Facultad de Ciencias Económicas y Empresariales
ICADE

ZIPO: Plataforma de optimización de inventarios para restaurantes mediante analítica de datos

Autor: Diego Gutiérrez-Colomer Beneyto

Tutor: Carlos Bellón Núñez-Mera

Grado en Business Analytics

Resumen ejecutivo

Este trabajo desarrolla la parte analítica y tecnológica de Zipo, una plataforma SaaS pensada para que los restaurantes independientes gestionen su inventario, anticipen la demanda y automaticen sus pedidos desde un único sitio. El sector se caracteriza por márgenes muy ajustados y una digitalización escasa de la gestión interna, lo que convierte el desperdicio alimentario y las roturas de stock en problemas con impacto económico directo. A partir de ese diagnóstico, el trabajo define el modelo de negocio de la plataforma, diseña su arquitectura en seis módulos y desarrolla por completo el módulo de predicción de demanda. Para ello se construye un pipeline en Python que, sobre un dataset sintético que simula un restaurante, predice la demanda de cada producto con el modelo Prophet y genera de forma automática las alertas de stock y la orden de compra. Los resultados muestran que la combinación de una predicción interpretable con un sistema de alertas basado en márgenes de seguridad produce recomendaciones de compra coherentes y con valor económico para el restaurante. El trabajo concluye que una herramienta así es viable con tecnologías maduras y de código abierto, y señala como principal limitación la ausencia de validación con datos reales.

Palabras clave: gestión de inventario, predicción de demanda, restauración, SaaS, Prophet, desperdicio alimentario.

Abstract

This project develops the analytical and technological side of Zipo, a SaaS platform designed to help independent restaurants manage their inventory, anticipate demand and automate their orders from a single place. The sector is characterised by very tight margins and limited digitalisation of internal management, which turns food waste and stockouts into problems with a direct economic impact. Building on this diagnosis, the project defines the platform's business model, designs its architecture across six modules and fully develops the demand forecasting module. To this end, a Python pipeline is built that, using a synthetic dataset simulating a restaurant, forecasts the demand for each product with the Prophet model and automatically generates stock alerts and the purchase order. The results show that combining

an interpretable forecast with an alert system based on safety margins produces coherent purchasing recommendations with economic value for the restaurant. The project concludes that such a tool is feasible with mature, open-source technologies, and identifies the lack of validation with real data as its main limitation.

Keywords: inventory management, demand forecasting, restaurants, SaaS, Prophet, food waste.

Índice

Capítulo 1. Introducción

1.1 Contexto del sector de la restauración

Este trabajo nace de una pregunta concreta: ¿por qué la mayoría de los restaurantes independientes siguen gestionando su inventario con una libreta, una hoja de cálculo o simplemente de memoria? El problema no es que los dueños no quieran hacerlo mejor, sino que las herramientas que existen están pensadas para grandes cadenas, son caras de implantar o requieren conocimientos técnicos que un restaurante pequeño no tiene. Este es el punto de partida de Zipo y el contexto en el que se enmarca este TFG.

La industria de la restauración en España representa uno de los pilares fundamentales de la economía nacional. Según Hostelería de España (2023), el sector agrupa más de 240.000 establecimientos activos y supone aproximadamente el 6,2% del PIB, siendo además uno de los mayores empleadores del país con más de 1,7 millones de trabajadores directos. A nivel europeo, la restauración genera en torno a 10 millones de empleos y factura más de 350.000 millones de euros anuales (EFFAT, 2022). Más del 95% de estos establecimientos en España tienen menos de 10 empleados (INE, 2023): son restaurantes independientes, bares de toda la vida y pequeñas cadenas familiares. Son, también, el cliente al que va dirigido Zipo.

A pesar de su peso económico y social, el sector tiene desde hace tiempo un problema de baja productividad. Los márgenes netos de un restaurante medio oscilan entre el 3% y el 9% (National Restaurant Association, 2023), lo que hace que cualquier ineficiencia en la gestión del aprovisionamiento y el inventario tenga un impacto directo sobre la viabilidad del negocio. Para un restaurante independiente, que no tiene ni un departamento de compras ni un sistema informático integrado, este problema es especialmente grave.

La digitalización del sector ha avanzado de forma desigual. Herramientas como los TPV, las plataformas de reservas online o los servicios de delivery se han generalizado incluso en los locales más pequeños. Sin embargo, todo lo que ocurre en la cocina y el almacén (lo que se llama el back-office) sigue funcionando en gran medida de forma manual: pedidos por teléfono o WhatsApp, control de stock con una hoja de cálculo y previsiones basadas en la

experiencia del cocinero. Esta brecha entre lo que ocurre en la sala y lo que ocurre en la cocina es exactamente el hueco que Zipo quiere cubrir.

1.2 Problemática identificada

El cliente al que va dirigido Zipo es el propietario o gerente de un restaurante independiente o una pequeña cadena de hasta 5 locales. Es alguien que lleva el negocio entero, que pide la mercancía, supervisa la cocina y cuadra la caja, a menudo sin un equipo de apoyo. Los problemas que enfrenta a diario en la gestión del inventario son recurrentes y tienen consecuencias económicas directas.

Los tres problemas que más pesan en su día a día son los mismos en casi todos los casos: no saber cuánto stock hay en cada momento (porque los recuentos se hacen a mano y siempre van con retraso), no acertar con los pedidos al proveedor (se pide de más y caduca, o se pide de menos y se queda sin servir un plato), y el desperdicio alimentario que viene de estos dos problemas y que además ahora exige cumplir la Ley 1/2025. Los capítulos 2 y 3 desarrollan estos tres puntos.

1.3 Presentación de Zipo

Zipo es una plataforma software as a service (SaaS) diseñada para que un restaurante independiente pueda gestionar su inventario, predecir cuánto va a necesitar, automatizar sus pedidos y controlar las fechas de caducidad, todo desde un único sitio y sin necesidad de conocimientos técnicos. Incorpora además un sistema de donación de excedentes alimentarios a entidades sociales como elemento diferencial frente a los competidores.

El objetivo central de este TFG es diseñar y construir la parte analítica y tecnológica que hace Zipo posible. Para ello, el trabajo define cómo está estructurada la plataforma por dentro, qué hace cada parte y cómo se conectan entre ellas, y desarrolla de forma completa el módulo de predicción de demanda, incluyendo los cálculos y los resultados obtenidos sobre datos representativos del sector.

1.4 Objetivos del trabajo

- Contextualizar Zipo: definir a qué cliente va dirigida, qué problema resuelve y bajo qué modelo de negocio opera, como marco necesario para el diseño técnico.
- Diseñar la arquitectura funcional y tecnológica de la plataforma: qué módulos la componen, qué cálculos realiza cada uno y cómo fluye la información entre ellos.
- Desarrollar de forma completa el módulo de predicción de demanda: modelo utilizado, fórmulas, parámetros y resultados sobre datos representativos del sector.
- Evaluar el impacto potencial de la plataforma en términos de eficiencia operativa, reducción del desperdicio y mejora del margen del restaurante.

1.5 Metodología

La metodología combina una parte de análisis de negocio con una parte más técnica de análisis de datos. Para entender el contexto del sector me apoyé en fuentes secundarias: el Informe anual de la hostelería española (Hostelería de España, 2023), el informe The State of Food and Agriculture de la FAO (2019) y la estrategia Farm to Fork de la Comisión Europea (2020). La conceptualización del modelo de negocio aplica el Business Model Canvas (Osterwalder & Pigneur, 2010) y el Value Proposition Canvas (Osterwalder, Pigneur, Bernarda & Smith, 2014). El desarrollo técnico utiliza Python con librerías de modelización predictiva sobre datos simulados que reproducen los patrones reales del sector.

Capítulo 2. Modelo de negocio

El modelo de negocio de Zipo se presenta a continuación siguiendo la estructura del Business Model Canvas (Osterwalder & Pigneur, 2010), que organiza los nueve bloques que explican cómo una empresa crea, entrega y captura valor. Se parte de la propuesta de valor y el segmento de clientes, que son el núcleo del modelo, y se desarrollan hacia fuera los canales, la relación con los clientes, las fuentes de ingresos, los recursos y actividades clave, los socios y la estructura de costes. La Figura 2.1 resume el canvas completo de Zipo.

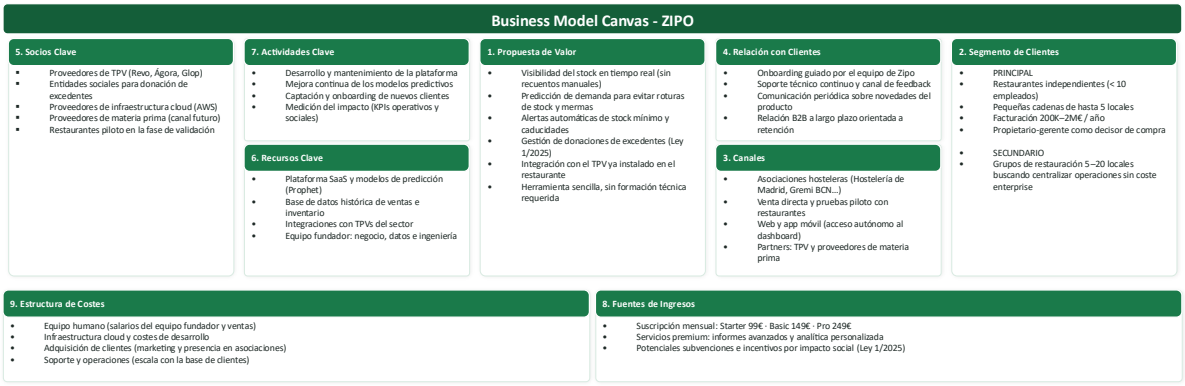

Figura 2.1. Modelo de negocio de Zipo según el Business Model Canvas.
Fuente: Elaboración propia.

2.1 El problema: por qué los restaurantes independientes necesitan Zipo

Más del 95% de los restaurantes en España tienen menos de 10 empleados (INE, 2023). Son negocios que funcionan con márgenes muy ajustados, de entre el 3% y el 9% sobre ventas (National Restaurant Association, 2023), y donde el coste de las materias primas y el desperdicio alimentario representan conjuntamente entre el 28% y el 35% de los ingresos (Comisión Europea, 2020). En ese contexto, una mala gestión del inventario no es un problema menor: es directamente lo que puede hacer inviable el negocio.

El problema no es que los propietarios no quieran mejorar su gestión, sino que no tienen las herramientas adecuadas para hacerlo. La mayoría gestiona el stock con hojas de cálculo, registros en papel o simplemente de memoria. Menos del 30% de los restaurantes independientes en Europa utiliza algún software específico para la gestión de inventarios (McKinsey & Company, 2022), y las pocas soluciones que existen están diseñadas para grandes cadenas: son caras, tardan semanas en implantarse y requieren un equipo dedicado para gestionarlas. El pequeño restaurante queda fuera.

A esto se suma una nueva presión regulatoria. La Ley 1/2025 de prevención del desperdicio alimentario, en vigor desde abril de 2026, obliga a los restaurantes a tener un plan de prevención del desperdicio y a firmar acuerdos con entidades sociales para donar sus excedentes. No cumplir ya no es una opción. Es en este contexto donde surge Zipo.

2.2 Propuesta de valor

Zipo es una plataforma SaaS diseñada para que un restaurante independiente pueda saber en todo momento cuánto stock tiene, cuánto va a necesitar y cuándo tiene que pedir. Todo desde un único sitio, sin formación técnica y conectado con el TPV que el restaurante ya usa.

La propuesta de valor se articula sobre cuatro ejes. El primero es la visibilidad: el responsable del restaurante sabe en todo momento cuánto hay de cada producto en el almacén, sin recuentos manuales. El segundo es la anticipación: el sistema predice cuánto se va a necesitar en los próximos días, teniendo en cuenta el historial de ventas, la estacionalidad y los festivos. El tercero es la automatización: los pedidos, las alertas de caducidad y los informes de consumo se generan solos, sin que nadie tenga que hacerlos. El cuarto es el impacto medible: Zipo muestra en tiempo real el ahorro generado, para que el propietario pueda ver qué le está aportando la herramienta. Además, incorpora un sistema de donación de excedentes que ayuda a cumplir con la Ley 1/2025 sin trabajo administrativo adicional.

