de mínimos por categoría
resumen_categoria = []

for suc in sucursales:
    for categoria, minimo in cantidad_minima_por_categoria.items():
        skus_cat = [sku for sku in skus if categorias[sku] == categoria]

        total_utilizado = sum(
            stock[sku].get(str(suc), 0) +
            value(x_deposito.get((sku, suc), 0)) +
            sum(value(x_inter.get((sku, i, suc), 0)) for i in sucursales if i != suc) -
            sum(value(x_inter.get((sku, suc, j), 0)) for j in sucursales if j != suc)
            for sku in skus_cat
        )
```

```
        cumple = "ok" if total_utilizado >= minimo else "not ok"

        resumen_categoria.append({
            "Sucursal": suc,
            "Categoría": categoria,
            "Total recibido/utilizable": total_utilizado,
            "Mínimo requerido": minimo,
            "¿Cumple mínimo?": cumple
        })

# Convertir a DataFrame
df_resumen_categoria = pd.DataFrame(resumen_categoria)
print(df_resumen_categoria)



# 5) Volumnen final por cd y por sucursal

ubicaciones = sucursales + [111]
capacidad_total = {**capacidad_sucursales, **capacidad_111}

resumen_volumen = []

for loc in ubicaciones:
    volumen_inicial = sum(stock[sku].get(str(loc), 0) * volumen_sku[sku] for sku in skus)

    entradas_cd = sum(
        value(x_deposito[(sku, loc)]) * volumen_sku[sku]
        for sku in skus
        if (sku, loc) in x_deposito and loc != 111
    ) if loc != 111 else 0
```

```
    entradas_suc = sum(
        value(x_inter[(sku, i, loc)]) * volumen_sku[sku]
        for sku in skus for i in sucursales
        if i != loc and (sku, i, loc) in x_inter
    )

    salidas_suc = sum(
        value(x_inter[(sku, loc, j)]) * volumen_sku[sku]
        for sku in skus for j in sucursales
        if j != loc and (sku, loc, j) in x_inter
    )

    salidas_cd = sum(
        value(x_deposito[(sku, j)]) * volumen_sku[sku]
        for sku in skus for j in sucursales
        if (sku, j) in x_deposito and loc == 111
    ) if loc == 111 else 0

    volumen_final = volumen_inicial + entradas_cd + entradas_suc - salidas_suc - salidas_cd
    capacidad = capacidad_total.get(loc, 0)

    resumen_volumen.append({
        "Ubicación": loc,
        "Volumen inicial": volumen_inicial,
        "Volumen final": volumen_final,
        "Capacidad": capacidad,
        "¿Excede capacidad?": "Warning" if volumen_final > capacidad else "ok",
        "¿Volumen final positivo?": "ok" if volumen_final >= 0 else "not ok"
    })

import pandas as pd
```

```
df_volumen = pd.DataFrame(resumen_volumen)
print(df_volumen)

# 6) Analisis de envios cruzados:

conflictos = []

# Solo recorremos cada par una vez (i < j)
for sku in skus:
    for i in sucursales:
        for j in sucursales:
            if i < j and (i, j) in transferencias_permitidas and (j, i) in transferencias_permitidas:
                envio_ij = value(x_inter[(sku, i, j)])
                envio_ji = value(x_inter[(sku, j, i)])
                if envio_ij > 0 and envio_ji > 0:
                    conflictos.append({
                        "SKU": sku,
                        "Sucursal A → B": f"{i} → {j}",
                        "Unidades A → B": envio_ij,
                        "Sucursal B → A": f"{j} → {i}",
                        "Unidades B → A": envio_ji,
                        "¿Cruce detectado?": " Sí"
                    })

# Convertimos a DataFrame
df_cruces = pd.DataFrame(conflictos)


if df_cruces.empty:
    print(" No se detectaron envíos cruzados entre pares de sucursales para ningún SKU.")
else:
```

```
    print(df_cruces)



# ================================

#  EXPORTACION DE RESULTADOS

# ================================


# 2️⃣ Nombre del archivo a exportar (podés cambiarlo si querés)

nombre_archivo = "Resumen_Optimizacion_Modelo 25 Sin datos de minimo de Sku-05-en-rstriccion.xlsx"