2.3 Segmento de clientes

El cliente principal de Zipo es el restaurante independiente o la pequeña cadena de hasta 5 locales, con una facturación anual de entre 200.000 y 2 millones de euros. Es el tipo de negocio que concentra la mayor parte de los problemas descritos y que hoy no encuentra una herramienta adecuada para resolverlos.

Quien toma la decisión de compra es normalmente el propietario-gerente: alguien que lleva el negocio entero, que pide la mercancía, supervisa la cocina y cuadra la caja, a menudo sin equipo de apoyo. Para este perfil, lo que más importa es que la herramienta sea rápida de poner en marcha, que muestre claramente cuánto ahorra y que funcione con el TPV que ya tiene. Cualquier solución que requiera semanas de formación o sustituir el sistema actual es directamente descartada.

Como clientes secundarios con potencial a medio plazo, el modelo contempla grupos de restauración de entre 5 y 20 locales que buscan centralizar y estandarizar su gestión operativa sin asumir el coste de una solución enterprise.

2.4 Canales de distribución y estrategia de entrada

La plataforma llega al cliente a través de tres canales principales. El primero son las asociaciones del sector hostelero (Hostelería de Madrid, Gremi de Restaurants de Barcelona y equivalentes provinciales), que ofrecen acceso directo a miles de potenciales clientes y dan credibilidad a la propuesta. El segundo son las pruebas piloto gratuitas de 30 días, que facilitan que el restaurante pruebe la herramienta y permiten demostrar el valor de Zipo con sus propios datos. El tercero es el marketing de contenidos orientado a gerentes, con recursos prácticos sobre gestión de costes, reducción del desperdicio y optimización del aprovisionamiento.

El foco geográfico inicial es Madrid y Barcelona, donde la concentración de restaurantes independientes con cierto nivel de sofisticación operativa es mayor y el acceso presencial más eficiente. La secuencia de lanzamiento contempla tres fases: desarrollo del producto y validación con pilotos (meses 1 a 6), comercialización activa del plan Starter con el primer equipo de ventas (meses 7 a 18), y activación de los planes Basic y Pro con escala de inversión en marketing y equipo (meses 19 a 36).

2.5 Relación con los clientes

La relación con el cliente es de acompañamiento a largo plazo. En la incorporación, el equipo de Zipo configura la plataforma, migra el catálogo de productos y forma al equipo del restaurante. A partir de ahí, la relación se mantiene vía soporte técnico, comunicación periódica sobre novedades y un canal de feedback directo con el equipo de producto.

2.6 Socios y recursos clave

Los socios clave son los proveedores de TPV (Revo, Ágora, Glop), cuya integración es la fuente principal de datos de ventas; los proveedores de infraestructura cloud, que alojan la plataforma; y las entidades sociales registradas en el sistema de donaciones. A medio plazo, los propios proveedores de alimentos pueden convertirse en socios estratégicos ya que la plataforma les permite recibir pedidos de forma directa y automatizada.

Los recursos clave son la plataforma tecnológica en sí misma, los modelos de predicción de demanda, la base de datos histórica de ventas e inventario que se va construyendo con cada

cliente, y el equipo humano. En la fase inicial, el equipo fundador está compuesto por tres perfiles: un CEO con perfil comercial, un CTO responsable de la arquitectura y el desarrollo, y un CFO para la gestión financiera y administrativa.

2.7 Actividades clave

Las actividades que sostienen el modelo son cuatro: el desarrollo y mantenimiento continuo de la plataforma; el análisis de datos y la mejora de los modelos predictivos a medida que se acumula información real de los restaurantes; la captación y onboarding de nuevos clientes; y la medición del impacto generado, tanto económico (ahorro en compras, reducción de mermas) como social (kilos donados, emisiones evitadas).

2.8 Fuentes de ingresos

Zipo funciona con suscripción mensual y tres niveles de plan adaptados al tamaño del restaurante. El plan Starter (99 €/mes) cubre gestión de inventario, alertas de stock y caducidades y dashboard básico, y está pensado para restaurantes de un solo local. El plan Basic (149 €/mes) añade el módulo de predicción de demanda y las sugerencias automatizadas de pedido, para establecimientos con más volumen o variabilidad. El plan Pro (229 €/mes) incluye gestión multisede, integración directa con proveedores y soporte prioritario, para pequeñas cadenas de hasta 5 locales. Además de la suscripción, se cobra una tarifa única de instalación y onboarding al inicio de la relación.

2.9 Estructura de costes

Los principales bloques de coste son el equipo humano, la infraestructura cloud y la adquisición de clientes. En la fase inicial el peso mayor recae en el desarrollo del producto y los salarios del equipo fundador. A medida que crece la base de clientes, escalan los costes de infraestructura y de ventas. El modelo apunta a márgenes brutos típicos del sector SaaS B2B (Business to Business), con los costes variables de infraestructura representando una fracción pequeña de los ingresos por cliente.

Capítulo 3. Entorno competitivo y oportunidad de mercado

3.1 Oportunidad de mercado

El mercado potencial de Zipo en España parte de 247.604 establecimientos de restauración (CNAE 5610 y 5630, INE DIRCE, 2024), que constituyen el TAM teórico del producto a un precio medio ponderado de 124 €/mes, equivalente a 368 M€/año en ingresos recurrentes. Aplicando filtros de madurez digital (Índice de Madurez Digital del 47,6%, SEGITTUR, 2023) e idoneidad operativa por tipología, el mercado servible real (SAM) se reduce a 55.475 establecimientos, o 82,5 M€/año. Para los tres primeros años, la estimación de captura (SOM) se sitúa en torno a 450 clientes, triangulada por dos métodos independientes: el ritmo de crecimiento de Haddock como benchmark y la cuota del mercado ya servido por competidores verificados, ambos convergiendo en 670.000 €/año de ingresos recurrentes al final del año 3.

3.2 El sector y su nivel de digitalización

El sector de la restauración en España supone en torno al 6,2% del PIB y emplea directamente a más de 1,7 millones de personas (Hostelería de España, 2023). A escala europea representa aproximadamente el 3,9% del valor añadido bruto total de la UE y emplea a cerca de 10 millones de trabajadores (Eurostat, 2022). Es un sector enorme pero con una productividad muy baja, en buena medida porque el back-office (lo que pasa en la cocina y el almacén) sigue sin digitalizarse.

El front-office ha vivido una transformación digital acelerada: los TPVs, las plataformas de delivery (Glovo, Uber Eats, Just Eat), los sistemas de reservas online y los menús QR se han generalizado incluso en los locales más pequeños. Sin embargo, menos del 30% de los restaurantes independientes en Europa utiliza algún software específico para la gestión de inventarios (McKinsey & Company, 2022). Esto genera ineficiencias sistemáticas: pedidos basados en estimaciones, incapacidad de anticipar variaciones de demanda y mayor riesgo de desperdicio.

Tres tendencias hacen que el momento sea favorable para una solución como Zipo: la reducción del coste de las tecnologías cloud, la disponibilidad de librerías de predicción de código abierto (scikit-learn, Prophet) que han bajado la barrera técnica para aplicar modelos

sobre datos reales, y el crecimiento de APIs que facilitan la integración entre sistemas sin sustituir la infraestructura existente.

3.3 El problema del desperdicio alimentario

Según el Ministerio de Agricultura, Pesca y Alimentación (2022), cada restaurante en España desperdicia entre el 4% y el 10% de los alimentos que compra antes de que lleguen al plato del cliente. A nivel global, según la FAO (2019), aproximadamente un tercio de todos los alimentos producidos para consumo humano se pierden o desperdician. Las causas en restauración están directamente relacionadas con la gestión deficiente del inventario: sobrecompra por falta de previsión y ausencia de mecanismos para redistribuir excedentes. El PNUMA (2021) estima que el desperdicio alimentario es responsable de entre el 8% y el 10% de las emisiones globales de gases de efecto invernadero.

El marco normativo español ha endurecido las obligaciones en este campo. La Ley 7/2022, de residuos y suelos contaminados para una economía circular, sentó las bases de la política de prevención de residuos. La norma que afecta directamente a la restauración es la Ley 1/2025, de prevención de las pérdidas y el desperdicio alimentario, aprobada en abril de 2025 y con las obligaciones de plan de prevención y donación en vigor desde abril de 2026. Reducir el desperdicio ha dejado de ser una buena práctica voluntaria para convertirse en una obligación legal. A nivel europeo, la estrategia Farm to Fork fija el objetivo de reducir a la mitad el desperdicio en distribución y consumo para 2030 (Comisión Europea, 2020).

3.4 Análisis competitivo

El mercado de software para la gestión operativa de restaurantes incluye soluciones muy diversas en cuanto a enfoque, alcance funcional y perfil de cliente. A continuación, se analizan los competidores más relevantes, con especial atención a sus limitaciones desde la perspectiva del restaurante independiente al que Zipo se dirige. Se incluye también el análisis del entorno de impacto social, donde operan plataformas con una lógica distinta pero que compiten por el mismo tipo de excedente.

Gstock. Gstock es la solución especializada en gestión de inventarios para hostelería con mayor presencia en el mercado español, con más de 1.800 establecimientos registrados (Asisman, 2024). Ofrece control de compras, escandallos, stock y consumos con integración con TPV y app móvil. Su fortaleza es la profundidad funcional y el conocimiento del mercado español. Sin embargo, el proceso de implantación se estima en torno a diez semanas, lo que es una barrera grande para restaurantes con pocos recursos. Está orientado a grupos de restauración y cadenas, no al restaurante independiente. No incorpora predicción ni componente social.

Choco. Choco es una aplicación gratuita cuyo enfoque es exclusivamente la digitalización del proceso de pedidos a proveedores. Permite centralizar todos los proveedores en una sola interfaz y enviar pedidos en pocos pasos. Su adopción ha sido rápida gracias a la gratuidad para el restaurante, financiando su modelo de negocio a través de tarifas cobradas a los distribuidores. El límite de Choco es precisamente la estrechez de su alcance: es una herramienta de comunicación con proveedores, no de optimización operativa. No ofrece control de inventario, no genera alertas de caducidad, no incorpora previsión de demanda y no produce ninguna analítica sobre consumos o costes.

MarketMan. MarketMan es una plataforma SaaS de gestión de inventario y compras con presencia en más de 55 países y más de 15.000 establecimientos clientes (MarketMan, 2024). Ofrece control de stock, gestión de proveedores, cálculo de coste de recetas e integración con los principales TPVs del mercado anglosajón. Sus dos limitaciones principales para el mercado español son que sus integraciones con TPV (Square, Toast, Lightspeed) tienen escasa penetración en España, y que su capacidad de predicción se basa en análisis básicos de tendencias históricas, sin modelos lo suficientemente potentes como para anticipar cambios complejos de demanda.

Apicbase. Apicbase es una plataforma belga de gestión del back-of-house orientada a grandes operadores, cadenas multisede, catering y ghost kitchens. Ofrece gestión avanzada de recetas, planificación de menús, control de inventario e integración con TPV, y ha incorporado recientemente capacidades de IA para previsiones de ventas. Su propio posicionamiento define su limitación: la plataforma “requiere un equipo de restauración con conocimientos y

compromiso para gestionarla correctamente" (Capterra, 2024), lo que la hace inadecuada para operadores pequeños. Su pricing se negocia caso a caso según el restaurante.

Too Good To Go y el entorno de impacto social. Too Good To Go es la plataforma líder en Europa para la redistribución de excedentes alimentarios de restaurantes y comercios. Permite a los establecimientos vender a precio reducido la comida que no han podido vender durante el día, conectándolos con consumidores finales a través de su app. Su modelo genera valor al restaurante al recuperar parte del coste de la merma, y al consumidor al ofrecerle comida a buen precio. En 2023 operaba en más de 17 países y había salvado más de 200 millones de comidas del desperdicio (Too Good To Go, 2023).