# 3️⃣ Exportar a múltiples hojas en un único archivo Excel

with pd.ExcelWriter(nombre_archivo, engine="openpyxl") as writer:

    df_resumen.to_excel(writer, sheet_name="Resumen_SKU_Sucursal", index=False)

    df_camiones.to_excel(writer, sheet_name="Resumen_Rutas_Costos", index=False)

    df_resumen_categoria.to_excel(writer, sheet_name="Resumen_categorias", index=False)

    df_volumen.to_excel(writer, sheet_name="Resumen_volumen", index=False)

    df_camiones_eficiencia.to_excel(writer, sheet_name="Resumen_camiones_eficiencia", index=False)

print(f" Archivo exportado correctamente como: {nombre_archivo}")
```

ANEXO 3: RESULTADOS MODELO 11

Nos parece importante destacar que se hicieron diferentes pruebas en este modelo que nos llevaron a encontrar errores que fueron claves y terminaron derivando en la versión final del modelo (modelo 22). Se plantearon 4 escenarios base que buscaron forzar ciertas situaciones para entender cómo funciona el modelo y asegurar que su generación sea óptima.

1) Escenario Base
2) Escenario Capacidad insuficiente en sucursal 81
3) No hay stock disponible para cubrir la demanda
4) Escenario con datos de predicción

Antes de avanzar con la explicación de la resolución de los escenarios vamos a definir cuáles fueron los parámetros que se tomaron como bases comunes utilizados en todos los escenarios (que se encuentran obviamente listados a nivel de código):

- Las sucursales 81 y 82, las únicas que se toman en cuenta en este modelo, tienen una capacidad máxima de almacenamiento de 500 m^3 cada una.
- Se trabajan con los SKUs 1001 a 1004, cada uno con un volumen asignado entre 0.3 m^3 y 5.0 m^3 por unidad.
- El costo fijo por camión es de ARS 10.000.
- El costo variable por kilómetro es de ARS 150.
- Las distancias definidas son: 100 km del CD a la sucursal 81, 100 km del CD a la sucursal 82, y 12 km entre la sucursal 81 y la 82.
- La carga mínima exigida por camión es del 80% de su capacidad.
- Se establecen mínimos obligatorios por categoría (10 unidades) y por SKU killer (entre 5 y 50 unidades según el caso).

Escenario 1 – Base

Análisis de resultados escenario base 1

Este escenario representa el funcionamiento del modelo bajo condiciones estándar: disponibilidad total de stock en el centro de distribución limitada para ciertos casos, capacidad operativa completa en ambas sucursales, y una demanda planteada sin alteraciones extremas. El objetivo fue evaluar si el modelo logra distribuir eficientemente el stock mientras satisface la demanda (hasta donde corresponde) y cumple con las restricciones de mínimos, sin estar condicionado por cuellos de botella.

El modelo encontró una solución óptima, con un costo total de ARS 708.400, desglosado en ARS 185.400 de costo logístico (ARS 90.000 de costo fijo por camiones y ARS 95.400 por kilómetro recorrido), y ARS 523.000 de penalizaciones por demanda no satisfecha. A pesar de las condiciones favorables, la alta penalización se debe a la falta de unidades en el CD para cubrir la demanda completa.

Se utilizaron 9 camiones en total: 5 saliendo del CD hacia la sucursal 81, 1 desde el CD a la sucursal 82, 1 desde la sucursal 81 hacia la 82 y 2 desde la 82 hacia la 81. El sistema alcanzó una eficiencia promedio de llenado del 99,78%, cumpliendo con el mínimo promedio de carga por camión.

Se plantea a continuación un ejemplo del desglose del costo por insatisfacción para tomar como parámetro a nivel de expresión de resultados finales:

- SKU 1001 en sucursal 81: Se requerían 300 unidades y se enviaron 50. Faltaron 250 unidades. Motivo: falta de stock en el CD.
- SKU 1001 en sucursal 82: Se requerían 300 unidades y no se enviaron unidades. Motivo: falta de stock en el CD.