Too Good To Go no es un competidor directo de Zipo en el sentido operativo: no gestiona inventario, no predice demanda y no automatiza pedidos. Sin embargo, opera sobre el mismo excedente que Zipo detecta y gestiona. La diferencia está en el destino: Too Good To Go redistribuye el excedente a consumidores a precio reducido, mientras que Zipo lo dona a entidades sociales sin ánimo de lucro, lo que permite al restaurante cumplir con la Ley 1/2025 y acceder a posibles beneficios fiscales. Ambas plataformas son más complementarias que sustitutas: un restaurante puede usar Zipo para gestionar su inventario y prevenir el desperdicio, y Too Good To Go para dar salida a los excedentes que de todas formas no van a consumirse.

El conjunto de soluciones analizadas confirma la existencia de un hueco de mercado claro: no existe actualmente en el mercado español una plataforma que combine control de inventario en tiempo real, previsión de demanda basada en análisis de datos, accesibilidad para restaurantes independientes y un sistema nativo de gestión de excedentes. Cada competidor cubre una parte del problema, pero ninguno lo aborda de forma integral desde una perspectiva analítica y accesible.

Capítulo 4. Diseño funcional de la plataforma

Zipo se organiza en seis módulos funcionales que trabajan juntos para cubrir el ciclo completo de gestión de un restaurante: saber cuánto hay en el almacén, predecir cuánto se va a

necesitar, hacer el pedido al proveedor, controlar las fechas de caducidad, ver el estado del negocio de un vistazo y gestionar las donaciones de excedentes. Cada módulo tiene su propia función, pero comparte datos con los demás, de forma que la información fluye de uno a otro sin que el responsable del restaurante tenga que hacer nada manual.

Este capítulo describe qué hace cada módulo, qué información necesita, qué produce y cómo se conecta con el siguiente. Al final del capítulo se explica cómo fluye la información entre todos ellos.

4.1 Visión general de la plataforma

La plataforma funciona principalmente a partir de un dato: cada vez que se vende un plato en el restaurante, el TPV (la caja registradora) le avisa a Zipo. A partir de ese aviso, Zipo descuenta los ingredientes del inventario, añade la venta al historial del modelo de predicción y actualiza el dashboard. Todo eso ocurre de forma automática, sin que nadie tenga que hacer nada.

El responsable del restaurante interactúa con la plataforma principalmente de dos formas: desde la app del móvil para registrar lo que llega del proveedor o apuntar una merma, y desde el panel web cuando quiere revisar los pedidos sugeridos, consultar los informes o aprobar una donación. El resto del tiempo, la plataforma trabaja sola.

4.2 Cómo fluye la información entre los módulos

Los seis módulos están conectados entre sí de forma que cada acción que ocurre en uno de ellos desencadena algo en otro. El flujo principal funciona así:

Cuando alguien pide un plato en el restaurante, el TPV avisa a Zipo. Ese aviso hace dos cosas al mismo tiempo: descuenta los ingredientes del inventario (M1) y añade la venta al historial que usa el modelo de predicción (M2). Cada día por la mañana, el modelo de predicción procesa ese historial actualizado y calcula cuánto se va a necesitar en los próximos 7 días. Con esa información, el módulo de pedidos (M3) compara lo que se necesita con lo que hay y genera el borrador de pedido para que el responsable lo apruebe.

Mientras todo eso ocurre, el módulo de caducidades (M4) está mirando constantemente si hay algún producto que se acerque a su fecha límite. Cuando salta una alerta, el responsable decide qué hacer: si opta por donar, el módulo de donaciones (M6) gestiona todo el proceso y actualiza el inventario. El dashboard (M5) recoge en tiempo real lo que está pasando en todos los módulos y lo muestra de forma resumida.

El resultado es que la misma venta que reduce el stock también mejora la previsión del próximo pedido, reduce el riesgo de caducidades y alimenta el informe de sostenibilidad. Todo conectado, sin pasos manuales entre medias.

4.3 Módulo 1: gestión de inventario

El módulo de inventario es la base de todo lo demás: lleva la cuenta de cuánto hay de cada producto en el almacén en todo momento. Lo hace de tres formas: descontando automáticamente lo que se vende (gracias a la conexión con el TPV), sumando lo que entra cuando llega un pedido del proveedor, y apuntando las mermas cuando el equipo de cocina registra que algo se ha roto o ha caducado. Desde la app móvil se puede escanear el código de barras de un producto para registrar su llegada, ver el stock en tiempo real y recibir una alerta cuando algún producto baja del mínimo configurado.

Aspecto	Detalle
Inputs necesarios	Catálogo de productos (nombre, unidad, proveedor asignado, stock mínimo). Datos de ventas del TPV para descuento automático de stock. Recepciones de mercancía (entrada manual o escaneo de albarán). Mermas registradas por el equipo.
Outputs generados	Stock actual por producto en tiempo real. Historial de movimientos (entradas, salidas, mermas). Alertas de stock mínimo. Informe de desviaciones entre stock teórico y real.

4.4 Módulo 2: predicción de demanda

El módulo de predicción es el componente más analítico de Zipo: anticipa cuántas unidades de cada producto se van a necesitar en los próximos 7 días, para que el restaurante pueda pedir lo que necesita sin quedarse corto ni acumular género que después pueda caducar. Para

hacerlo aprende del historial: tiene en cuenta el día de la semana (si los viernes se vende mucho más que los lunes), la temporada del año (verano frente a invierno) y los festivos o eventos cercanos. El resultado es una cifra por producto y día, junto con un margen de error que indica el rango dentro del cual es probable que se mueva la demanda real.

Aspecto	Detalle
Inputs necesarios	Historial de ventas por producto (mínimo 8-12 semanas para calibración inicial). Calendario de festivos nacionales y locales. Datos de eventos externos opcionales (API de agenda cultural o deportiva). Mermas históricas para ajustar consumo real frente a consumo vendido.
Outputs generados	Previsión de demanda por producto y día para el horizonte configurado. Intervalo de confianza (cantidad mínima y máxima esperada). Métrica de precisión MAPE visible en el dashboard. Recomendación de stock objetivo para el periodo previsto.

4.5 Módulo 3: automatización de pedidos

Una vez que el módulo de predicción sabe cuánto se va a consumir en los próximos días, el módulo de pedidos lo compara con el stock actual y calcula cuánto hay que pedir: lo que se necesita, menos lo que ya hay, más un margen de seguridad para no quedarse justo. El resultado se agrupa por proveedor y se convierte en un borrador de pedido que el responsable revisa, ajusta a mano si quiere y aprueba con un clic; en ese momento el pedido sale automáticamente al proveedor. El sistema también detecta si el precio que ha mandado el proveedor ha cambiado respecto al último pedido y lanza una alerta antes de que el responsable lo apruebe, para que siempre sea consciente de lo que va a pagar. Cuando llega el pedido, el responsable puede registrar incidencias en la recepción: si un producto viene en mal estado o no cumple la calidad esperada, la incidencia queda anotada en el almacén; si el problema es recurrente con un proveedor (retrasos, errores de cantidad), se acumula en su ficha para tenerlo en cuenta en futuros pedidos.

Aspecto	Detalle
Inputs necesarios	Stock actual de cada producto. Previsión de demanda para el horizonte de aprovisionamiento. Stock mínimo y margen de seguridad configurados. Proveedor asignado y condiciones de pedido (cantidad mínima, plazo de entrega). Historial de precios y tiempos y calidad de servicio de proveedor para detectar variaciones.

Outputs generados	Borradores de pedido por proveedor listos para aprobar. Pedidos confirmados enviados con acuse de recibo. Historial de pedidos con estado (enviado, confirmado, recibido, con incidencias). Alertas de variación de precio.

4.6 Módulo 4: gestión de caducidades

El módulo de caducidades resuelve uno de los problemas más cotidianos de cualquier cocina: saber qué hay en el almacén, cuándo caduca cada cosa y qué hay que usar primero. Para hacerlo, necesita conocer la fecha de caducidad de cada producto en el momento en que llega al restaurante. El reto es que esa información no siempre viene en el mismo formato: hay productos con código QR o de barras que incluyen todos los datos, otros que solo tienen una fecha impresa en el envase, y otros, como las frutas frescas o los productos a granel, que no llevan ninguna indicación.

Para cubrir los tres casos, el módulo incorpora tres vías de registro, que se activan automáticamente según el producto. Una cuarta vía permite ajustar o corregir la información a mano cuando haga falta.

Cómo se registra la caducidad al recibir un producto. Vía 1, código QR o EAN: si el producto llega con un código QR o de barras que incluye los datos del lote y la caducidad, el empleado simplemente lo escanea con la cámara del móvil. La app lee el código, identifica el producto en el catálogo de Zipo y registra automáticamente la fecha de caducidad, la cantidad y el peso sin ningún paso adicional. Este es el caso de la mayoría de los productos envasados que llegan de proveedor.

Vía 2, reconocimiento de texto (OCR): cuando el producto tiene una fecha impresa en el envase pero no tiene código escaneable, el empleado hace una foto con el móvil. La app envía la imagen a un motor de reconocimiento de texto (tecnología OCR, como la que utiliza Dijit.app para el sector de la hostelería), que lee la fecha del envase y la introduce automáticamente en el registro. También puede leer el albarán del proveedor cuando éste incluye las fechas de caducidad de los lotes entregados, lo que permite registrar varios productos a la vez con una sola foto.

Vía 3, vida útil estimada por base de datos propia: cuando el producto no tiene ni código ni fecha visible (frutas, verduras frescas, productos a granel), la app identifica el tipo de alimento por la foto o el nombre y consulta una base de datos interna de vidas útiles medias por categoría. Por ejemplo, si se registran zanahorias frescas, el sistema asigna automáticamente una vida útil de 7 días desde la fecha de recepción. Estos valores por defecto son configurables por el restaurante si su experiencia indica algo diferente.

Vía 4, dictado por voz: para situaciones en las que llega un pedido voluminoso y el responsable quiere registrarlo rápidamente sin detenerse a escanear cada producto, la aplicación ofrece la opción de grabar un audio. El responsable habla directamente ("han llegado 3 kg de lomo ibérico con caducidad a 5 días, 2 kg de solomillo a una semana y 10 litros de leche entera a 12 días") y la aplicación transcribe y estructura la información automáticamente mediante tecnologías de reconocimiento de voz similares a OpenAI Whisper, capaces de convertir el audio en texto e identificar datos clave como el producto, la cantidad y la fecha de caducidad. Posteriormente, el responsable revisa y confirma la información antes de guardarla en el sistema.

Independientemente de la vía utilizada, la pantalla de confirmación siempre permite al responsable editar cualquier campo antes de guardar: nombre del producto, cantidad, peso, fecha de caducidad y estado del producto (fresco, en buen estado, con golpes, etc.). Esto garantiza que incluso cuando el reconocimiento automático comete un error, el dato que queda guardado es correcto.

Vigilancia continua y alertas. Una vez registrado un producto, el módulo lo vigila continuamente. Varias veces al día, un proceso automático calcula para cada lote en stock cuántas horas quedan hasta su caducidad y compara ese valor con los umbrales configurados por el restaurante. Cuando un producto cruza el primer umbral (por defecto, 72 horas), se genera una alerta amarilla; cuando cruza el segundo (por defecto, 24 horas), la alerta pasa a roja. Si el responsable no actúa, la alerta escala automáticamente para asegurarse de que no pasa desapercibida.

Al recibir una alerta, el responsable tiene tres opciones. La primera es darle prioridad al producto en cocina, marcando en el sistema que ese lote debe usarse antes que otros del

mismo tipo. La segunda es activar una promoción rápida para acelerar su consumo en sala. La tercera es convertirlo en una donación: con un solo toque, el módulo de caducidades activa el flujo de donaciones (módulo 6), que gestiona automáticamente la comunicación con las entidades sociales y genera el justificante. Todas las acciones quedan registradas para el informe de sostenibilidad.