- SKU 1002 en sucursal 81: Se requerían 150 unidades y se enviaron 50. Faltaron 100 unidades. Motivo: falta de stock.
- SKU 1002 en sucursal 82: Se requerían 90 unidades y se enviaron 70. Faltaron 20 unidades. Motivo: falta de stock.
- SKU 1003 en sucursal 81: Se requerían 70 unidades y se enviaron 66. Faltaron 4 unidades. Motivo: stock insuficiente a nivel general en el sistema
- SKU 1004 en sucursal 82: Se requerían 40 unidades y se enviaron 25. Faltaron 15 unidades. Motivo: stock insuficiente.

Ninguno de los faltantes se debió a capacidad logística limitada o a altos costos de envío. Todo estuvo determinado por restricciones de stock. **Los resultados fueron los esperados, el modelo se comportó de manera correcta en función a las limitaciones planteadas.**

Escenario 2 – Capacidad insuficiente en sucursal 81

Parámetros específicos del escenario:

- Demanda en sucursal 81 para SKU 1001: 1000 unidades.
- Capacidad en sucursal 81: limitada a 50 m^3.
- Capacidad en sucursal 82: normal (500 m^3).
- Objetivo: validar que el modelo identifique que no puede enviar toda la demanda a sucursal 81 debido a falta de espacio, y observe cómo se prioriza el cumplimiento parcial bajo esa condición.

Análisis de resultados escenario base 2

En este escenario se forzó al modelo a resolver un caso extremo: enviar 1000 unidades del SKU 1001 a la sucursal 81, la cual solo puede recibir un máximo de 50 m^3. Como el volumen por unidad del SKU 1001 es de 1 m^3, era físicamente imposible cumplir con la demanda. El modelo respondió correctamente, encontrando una solución óptima, con un costo total de ARS 1.239.600 compuesto por ARS 234.000 de costo logístico (ARS 120.000 de costos fijos por camión y ARS 114.000 de costos variables) y ARS 1.005.600 de penalización por insatisfacción.

Se utilizaron 12 camiones, de los cuales 7 fueron despachados del CD hacia la sucursal 82 y 5 realizaron traslados desde la 82 hacia la 81. No se realizaron envíos directos a la sucursal 81 desde el CD. El sistema alcanzó una eficiencia de llenado de camiones del 98,33%, y se cumplieron todos los mínimos exigidos por categoría y por SKU killer.

Con este escenario, se probó de forma clara que el modelo responde adecuadamente cuando la restricción de espacio se convierte en el principal factor limitante. Además, el modelo logra cumplir con los mínimos requeridos, aún priorizando el uso del espacio disponible de forma eficiente.

Ejemplo del desglose del costo por insatisfacción planteado por el modelo:

- Sucursal 81, SKU 1001: Demanda de 1000 unidades. Solo se enviaron 10. Faltaron 990 unidades.

  Motivo: falta de espacio físico en la sucursal, ya que el SKU 1001 ocupa 1 m^3 por unidad y la capacidad era de 50 m^3.

- Sucursal 82, SKU 1001: Demanda de 10 unidades. No se enviaron unidades.
  Motivo: priorización de envío hacia la sucursal 81 y agotamiento del stock

disponible en el CD.

- Sucursal 81, SKU 1002: Demanda de 5 unidades. No se enviaron unidades. Motivo: falta de espacio en la sucursal (los 50 m^3 disponibles ya estaban ocupados por el SKU 1001).

- Sucursal 82, SKU 1002: Demanda de 5 unidades. Se enviaron 3 unidades. Motivo: stock limitado disponible, luego de priorizar SKU 1001.

Escenario 3 No hay stock disponible para cubrir la demanda.Parámetros específicos del escenario:

- Stock disponible en el CD: 0 unidades
- Stock en sucursales: solo cantidades mínimas
- Capacidad de almacenamiento: sin restricciones
- Demanda en sucursales: normal
- Objetivo: comprobar cómo responde el modelo ante un escenario sin disponibilidad real de mercadería

Descripción general:

Este escenario fue diseñado para probar la capacidad del modelo de responder ante una **ausencia total de stock en el centro de distribución**, y una red de sucursales sin inventario suficiente para cubrir la demanda. El objetivo era que el modelo evaluará correctamente la imposibilidad de cumplimiento y prioriza la insatisfacción de forma estructurada, sin movimientos innecesarios.

El modelo devuelve una **solución técnicamente no factible** (estado "Infeasible") tal cual se esperaba y se puede ver en la **figura 14**.

Desglose del costo por insatisfacción:

- Sucursal 81:
    - SKU 1001: faltaron 20 unidades
    - SKU 1002: faltaron 20 unidades
- Sucursal 82:

- SKU 1001: faltaron 20 unidades
- SKU 1002: faltaron 20 unidades

Motivo principal: ausencia total de stock para satisfacer la demanda. No hubo disponibilidad operativa ni en el CD ni en las sucursales.

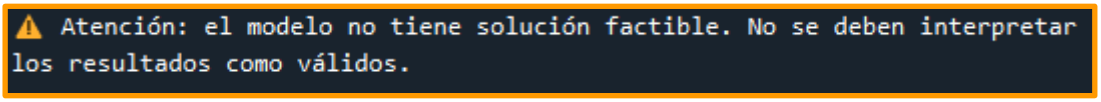

Figura 14: Resultado arrojado por modelo ante la corrida del script para un modelo no factible.

Escenario 4 - Prueba del modelo 12 con información asociada al modelo de predicción Prophet.

Dado que hasta ese momento veíamos al modelo robusto decidimos continuar cargando la información asociada al output del modelo predictivo.

Las modificaciones de datos, la inclusión de sucursales y también el aumento de la cantidad de Skus terminó derivando en errores que no habíamos contemplado como:

-Envíos cíclicos.
-creación de unidades fantasmas.

Estos errores se pueden ver en **la figura 15**:

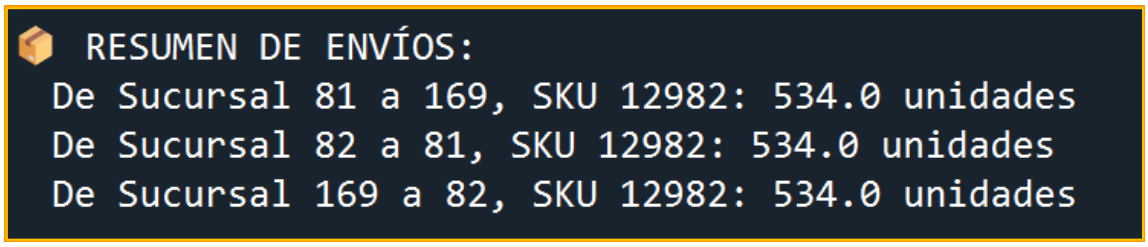

Figura 15: Resultado arrojado por modelo que demuestra envíos para un SKU especifico generando unidades que no existen.

Cuando cómo se puede ver claramente en la **figura 16** asociado a la creación del modelo la cantidad de unidades que teníamos inicialmente del sku 12982 era mucho más pequeña tanto en 111 como en las sucursales.

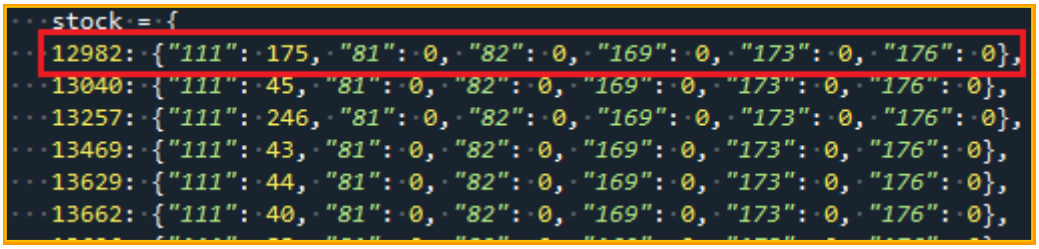

Figura 16: Figura que demuestra que el stock inicial para 12982 no puede ser 534 unidades.

Revisando el modelo a detalle existían varios puntos que fueron necesario adaptar a lo largo de otros 10 modelos, errores en como técnicamente estaban implementadas las restricciones que ataban las variables de camiones con las variables a enviar desde el centro de distribución y de las sucursales, errores asociados a cómo se trataba de limitar la cantidad de mercadería que podía salir de una sucursal a otra y desde el depósito y también había errores en hasta cómo se habían definido las variables de decisión.

ANEXO 4:EJEMPLOS DE CAMBIO EN EL CÓDIGO ENTRE MODELO 11 Y MODELO 12

Restricción 1 – Uso de camiones

Modelo 11: Requiere que el volumen transportado en una ruta sea al menos el 50% de la capacidad del camión (carga mínima), lo cual fuerza al modelo a llenar cada camión al menos hasta ese porcentaje. No permite enviar camiones por debajo de ese mínimo.

El problema de esta restricción fue que por la manera en la cual estaba construida a nivel de código no terminaba de activar camiones en la función objetivo, esto generaba un problema particular en donde existía envío de unidades desde un centro de distribución a una sucursal o entre sucursal (dado que las variables de decisión que miden las unidades que se enviaban por ruta de un sku se activaban) pero sin la necesidad de transportar camiones.

Modelo 12: Introduce una lógica bifurcada, tal cual se puede ver en **la figura 24** si se define una carga mínima, se controla la activación con eficiencia; si no, se permite enviar camiones parcialmente llenos pero solo si hay volumen. Esta flexibilidad permite probar distintos escenarios sin romper el modelo, y mejora la precisión del cálculo de costos.

La modificación del código fue clave para lograr que se activen correctamente los costos a nivel de función objetivo, básicamente esta restricción permitió seguir correctamente con el modelado de la solución del problema.