4.7 Módulo 5: dashboard y KPIs

El dashboard es la pantalla principal de Zipo: el sitio donde el responsable ve, de un vistazo, cómo va el negocio y a qué alertas debe atender hoy. Además, muestra cómo ha evolucionado el coste de la materia prima, cuánto se ha desperdiciado este mes comparado con el anterior y qué productos tienen más rotación. Las alertas urgentes llegan también como notificación al móvil, sin que el responsable tenga que abrir la aplicación.

Aspecto	Detalle
Inputs necesarios	Datos agregados de todos los módulos: stock actual, alertas activas, pedidos pendientes, previsiones de demanda, historial de ventas del TPV, registro de mermas y donaciones.
KPIs principales	Coste de materia prima (real y porcentaje sobre ventas). Tasa de desperdicio (mermas sobre compras totales). Rotación de inventario por categoría. Precisión del modelo de predicción (MAPE). Número de alertas activas de stock y caducidad. Ahorro acumulado estimado por Zipo.

4.8 Módulo 6: sistema de donaciones

Cuando el módulo de caducidades detecta que hay género que no va a poder consumirse a tiempo, se activa el flujo de donaciones. El sistema busca automáticamente qué entidades sociales registradas están cerca y les manda una notificación con los detalles del excedente disponible; la primera que confirma la recogida se queda con la donación. Una vez confirmada, el sistema descuenta los productos del inventario, genera el justificante en PDF (necesario para cumplir con la Ley 1/2025 y para las ventajas fiscales) y añade los datos al informe de sostenibilidad: kilos donados, valor económico del excedente y estimación de CO_2 evitado. Todo sin que el responsable tenga que rellenar ningún formulario.

Aspecto	Detalle

Inputs necesarios	Producto disponible para donar (tipo, cantidad, fecha de caducidad). Entidades sociales registradas con área geográfica y disponibilidad horaria. Preferencias del restaurante sobre qué entidades reciben sus donaciones.
Outputs generados	Notificación a entidades elegibles. Justificante de donación (PDF) para uso legal y fiscal. Actualización del inventario. Acumulación en el informe de sostenibilidad (kilos donados, CO2 evitado estimado, valor económico).

Capítulo 5. Arquitectura del sistema

Este capítulo describe la arquitectura tecnológica de Zipo, detallando los principales componentes que conforman el sistema, las interacciones entre ellos y el flujo de información desde el momento en que el TPV registra una venta hasta que dichos datos son procesados y visualizados por el responsable a través del dashboard.

5.1 Visión general

La arquitectura de Zipo se puede conceptualizar como una cadena de cuatro capas apiladas. En la capa de arriba están los datos que entran desde fuera del sistema: las ventas del TPV, los avisos de los proveedores, el calendario de festivos y el portal de las entidades sociales. Esos datos pasan por una capa de entrada, un filtro, que comprueba quién envía cada cosa y la dirige al módulo correcto. Debajo está el núcleo de la aplicación, con los seis módulos funcionales y el motor de predicción. Y en la parte de abajo están las bases de datos donde se guarda toda la información. La app y la web del restaurante están conectadas al núcleo y muestran al usuario lo que necesita ver (véase Figura 5.1).

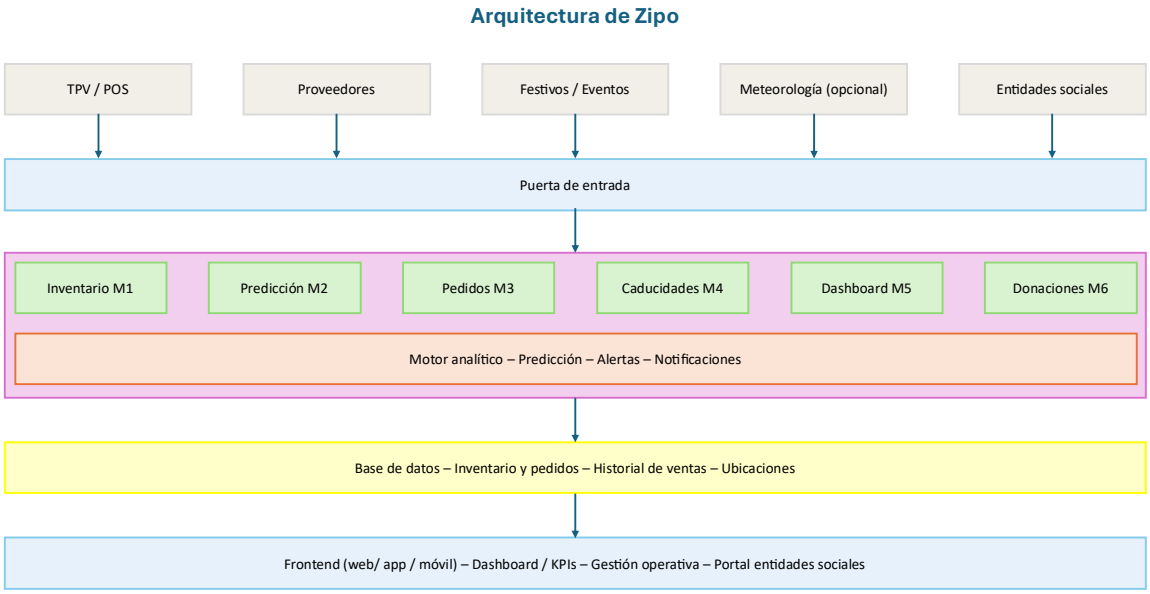

Figura 5.1. Arquitectura general de la plataforma Zipo.
Fuente: Elaboración propia.

5.2 Lo que ve el usuario: app y web

El responsable del restaurante accede a Zipo de dos formas. Desde el ordenador, a través de la web, puede ver los informes, configurar los parámetros y aprobar los pedidos. Desde el móvil, con la app, puede hacer el trabajo del día a día: escanear productos que llegan, registrar mermas y consultar las alertas. Ambas formas de acceder muestran la misma información y comparten los mismos datos. Las gráficas del dashboard se generan con librerías estándar (Chart.js o Recharts) que se pueden renderizar tanto en el navegador como en el móvil.

5.3 La puerta de entrada: conexión con el TPV y los proveedores

Toda la información que llega de fuera (ventas, pedidos, festivos) pasa primero por un punto de control centralizado. Este punto comprueba que quien manda los datos tiene permiso para hacerlo, y los dirige al módulo que corresponde. La conexión con el TPV es la más crítica porque es la fuente principal de datos de ventas. Dependiendo del sistema que use cada restaurante, la conexión funciona mediante avisos automáticos (el TPV notifica a Zipo cada vez que se registra una venta) o mediante consultas periódicas (Zipo le pregunta al TPV qué ha vendido en las últimas horas). Zipo ya tiene conexiones preparadas para los TPVs más

usados en España (Revo, Ágora, Glop) y ofrece una vía genérica de importación para los que no estén cubiertos.

5.4 El núcleo: los seis módulos

El centro de la aplicación es donde vive la lógica de los seis módulos descritos en el capítulo 4. Cada módulo recibe información, y produce un resultado que puede ser un dato guardado en la base de datos, una alerta al usuario o una acción automática como enviar un pedido. Los módulos se comunican entre sí a través de mensajes internos mediante un sistema de colas de mensajes (RabbitMQ): cuando algo cambia en uno, publica un evento en la cola y los módulos suscritos reaccionan automáticamente. Por ejemplo, cuando el módulo de inventario actualiza el stock, el módulo de predicción sabe que tiene datos nuevos; y cuando el módulo de caducidades detecta una alerta grave, puede arrancar automáticamente el flujo de donaciones. Todos estos módulos se ejecutan en servidores en la nube (AWS), no en el local del restaurante. El restaurante solo necesita su TPV y un dispositivo con conexión a internet para acceder a la app o la web; toda la lógica y el procesamiento ocurren en el servidor.

5.5 El motor analítico: predicción y alertas

El motor analítico es la pieza que diferencia a Zipo de una simple aplicación de gestión. Funciona de forma independiente al resto y se compone de dos partes: el motor de predicción, que genera las previsiones de demanda, y el motor de alertas, que revisa continuamente el inventario y las caducidades y envía notificaciones cuando detecta algo que requiere atención. El detalle de cómo funciona cada uno se explica en la sección 5.7.

5.6 Dónde se guarda la información

Zipo usa tres tipos de almacenamiento, cada uno pensado para un tipo de información distinto:

La base de datos principal (PostgreSQL) guarda todo lo que tiene que ser siempre fiable: el catálogo de productos, los movimientos de stock, el historial de pedidos, los usuarios y sus permisos, los proveedores y las entidades sociales. Es la base de datos que garantiza que ningún dato se pierde ni se contradice.

La base de datos de series temporales (TimescaleDB, una extensión de PostgreSQL) almacena el historial de ventas por producto y día. Está optimizada para guardar y consultar grandes cantidades de datos asociados a fechas, que es exactamente lo que necesita el modelo de predicción.

La base de datos geoespacial (PostGIS, también una extensión de PostgreSQL) guarda la ubicación de las entidades sociales y permite encontrar rápidamente las más cercanas cuando hay una donación disponible. Al ser una extensión de PostgreSQL, no es necesario montar ningún sistema adicional. Las tres bases de datos viven en la infraestructura cloud de Zipo (AWS), no en el local del cliente. El restaurante no tiene que instalar ni mantener ningún servidor: accede siempre a través de la app o la web.

5.7 Cómo funciona cada módulo por dentro

Esta sección describe los cálculos que realiza cada módulo, las fórmulas que aplica y qué produce para el siguiente.

5.7.1 Módulo 1: gestión de inventario

El cálculo central de este módulo es la actualización del stock en tiempo real. Cada vez que ocurre un movimiento (una venta registrada por el TPV, una recepción de mercancía o una merma anotada por el equipo), el sistema aplica la operación:

$$Stock_t = Stock_{t-1} + Entradas - Salidas - Mermas$$

Las salidas por venta se calculan a partir del escandallo: cuando el TPV registra la venta de un plato, el sistema consulta la receta y descuenta de cada ingrediente la cantidad correspondiente (por ejemplo, una hamburguesa descuenta 150 g de carne, 1 pan, 5 g de cebolla y 20 g de queso) incrementada por la merma presupuestada. Si al limpiar y pelar la cebolla se pierde un 30% (recogido en el escandallo), la receta genera un consumo total de $^{5}/_{(1-0.3)} = 7.14$g de cebolla.

Además del stock actual, el módulo compara periódicamente cada producto con el stock mínimo configurado y, si Stock actual < Stock mínimo, genera una alerta al dashboard y al

móvil del responsable. El stock mínimo de cada producto lo propone el sistema por defecto a partir del consumo medio diario y el plazo de entrega del proveedor, y el responsable puede ajustarlo si lo necesita. El cálculo detallado del margen de seguridad se explica en el capítulo 6. El stock actualizado se publica en un canal interno que el módulo de predicción (M2) y el de pedidos (M3) consultan para sus cálculos. Véase Figura 5.2.

Figura 5.2. Flujo de datos del módulo de gestión de inventario (M1).

Fuente: Elaboración propia.

5.7.2 Módulo 2: predicción de demanda

Este módulo ejecuta cada día, de madrugada, el modelo de predicción de demanda. El modelo descompone el historial de ventas de cada producto en tres componentes que suma para obtener la previsión:

$$demanda_t = tendencia_t + \text{efecto día de la semana}_t + \text{efecto temporada}_t + error_t$$

La tendencia captura si las ventas están subiendo o bajando en general; el efecto del día de la semana recoge que los viernes y sábados se vende más que los lunes; el efecto de temporada refleja diferencias entre verano, navidades u otras épocas; y el error es lo que queda sin

explicar, y se usa para calcular el intervalo de confianza. El capítulo 6 desarrolla en detalle el modelo, sus parámetros y los resultados sobre datos del sector.

El resultado de cada ejecución es una tabla con la previsión de demanda por producto y día para los próximos 7 días, junto con los intervalos de confianza. Esa tabla se guarda en una memoria intermedia de acceso rápido que el módulo de pedidos (M3) y el dashboard (M5) leen cuando la necesitan, sin volver a ejecutar el modelo. Véase Figura 5.3.

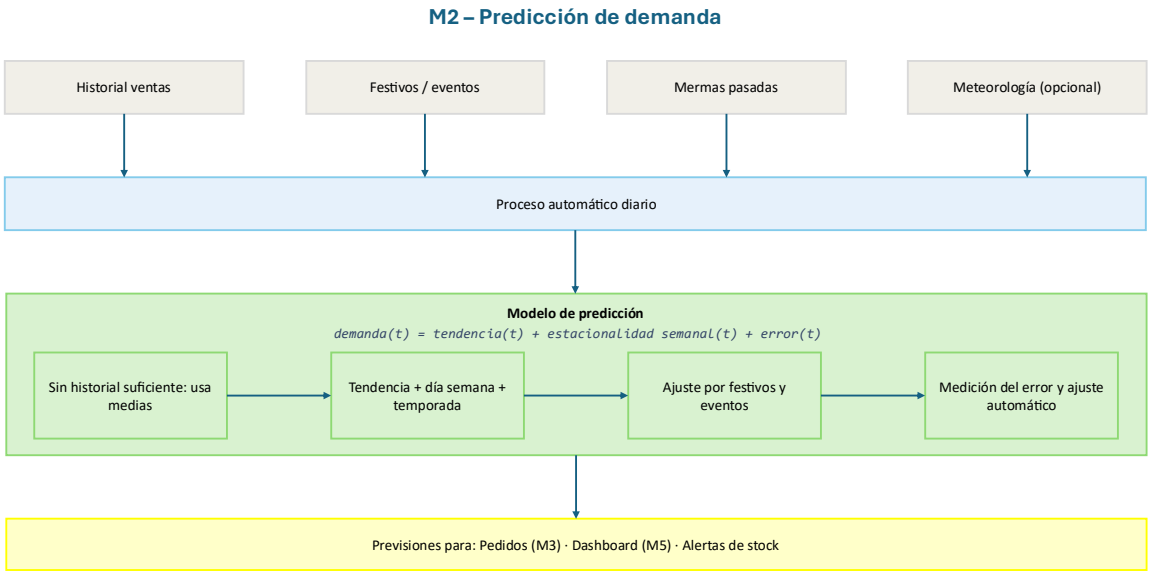

Figura 5.3. Flujo de datos del módulo de predicción de demanda (M2).

Fuente: Elaboración propia.

5.7.3 Módulo 3: automatización de pedidos

Con la previsión de M2 y el stock actual de M1, este módulo calcula cuánto hay que pedir de cada producto en dos pasos. Primero calcula el stock objetivo, que es la demanda prevista más un stock de seguridad que protege contra desviaciones al alza de la demanda. El stock de seguridad se calcula como:

$$Stock\ de\ seguridad\ =\ Z\ \times\ \sigma_{demanda} \times \sqrt{Plazo de\ entrega}$$

donde Z es el nivel de servicio (el porcentaje de la demanda que se quiere tener cubierto, según una distribución normal), σ_demanda es la volatilidad de la demanda del producto y el

plazo de entrega es el tiempo que tarda el proveedor en servir el pedido. Cuanto más variable es la demanda de un producto o más tarda el proveedor en entregar, mayor es el stock de seguridad necesario. Cuando todavía no se conoce la volatilidad de un producto (por ejemplo, al empezar a usar Zipo), el sistema aplica por defecto un 30% sobre la demanda prevista, una regla habitual en gestión de inventarios. Luego calcula la cantidad a pedir:

$$Cantidad\ a\ pedir\ =\ máx(0, Stock\ objetivo\ -\ Stock\ actual\ -\ Pedidos\ en\ tránsito)$$

El máx(0, …) garantiza que nunca se genera un pedido negativo si ya hay stock suficiente. El plazo de entrega del proveedor se tiene en cuenta también para decidir el momento del pedido: el sistema avisa con la antelación suficiente para que el pedido llegue antes de quedarse sin stock.

Los resultados se agrupan por proveedor, respetando las cantidades mínimas pactadas, y se genera un borrador que el responsable revisa y aprueba con un clic. Si el precio del proveedor ha cambiado respecto al último pedido, el sistema lanza una alerta antes de confirmar. Véase Figura 5.4.

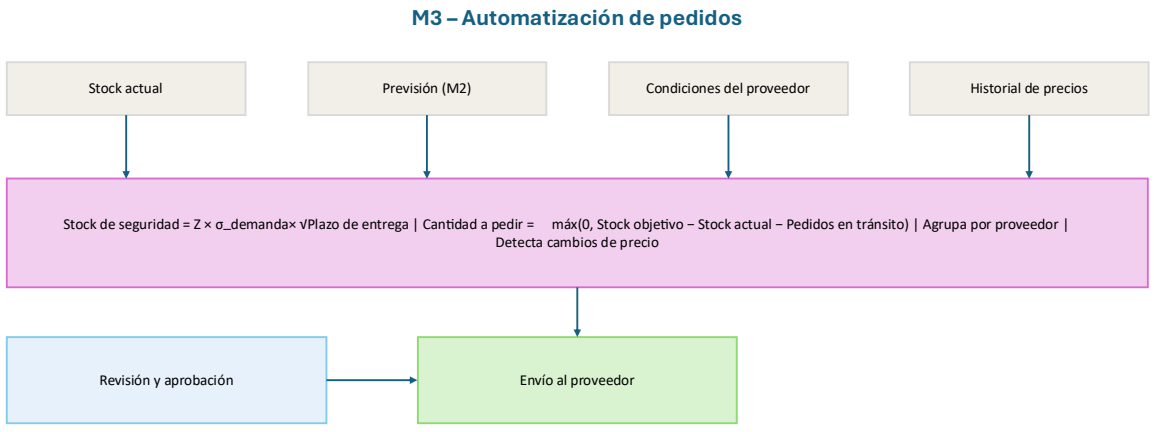

Figura 5.4. Flujo de datos del módulo de automatización de pedidos (M3).

Fuente: Elaboración propia.

5.7.4 Módulo 4: gestión de caducidades

El cálculo clave de este módulo es el tiempo restante hasta la caducidad de cada lote en stock. Varias veces al día el sistema ejecuta:

$$Horas\ restantes\ =\ Fecha\ de\ caducidad\ -\ Momento\ actual$$

Si Horas restantes < umbral amarillo (por defecto 72 h) → alerta amarilla. Si Horas restantes < umbral rojo (por defecto 24 h) → alerta roja. Ambos umbrales son configurables por el restaurante y por categoría de producto. La alerta activa el flujo de acción del responsable y, si éste opta por donar, pasa el control al módulo de donaciones (M6). Todo queda registrado para el informe de sostenibilidad. Véase Figura 5.5.

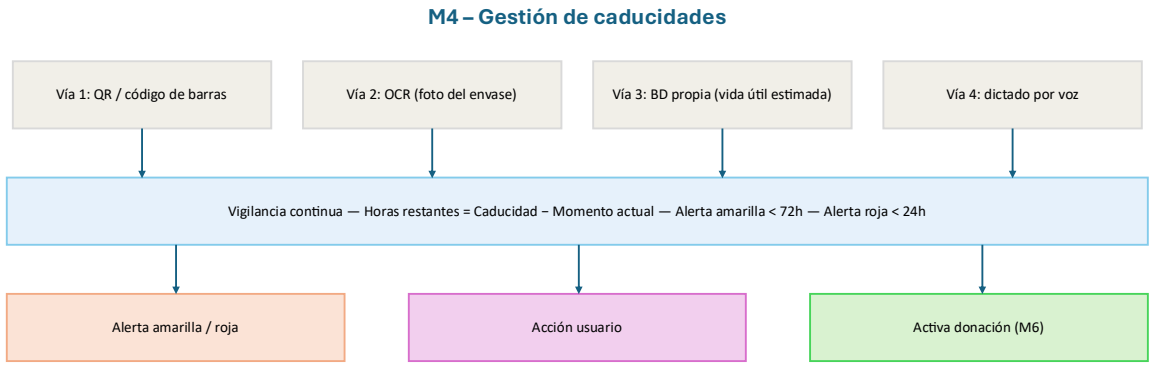

Figura 5.5. Flujo de datos del módulo de gestión de caducidades (M4).

Fuente: Elaboración propia.

5.7.5 Módulo 5: dashboard y KPIs

El dashboard no genera datos propios: agrega y presenta los de todos los demás módulos. Los cálculos principales que realiza son cuatro KPIs operativos:

$$Coste\ de\ ventas\ (\%) = Coste\ real\ de\ las\ ventas\ (según\ escandallo)\ /\ Ingresos\ por\ ventas \times 100$$

$$Tasa\ de\ desperdicio\ (\%) = Valor\ económico\ de\ mermas\ y\ caducidades\ /\ Valor\ total\ de\ compras\ \times 100$$

$$Rotación\ de\ inventario\ (unidades) = Unidades\ consumidas\ en\ el\ período\ /\ Stock\ medio\ del\ período$$

$$Precisión\ del\ modelo\ (MAPE) = Media\ del\ absoluto\ entre\ \left(Demanda\ real\ -\ \frac{Previsión}{Demanda\ real} \times 100\right)$$

Las alertas activas (stock mínimo, caducidades) se muestran en tiempo real, mientras que las métricas históricas se recalculan una vez al día. Las alertas críticas se envían también como notificación al móvil sin que el responsable tenga que abrir la app. Véase Figura 5.6.

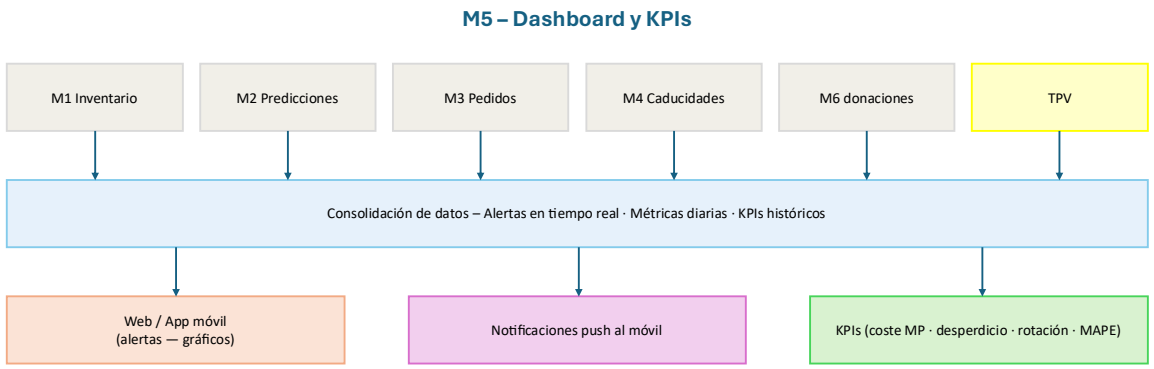

Figura 5.6. Flujo de datos del módulo de dashboard y KPIs (M5).

Fuente: Elaboración propia.

5.7.6 Módulo 6: sistema de donaciones

Cuando el responsable activa una donación desde M4, este módulo calcula qué entidades sociales son elegibles. La selección usa dos criterios: proximidad geográfica (distancia en línea recta entre el restaurante y la entidad, usando sus coordenadas almacenadas) y disponibilidad horaria (si la entidad puede recoger en el margen de tiempo que queda antes de la caducidad). La primera entidad que confirma bloquea la donación. A partir de ahí el módulo descuenta el producto del inventario (M1), genera el justificante en PDF y calcula las métricas de sostenibilidad:

$$CO^2 evitado\ (kg) = Kilos\ donados \times Factor\ de\ emisión\ por\ kg\ de\ alimento\ (según\ categoría).$$

$$Valor\ económico\ del\ excedente = Kilos\ donados \times Precio\ de\ coste\ por\ kg.$$

Estos datos se acumulan en el informe de sostenibilidad del restaurante. Véase Figura 5.7.