```
# ========================
# 1) RESTRICCIONES RELACIONADAS A CAMIONES
# ========================

for (origen, destino) in camiones:

    volumen_total = lpSum(
        (x_deposito[(sku, destino)] if origen == 111 else x_inter[(sku, origen, destino)]) * volumen_sku[sku]
        for sku in skus
        if (origen, destino) in transferencias_permitidas
    )

    # Siempre: el volumen no puede exceder la capacidad total de los camiones activados
    modelo += volumen_total <= camiones[(origen, destino)] * capacidad_camion, f"Max_Carga_Camion_{origen}_{destino}"

    if carga_minima_por_camion > 0:
        # El camión solo se activa si transporta al menos un % de su capacidad
        modelo += volumen_total >= camiones[(origen, destino)] * capacidad_camion * carga_minima_por_camion, f"Min_Carga_Camion_{origen}_{destino}"
        modelo += camiones[(origen, destino)] <= volumen_total / (capacidad_camion * carga_minima_por_camion), f"No_Exceder_Camion_{origen}_{destino}"
    else:
        # Si no hay mínimo, se activa con cualquier volumen, pero debe haber algo
        modelo += camiones[(origen, destino)] >= volumen_total / (capacidad_camion + 1e-6), f"Activa_Camion_Si_Hay_Volumen_{origen}_{destino}"
```

Figura 24: Lógica de código asociada a restricción de activación de camiones.

La restricción en sí cuenta con diferentes sentencias, para poder explicarla de manera eficiente pensemos en el siguiente ejemplo:

Desde el centro de distribución (111) queremos enviar 4 unidades del Sku “A” con un volumen unitario de 1m3 y 6 unidades del Sku “B” con un volumen unitario de 1m3 (un volumen total de 10m3) a la sucursal 81. La variable de decisión “camiones” en este caso es una variable que se activa por ruta (en este caso sería camiones [(111,81)]) y la capacidad mínima del camión 0,8 (80% del camión tiene que estar utilizado para decidir que un camión salga).

En primer lugar, calcula el volumen total de todas las unidades enviadas desde un depósito a una sucursal y también entre sucursales siempre y cuando las transferencias se den dentro de rutas permitidas.

Luego plantea una primera restricción al modelado del problema planteado que ningún camión activado pueda transportar más cantidad de mercadería de la que puede llevar, este caso 10 ≤ camiones [(111,81)] * 10 , mínimo 1 camión.

La segunda parte de la restricción asegura que los camiones **no viajen casi vacíos**, es decir, que, si se activa un camión, esté al menos un 80% lleno. En nuestro ejemplo: 10 ≥ 2 * 10 * 0.8 → 10 ≥ 16 falla, entonces el modelo evitará enviar 2 camiones y preferiría mandar 1 solo.

La tercera parte del modelo evita que el modelo active más camiones si no hay volumen para cumplir el mínimo de carga en todos ellos. En nuestro ejemplo: camiones ≤ 10 / (10 * 0.8) = 1.25, entonces se redondea a un camión.

Por último, también existe una cláusula que asegura que si se envía algo, se active **al menos un camión** (aunque no haya mínimo de carga).

Restricción 7 – Conservación de stock

Modelo 11: Utiliza una serie de restricciones para evitar que las sucursales o el CD terminen con stock negativo. Sin embargo, estas restricciones eran locales y no aseguraban la conservación total del stock del sistema, lo cual permitía la creación ficticia de unidades.

Modelo 12: Agrega una restricción de conservación **global**, tal cual se puede ver en las **figura 25 y 26**, que asegura que el total de unidades enviadas (desde el CD y entre sucursales) no exceda el stock total inicial del sistema. A su vez, mantiene restricciones locales que aseguran que ni el CD ni una sucursal envíen más de lo que tienen realmente disponible.