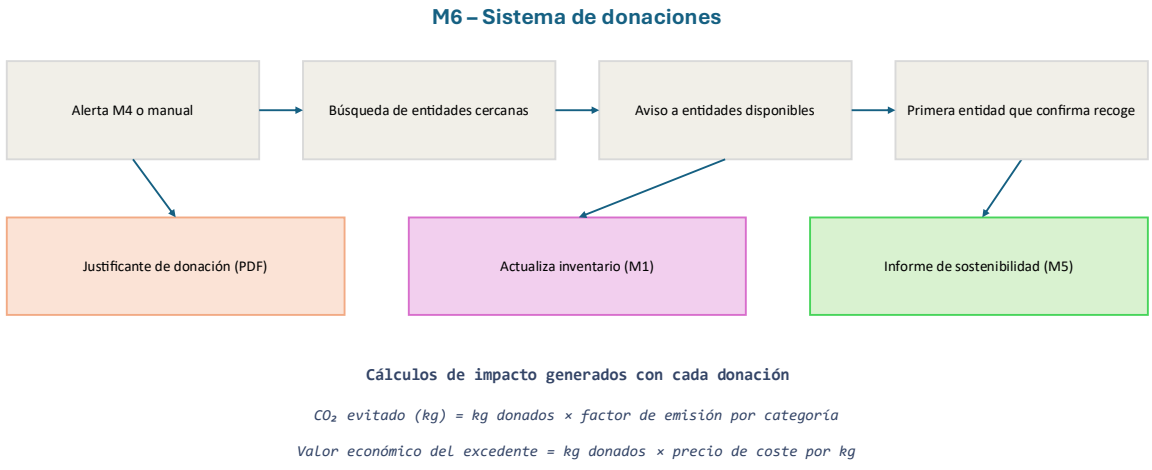

Figura 5.7. Flujo de datos del módulo de sistema de donaciones (M6).

Fuente: Elaboración propia.

5.8 Principios de diseño

Modularidad. La arquitectura permite que cada módulo funcione de forma independiente. Esto tiene tres ventajas prácticas: permite lanzar la plataforma con solo parte de los módulos e ir añadiendo los demás de forma incremental; si algo falla en un módulo, los demás siguen funcionando; y si un módulo necesita más potencia de cálculo, se puede reforzar sin tocar el resto.

Escalabilidad. Los módulos operativos pueden crecer simplemente añadiendo más servidores que repartan el trabajo a medida que entran más clientes. El motor analítico crece ejecutando sus tareas con más frecuencia o repartiéndolas en varios procesos a la vez. La base de datos puede apoyarse en copias adicionales para repartir las consultas del dashboard, que son las más habituales.

Seguridad. El diseño de seguridad parte de una premisa básica: no todos los datos de Zipo tienen el mismo nivel de sensibilidad, y por eso no todos están localizados en el mismo sitio ni se gestionan de la misma manera. Hay que distinguir entre dos tipos de información.

Por un lado, están los datos operativos: movimientos de stock, historial de ventas, previsiones, alertas de caducidad, borradores de pedido. Esta información es la que procesa el motor analítico y la que se comparte con servicios externos como la infraestructura cloud para ejecutar los modelos de predicción. No contiene información personal identificable y puede circular entre servidores sin riesgo para la privacidad del restaurante o sus clientes.

Por otro lado, están los datos sensibles: información de los usuarios (nombre, credenciales de acceso, rol), datos del restaurante (dirección, CIF, datos bancarios para la facturación), datos de contacto de proveedores y entidades sociales, y los justificantes de donación. Esta información reside exclusivamente en los servidores propios de Zipo, nunca se comparte con servicios externos en texto claro, y viaja siempre cifrada. El acceso está restringido por rol: un empleado de cocina solo puede ver y editar el inventario de su local, mientras que el propietario tiene acceso a todos los informes y configuraciones. Cualquier acción relevante (un cambio de stock, una confirmación de pedido, un registro de donación) queda registrada con su fecha, hora y usuario, de forma que siempre se puede saber quién hizo qué y cuándo.

5.9 Por qué es necesaria la predicción: impacto económico de no tener el ingrediente

La justificación más directa de por qué merece la pena predecir la demanda es económica. Cuando un restaurante se queda sin un ingrediente clave, no vende el plato que lo contiene. El coste inmediato no es el precio del ingrediente que falta, sino el margen que deja de ingresar por no poder vender ese plato.

$$Margen\ perdido\ por\ rotura\ de\ stock = Precio\ de\ venta\ del\ plato - Coste\ total\ de\ ingredientes\ del\ plato$$

Si ese plato es además uno de los más caros de la carta, el impacto es mayor. Por ejemplo, si un restaurante tiene un solomillo a 24 € con un coste de ingredientes de 9 €, el margen por plato es de 15 €. Si por una rotura de stock ese plato no puede servirse en 10 ocasiones durante el fin de semana, la pérdida directa es de 150 € en margen no generado, que no aparece en ningún balance pero que es completamente real.

A esto se suma otro efecto: la pérdida de cliente. Un comensal que llega con la intención de pedir un plato concreto y se encuentra con que no está disponible puede decidir no volver, lo que en restauración, donde la fidelización es uno de los principales motores de rentabilidad, tiene un coste a largo plazo muy superior a la venta perdida ese día (un cliente habitual que gasta 35 € al mes supone 420 € al año en ventas). Conviene matizar que el margen directo perdido es en realidad un máximo: cuando un cliente no puede pedir el plato que quería, normalmente pide otro, así que el restaurante no deja de ingresar del todo y el coste real es algo menor.

Estos dos efectos (pérdida de margen directa y pérdida de cliente) son la razón por la que predecir bien la demanda no es un ejercicio técnico accesorio, sino una herramienta con impacto económico directo. Evitar una sola rotura de stock a la semana en un plato de alto margen puede recuperar con creces el coste mensual de la suscripción a Zipo.

Capítulo 6. Desarrollo técnico: pipeline analítico de predicción y alertas de stock

Este capítulo desarrolla la parte técnica del pipeline analítico que predice la demanda de cada producto y genera la orden de compra automatizada. A diferencia de los capítulos anteriores, que describen Zipo a nivel de diseño, aquí hay código funcional ejecutándose sobre datos sintéticos y resultados concretos que demuestran que el sistema funciona.

El pipeline implementa dos cosas conectadas entre sí: por un lado, la predicción de demanda producto a producto para los próximos 7 días; por otro, el sistema de alertas que convierte esa predicción en una orden de compra agrupada por proveedor. Todo está implementado en Python y documentado en un Jupyter Notebook adjunto como anexo.

6.1 Dataset sintético: simulación del Restaurante El Mirador

Como no ha sido posible acceder a datos reales de ningún restaurante, el desarrollo usa un dataset sintético que simula un restaurante ficticio, el Restaurante El Mirador, generado específicamente para este trabajo con apoyo de herramientas de IA para la creación de datos de prueba. El conjunto de datos parte de un nivel de ventas base para cada producto, coherente con su tipo y su rotación esperada, y sobre ese nivel se aplican tres efectos sucesivos: una variación según el día de la semana (más ventas en fin de semana, mínimo los lunes), una estacionalidad a lo largo del año (mayor demanda en verano y descenso hacia el otoño) y una componente aleatoria que reproduce el ruido natural de las ventas diarias. De esta forma, las series resultantes se comportan como las de un restaurante real sin corresponder a ninguno. Los datos no representan a ningún restaurante real y se han creado exclusivamente con fines académicos. El procedimiento de generación se detalla en el Anexo III.

La generación del dataset incorpora los patrones documentados en la literatura del sector (Hostelería de España, 2023; Garzón-Mosquera, 2024), lo que garantiza que los datos simulados se parezcan al comportamiento real de un restaurante. En concreto, el script reproduce tres fenómenos:

Patrón semanal. Los viernes y sábados tienen una demanda entre 1,5 y 2 veces superior a la del lunes. Esto refleja el comportamiento típico de un restaurante de cena en zona urbana.

Estacionalidad anual. El verano tiene un 25-35% más de demanda que la media, y enero un 20% menos. Esto recoge la subida de actividad en temporada alta y la caída del período post-Navidad.

Variabilidad aleatoria. Cada producto tiene un nivel de ruido distinto según su volatilidad natural. Las carnes y pescados oscilan más; las bebidas y los productos con larga vida útil son más estables.

El dataset cubre 180 días para 11 productos distribuidos en cinco categorías: carnes (Solomillo, Pollo), pescados (Lubina, Salmón), verduras (Tomate, Lechuga, Patata), lácteos (Nata, Queso) y bebidas (Vino tinto, Cerveza). El resultado son 1.980 registros de ventas diarias (180 días × 11 productos), distribuidos en tres hojas: ventas_diarias, stock_actual y parametros.

6.2 Análisis exploratorio

Antes de modelar, el pipeline analiza los patrones de demanda del restaurante. La Figura 6.1 muestra las ventas medias por día de la semana agregadas para todos los productos.

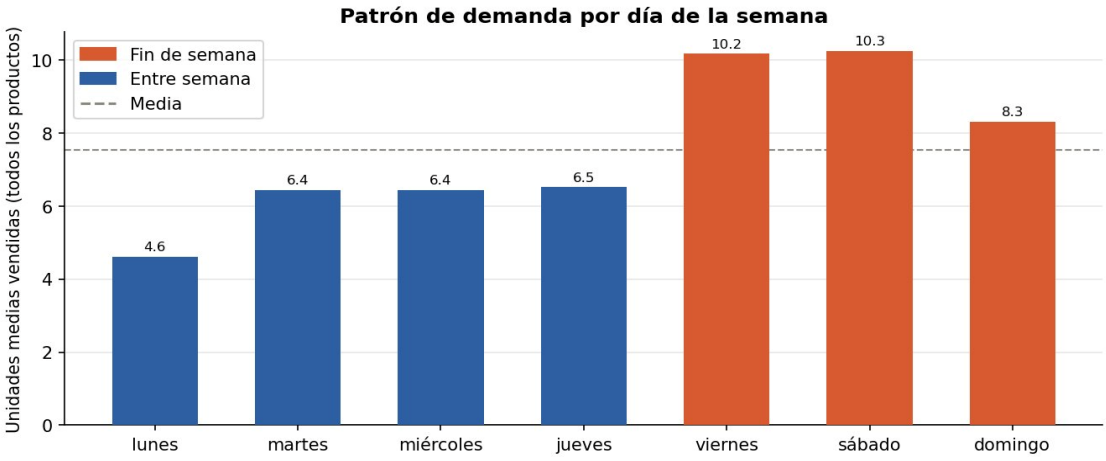

Figura 6.1. Patrón de demanda medio por día de la semana (todos los productos).

Fuente: Elaboración propia a partir de datos sintéticos simulados.

El resultado es muy claro: los lunes se venden 4,6 unidades de media; los viernes, 10,2. El ratio entre los dos días es de 2,2, lo que se calcula como:

$$ratio = \frac{ventas_{medias_{viernes}}}{ventas_{medias_{lunes}}} = \frac{10{,}2}{4{,}6} \approx 2{,}22$$

Este 2,22 significa que el día de la semana es, con diferencia, el factor que más afecta a la demanda diaria de un producto, y cualquier modelo que ignore este factor va a fallar sistemáticamente. Por eso el modelo de predicción incluye el día de la semana como componente explícito (lo veremos en 6.3).

La Figura 6.2 muestra la evolución temporal de cuatro productos representativos con su media móvil de 7 días. La media móvil es una herramienta básica para suavizar series temporales: para cada día, elimina el ruido diario y deja ver la tendencia. La fórmula es:

$$\overline{media\ móvil}_T = \frac{\sum_{t-6}^{T} ventas_t}{7}$$

En la figura se ve claramente la tendencia estacional: la demanda del verano (julio-agosto) es notablemente superior a la del otoño, con una pequeña recuperación en diciembre por el período navideño. El ruido diario es considerable, sobre todo en carnes como el Solomillo, lo que refuerza la necesidad de un modelo estadístico bien construido frente a reglas simples del tipo “pide lo mismo que la semana pasada”.

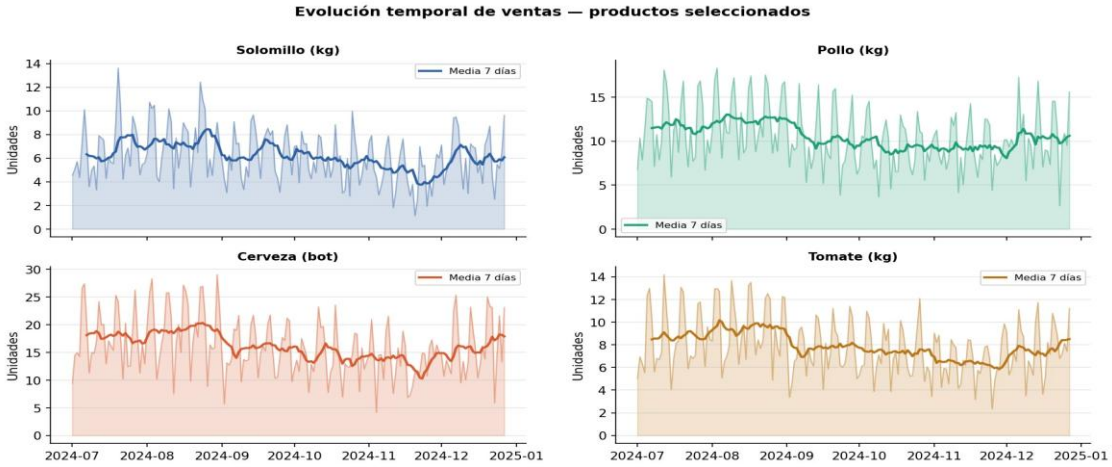

Figura 6.2. Evolución temporal de ventas para cuatro productos seleccionados. Línea continua: media móvil de 7 días.

Fuente: Elaboración propia a partir de datos sintéticos simulados.

6.3 Modelo de predicción de demanda

Elección del modelo. El modelo seleccionado es Prophet, la librería de predicción de series temporales desarrollada por Meta y publicada como código abierto (Taylor & Letham, 2018). Prophet implementa un modelo de descomposición aditiva:

$$demanda_t = tendencia_t + \text{estacionalidad semanal}_t + error_t$$

La idea es simple: la demanda de cada día se explica como la suma de tres piezas. La tendencia es la dirección general en la que se están moviendo las ventas a largo plazo (subiendo, bajando, plana). La estacionalidad semanal es el efecto del día de la semana (el +5,6 del viernes respecto al lunes que vimos antes). Y el error es lo que queda sin explicar, que será ruido aleatorio si el modelo está bien construido.

Prophet se eligió frente a otras alternativas como ARIMA, las redes neuronales (LSTM) o el gradient boosting por tres razones. La primera es la interpretabilidad: el modelo descompone la predicción en componentes (tendencia, estacionalidad y festivos) que el responsable del restaurante puede entender y comprobar, una característica de diseño que sus autores señalan como uno de sus objetivos principales (Taylor & Letham, 2018). La segunda es su buen comportamiento con series cortas, algo habitual en este contexto, ya que muchos restaurantes no dispondrán de más que unos meses de historial digitalizado al empezar a usar Zipo. La tercera es su bajo coste de cómputo: puede ejecutarse a diario sobre cientos de productos sin necesidad de tarjeta gráfica ni infraestructura especializada.

Implementación. La función principal del código, entrenar_prophet, recibe el historial de ventas de un producto y reserva los últimos 30 días para comprobar después si el modelo acierta. Esta técnica se llama hold-out: apartar una parte de los datos que el modelo no ve durante el entrenamiento, para luego medir cómo de buenas son sus predicciones sobre datos que no conocía. Es la forma estándar de evaluar un modelo predictivo de manera honesta.

El código de la función entrenar_prophet se incluye al final del documento como Anexo I. La función hace cuatro cosas: prepara los datos en el formato que Prophet espera (columnas ds para la fecha e y para la variable a predecir), divide en train (todos los datos menos los últimos 30 días) y test (esos últimos 30 días), entrena el modelo solo con los datos de train, y evalúa el modelo prediciendo el período de test y comparando con la realidad mediante el MAPE.

El MAPE (Mean Absolute Percentage Error o error porcentual absoluto medio) es la métrica que se usa para medir la precisión del modelo. Se calcula así:

$$MAPE = media\ de \frac{\left|demanda_{real} - demanda_{predicha}\right|}{demanda_{real}} \times 100$$

Es decir, para cada día del período de test, calculamos cuánto se ha equivocado el modelo en porcentaje sobre la demanda real, y luego promediamos esos errores. Un MAPE del 25% significa que, de media, el modelo se equivoca en un 25% sobre el valor real. Por ejemplo, si el Solomillo se vende realmente 8 unidades un día y el modelo predice 6, el error de ese día es |8 - 6| / 8 = 25%.

Resultados del modelo y descomposición. La Figura 6.3 muestra las previsiones de demanda para los próximos 7 días en seis productos, junto con el ajuste del modelo sobre el historial y el intervalo de confianza al 80%. El intervalo de confianza es la banda sombreada alrededor de la línea de predicción: indica que el modelo está un 80% seguro de que la demanda real va a caer dentro de esa banda. Cuanto más estrecha es la banda, más seguro está el modelo.

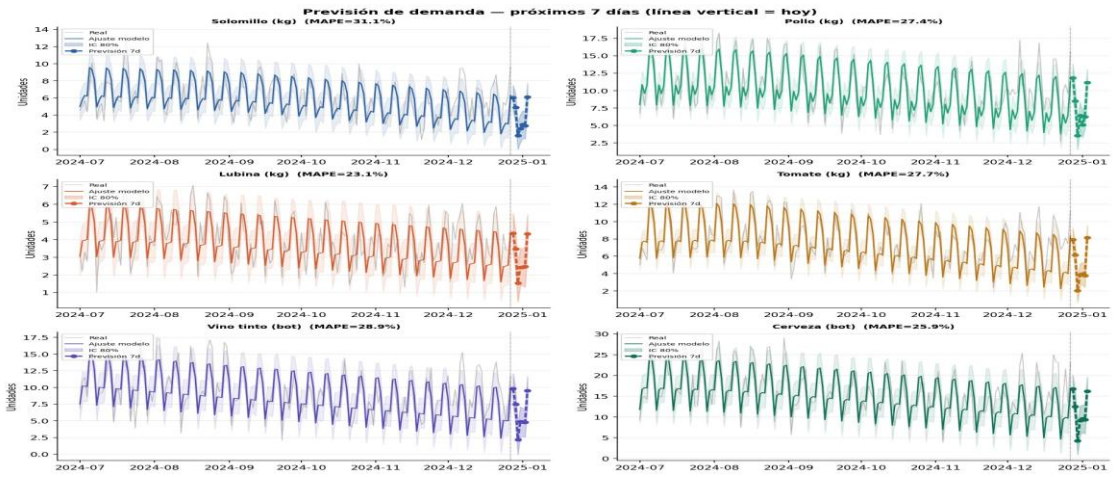

Figura 6.3. Previsión de demanda para los próximos 7 días por producto (línea vertical = hoy). Banda sombreada: intervalo de confianza al 80%.

Fuente: Elaboración propia a partir de datos sintéticos simulados.

Los MAPEs obtenidos están entre el 23% y el 31% según el producto. Los valores más altos los tienen las carnes (Solomillo, 31,1%; Pollo, 27,4%) que son los productos con más variabilidad diaria. Estos valores son coherentes con la literatura sobre predicción de demanda en

restauración con series temporales cortas (Garzón-Mosquera, 2024), y se consideran aceptables para tomar decisiones de aprovisionamiento siempre que se combinen con un margen de seguridad adecuado (Taylor, 2008).

La Figura 6.4 muestra la descomposición del modelo para el Solomillo, que es el producto con más variabilidad. La descomposición consiste en separar las dos piezas de las que el modelo construye la predicción para poder mirarlas por separado:

El panel de arriba muestra la tendencia, que va bajando a lo largo del período estudiado (cuadra con la caída de demanda del verano al invierno). El de abajo muestra el efecto del día de la semana: viernes y sábado suman entre +2 y +2,5 unidades sobre la media, mientras que el lunes resta unas 2,5; un viernes vendemos en promedio unas 5 unidades más de Solomillo que un lunes. Lo interesante de esta descomposición es que el responsable del restaurante puede leer el gráfico directamente y entender por qué el modelo predice lo que predice, sin necesidad de saber estadística.

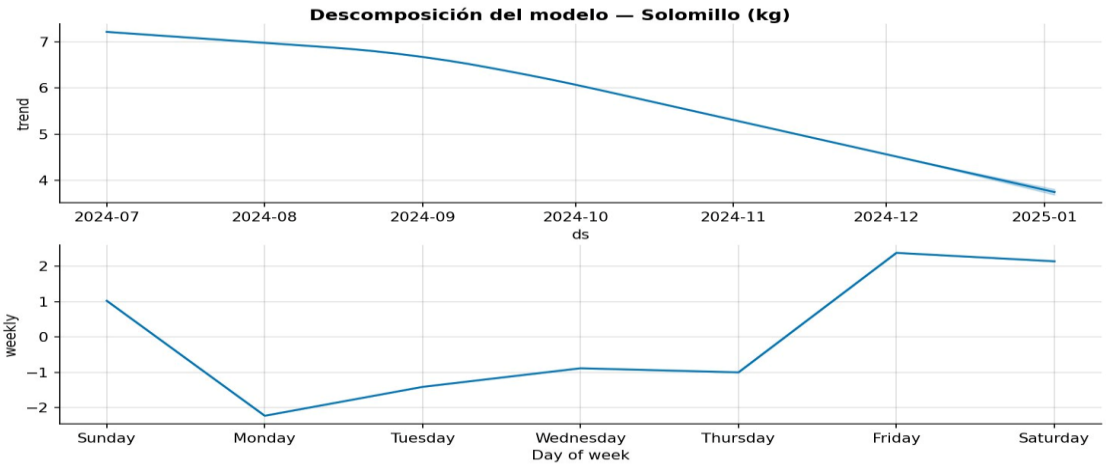

Figura 6.4. Descomposición del modelo Prophet para el Solomillo (kg): tendencia temporal y efecto del día de la semana.

Fuente: Elaboración propia a partir de datos sintéticos simulados.

6.4 Sistema de alertas de stock

Lógica de cálculo. El sistema de alertas toma las previsiones que ha generado el modelo y las compara con el stock que hay ahora mismo en el restaurante para decidir si hace falta pedir más, y si es así, cuánto. La lógica son cuatro cálculos encadenados:

$$consumo_{diario} = \frac{demanda_{prevista_{7d}}}{7}$$

$$margen_{seguridad} = consumo_{diario} \times stock_{minimo_{dias}}$$

$$stock_{necesario} = demanda_{prevista_{7d}} + margen_{seguridad}$$

$$cantidad_{a_{pedir}} = \max(0, stock_{necesario} - stock_{actual})$$

La lógica de cada paso es la siguiente:

La lógica encadenada se entiende así: primero se divide la demanda total prevista para los próximos 7 días entre 7 para obtener la demanda diaria media; segundo, se calcula el margen de seguridad como un colchón de días de stock extra (configurable por producto según su vida útil: 2 días para un pescado fresco, 5 o más para una botella de vino); tercero, se suma demanda prevista más colchón para obtener el stock necesario; y cuarto, se calcula lo que falta para llegar a ese stock necesario, con un máx(0, ...) que garantiza que nunca se genera un pedido negativo si ya hay stock suficiente. Este cálculo por días de colchón es una versión simplificada del stock de seguridad descrito en la sección 5.7.3: en lugar de estimar la volatilidad de cada producto con la fórmula Z × σ × √Plazo de entrega, el prototipo usa un número fijo de días configurado por producto, que es más sencillo de calibrar cuando todavía no hay suficiente histórico para medir la volatilidad real. En la versión completa de Zipo, este colchón se sustituiría por el cálculo de stock de seguridad basado en la volatilidad.