```
# ==============================
# 7) RESTRICCIÓN DE CONSERVACIÓN DE STOCK
# ==============================

for sku in skus:

    # STOCK TOTAL INICIAL EN TODO EL SISTEMA (CD + SUCURSALES)
    stock_total_inicial = stock[sku].get("111", 0) + sum(
        stock[sku].get(str(suc), 0) for suc in sucursales
    )

    # MOVIMIENTOS TOTALES SALIENTES DEL SISTEMA (CD + ENTRE SUCURSALES)
    movimientos_totales = lpSum([
        x_deposito[(sku, j)]
        for j in sucursales
        if (111, j) in transferencias_permitidas and (sku, j) in x_deposito
    ]) + lpSum([
        x_inter[(sku, i, j)]
        for i in sucursales
        for j in sucursales
        if i != j and (i, j) in transferencias_permitidas and (sku, i, j) in x_inter
    ])

    # RESTRICCIÓN GLOBAL → NO INVENTAR UNIDADES
    modelo += stock_total_inicial >= movimientos_totales, f"Conservacion_Global_Total_Stock_{sku}"
```

Figura 25: Lógica de código asociada a restricción de activación de camiones. Definición de Stock y movimientos logísticos.

La primera parte del código se encarga de consolidar cantidades iniciales de Stock y movimientos salientes del sistema (CD a sucursales y entre sucursales).

Luego se plantea una primera restricción global de conservación general del stock que busca evitar que existan más movimientos de cantidades de las permitidas por el stock inicial del problema. Esto evita que se “inventen unidades” como sucedía en el modelo 11.

```
    # 4. RESTRICCIÓN LOCAL → CD NO ENVÍA MÁS DE LO QUE TIENE
    salidas_desde_cd = lpSum(
        x_deposito[(sku, j)]
        for j in sucursales
        if (111, j) in transferencias_permitidas and (sku, j) in x_deposito
    )
    modelo += salidas_desde_cd <= stock[sku].get("111", 0), f"Conservacion_CD_{sku}"

    # 5. RESTRICCIONES POR SUCURSAL → CADA UNA NO ENVÍA MÁS DE LO QUE TIENE + LE LLEGA
    for suc in sucursales:

        stock_real = stock[sku].get(str(suc), 0)

        entradas_cd = x_deposito[(sku, suc)] if (sku, suc) in x_deposito else 0

        entradas_sucursales = lpSum(
            x_inter[(sku, i, suc)]
            for i in sucursales
            if i != suc and (i, suc) in transferencias_permitidas and (sku, i, suc) in x_inter
        )

        salidas_sucursales = lpSum(
            x_inter[(sku, suc, j)]
            for j in sucursales
            if j != suc and (suc, j) in transferencias_permitidas and (sku, suc, j) in x_inter
        )

        modelo += (
            stock_real + entradas_cd + entradas_sucursales >= salidas_sucursales
        ), f"Conservacion_Sucursal_{sku}_{suc}"
```

Figura 26: Lógica de código asociada a restricción de activación de camiones. Conservación de Stock.

En segundo lugar se plantea una restricción local para el centro de distribución, dado que es un nodo que únicamente puede enviar mercadería se le pide al modelo que se asegure de no mandar más cantidad de la que tiene en un inicio.

Finalmente se plantea el vínculo final para la conservación de stock haciendo algo parecido a lo que se hace en la segunda restricción para el centro de distribución con la particularidad de que acá se aborda la problemática de las sucursales exigiendo al modelo de que cada sucursal no puede enviar más de lo que tiene, le ingresa desde el centro de distribución y otras sucursales.

En función a estos cambios se logró una integridad general en el modelo que permitió avanzar al modelo 22 donde se logró correr efectivamente el problema con datos reales.