Veamos el cálculo completo con un producto concreto del dataset, el Solomillo:

Ejemplo: Solomillo. Demanda prevista para los próximos 7 días: 38 kg. Stock mínimo configurado: 3 días. Stock actual en el almacén: 12 kg.

$$consumo_{diario} = \frac{38}{7} \approx 5{,}43 \frac{kg}{día}$$

$$margen_{seguridad} = 5{,}43 \times 3 \approx 16{,}3\ kg$$

$$stock_{necesario} = 38 + 16{,}3 \approx 54{,}3\ kg$$

$$cantidad_{a_{pedir}} = \max(0, 54{,}3 - 12) \approx 42{,}3\ kg$$

Es decir, para cubrir los próximos 7 días con un colchón de 3 días extra, hay que pedir 42,3 kg de Solomillo. Si solo tuviéramos en cuenta la demanda prevista (38 kg) sin colchón, pediríamos 26 kg y nos quedaríamos justos al menor desvío de la demanda.

Niveles de alerta y orden de compra. El sistema clasifica cada producto en uno de tres niveles según su situación de stock:

OK (verde). Stock actual > stock necesario. Hay suficiente para cubrir la demanda prevista y el margen de seguridad. No hace falta pedir.

Atención (amarillo). Stock actual cubre la demanda prevista, pero no llega a cubrir también el margen de seguridad. Hay que pedir aunque no sea urgente.

Crítico (rojo). Stock actual no cubre ni siquiera la demanda prevista de 7 días. Hay riesgo real de rotura de stock. Hay que pedir cuanto antes.

Los productos en Atención o Crítico se agregan automáticamente en una orden de compra agrupada por proveedor. Así, en lugar de generar 11 pedidos separados, el sistema produce uno por cada proveedor con todos los productos que le corresponden, listo para que el responsable lo apruebe con un clic.

6.5 Resultados del pipeline completo

La Figura 6.5 muestra el panel de alertas de stock generado por el pipeline para el horizonte de 7 días. El panel izquierdo representa los días de stock disponible por producto: la línea discontinua marca el horizonte de 7 días. Todos los productos están por debajo de esa línea, lo que indica que el stock inicial configurado en el dataset es deliberadamente ajustado para forzar la generación de alertas y demostrar que el sistema reacciona como debe.

El cálculo de días de stock disponible es simplemente:

El cálculo de días de stock disponible es simplemente dias_stock_disponible = stock_actual / consumo_diario_previsto. Por ejemplo, con 12 kg de Solomillo y un consumo medio previsto de 5,43 kg/día, los días de stock son 12 / 5,43 ≈ 2,2: si no pedimos más, nos quedamos sin Solomillo en algo más de dos días. Esta métrica traduce el stock (kilos, litros o unidades) a una unidad común e intuitiva, lo que permite al responsable comparar de un vistazo qué productos son más urgentes. El panel derecho de la Figura 6.5 compara el stock actual con el necesario para 7 días: en todos los casos el necesario es mayor, lo que confirma que hay que generar la orden de compra para el conjunto completo de productos.

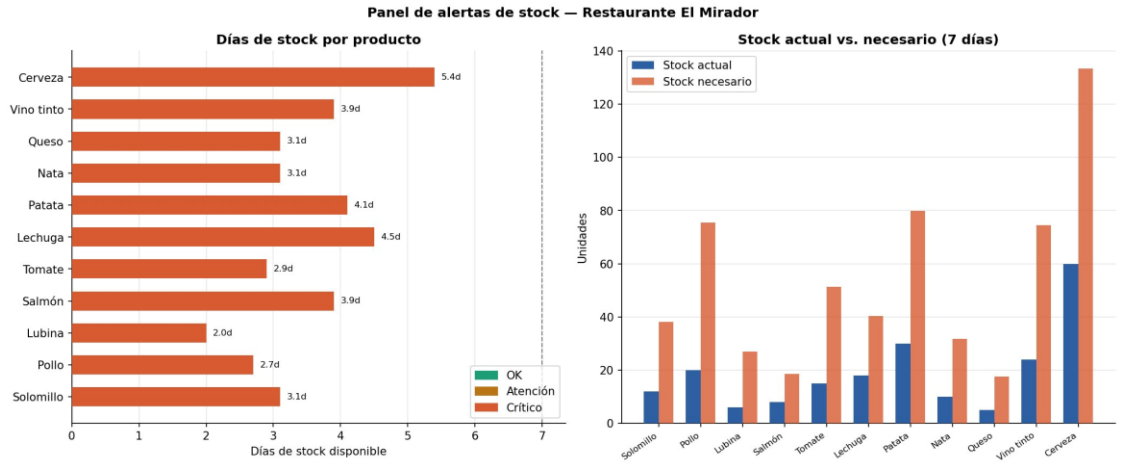

Figura 6.5. Panel de alertas de stock: días de stock disponible por producto (izquierda) y comparativa stock actual vs. necesario para 7 días (derecha).

Fuente: Elaboración propia a partir de datos sintéticos simulados.

Este resultado demuestra el funcionamiento completo del pipeline: partiendo de un historial de ventas, el sistema predice la demanda futura producto a producto, calcula el stock necesario con margen de seguridad, clasifica cada producto en su nivel de alerta y genera una orden de compra accionable. En un escenario real, el responsable del restaurante recibiría esta información cada mañana en su dashboard, con la orden de compra lista para revisar y aprobar con un solo clic.

6.6 Limitaciones del desarrollo

El desarrollo tiene tres limitaciones importantes que conviene reconocer.

Uso de datos sintéticos. El dataset, aunque reproduce los patrones documentados en la literatura, no puede replicar la complejidad de los datos reales. No incluye eventos puntuales (un fin de semana grande de competiciones deportivas, una promoción), no captura la correlación entre ingredientes que comparten varios platos del menú, y no recoge el efecto de variables externas como la meteorología. La validación con datos reales es un paso necesario antes de poner Zipo en producción.

Horizonte de predicción limitado a 7 días. El modelo está calibrado para predecir a 7 días vista. Para horizontes más largos (por ejemplo, hacer un pedido mensual a un proveedor de bebidas) la incertidumbre crece y los intervalos de confianza se vuelven tan anchos que dejan de ser útiles. Para estos casos haría falta extender el modelo o usar uno más complejo.

Ausencia de escandallos en el prototipo. En la versión completa de Zipo, el sistema descontaría del stock el consumo de ingredientes derivado de cada plato vendido, no sólo las ventas directas de producto. Esto requiere una ficha técnica completa por plato (qué ingredientes lleva y en qué cantidades) que en este prototipo se ha simplificado, prediciéndose directamente la venta de cada ingrediente en lugar de derivarla de las ventas de platos.

Capítulo 7. Conclusiones

7.1 Conclusiones principales

Sobre el diseño de la plataforma. El diseño funcional y arquitectónico desarrollado en los capítulos 4 y 5 demuestra que una plataforma con las características descritas es viable con tecnologías maduras y de código abierto. La arquitectura modular permite lanzar la plataforma con un subconjunto de módulos e ir añadiendo los demás de forma incremental, reduciendo el riesgo técnico y financiero. El sistema de donaciones convierte a Zipo en una herramienta de cumplimiento regulatorio (Ley 1/2025) además de operativa, ampliando la propuesta de valor más allá del ahorro económico directo.

Sobre el desarrollo analítico. El pipeline desarrollado en el capítulo 6 demuestra que la combinación de un modelo de predicción interpretable (Prophet) con un sistema de alertas

basado en márgenes de seguridad produce recomendaciones de aprovisionamiento accionables a partir del historial de ventas. Los MAPEs obtenidos (23-31%) son coherentes con la literatura sobre predicción de demanda en restauración con series temporales cortas. Más allá de los números, la pieza más importante es la interpretabilidad: que el responsable pueda mirar la descomposición del modelo y entender por qué le sugiere pedir más Solomillo el jueves que el martes reduce el rechazo al cambio y hace que la herramienta se use de verdad.

Sobre la viabilidad del modelo de negocio. El modelo financiero desarrollado como soporte al trabajo apunta a fundamentos sólidos: margen bruto típico del sector SaaS B2B, ratio LTV/CAC consistentemente superior a 4,5x desde el primer año, y trayectoria razonable hacia el breakeven operativo sin requerir grandes rondas de financiación. Estas cifras son consistentes con los estándares del sector y respaldan la viabilidad económica del proyecto a medio plazo.

7.2 Limitaciones del trabajo

El trabajo presenta cuatro limitaciones principales que conviene reconocer con honestidad.

Ausencia de validación con datos reales. El uso de datos sintéticos, aunque justificado y documentado, impide afirmar con certeza que el pipeline produciría los mismos resultados sobre datos reales de un restaurante. Los datos reales incorporan una complejidad que ningún proceso de simulación puede replicar del todo: relaciones entre ingredientes que comparten varios platos, efecto de cambios en el menú, impacto de campañas de marketing o cierres inesperados.

Alcance del prototipo técnico. El pipeline desarrollado cubre la parte central del módulo analítico, pero la plataforma completa incluye funcionalidades (gestión de escandallos, integración con TPV, portal para entidades sociales) cuya implementación excede el alcance de un TFG.

Falta de entrevistas con restaurantes. El análisis de mercado se apoya en fuentes secundarias sólidas, pero la validación de la propuesta de valor mediante entrevistas con propietarios y gestores habría reforzado la base empírica del trabajo.

Horizonte de predicción limitado. El modelo está calibrado para 7 días. Para productos con ciclos de aprovisionamiento más largos o pedidos de mayor volumen sería necesario extender el horizonte, lo que requiere modelos más complejos y más datos históricos.

7.3 Líneas de desarrollo futuro

El trabajo abre varias líneas de continuación natural:

Validación con datos reales. El siguiente paso es ejecutar el pipeline sobre datos reales de uno o varios restaurantes colaboradores, calibrando el modelo con sus patrones específicos y validando empíricamente las hipótesis sobre el comportamiento de la demanda.

Incorporación de escandallos. Integrar la ficha técnica de cada plato para calcular el consumo de ingredientes desde las ventas del TPV con mayor granularidad, eliminando la simplificación del prototipo y mejorando la precisión de las alertas.

Extensión del modelo predictivo. Explorar modelos más complejos (gradient boosting, redes recurrentes) para horizontes más largos o productos con patrones especialmente irregulares.

Desarrollo del MVP. Construir la primera versión desplegable integrando los módulos de inventario y predicción con un TPV real, y lanzarla en piloto con un grupo reducido de restaurantes en Madrid o Barcelona.

Medición del impacto real. Diseñar un experimento controlado que compare los resultados operativos de restaurantes que usen Zipo frente a un grupo de control, generando evidencia empírica del valor creado.

Referencias bibliográficas

Asisman. (2024). *Gstock: sistema de gestión de las compras, escandallos, stock y consumos para hoteles y restaurantes.* https://info.asisman.com/gstock-sistema-de-gestion-de-las-compras-escandallos-stock-y-consumos-para-hoteles-y-restaurantes/

Capterra. (2024). *Apicbase Restaurant Management Reviews.* https://www.capterra.com/p/171584/Apicbase-Restaurant-Management/reviews/

Comisión Europea. (2020). *Farm to Fork Strategy: For a fair, healthy and environmentally-friendly food system.* Publications Office of the European Union. https://ec.europa.eu/food/horizontal-topics/farm-fork-strategy_en

EFFAT. (2022). *The hospitality sector in Europe: Key figures and trends.* European Federation of Food, Agriculture and Tourism Trade Unions.

Eurostat. (2022). *Tourism and the hospitality sector in the EU.* Statistical Office of the European Union. https://ec.europa.eu/eurostat

Food and Agriculture Organization of the United Nations (FAO). (2019). *The state of food and agriculture: Moving forward on food loss and waste reduction.* FAO. https://www.fao.org/publications/sofa/2019/en/

Garzón-Mosquera, F. F. (2024). Utilización de las mermas para la optimización de procesos operativos en el restaurante. *Revista de Gastronomía y Cocina, 3*(1), 62–88.

Haddock. (2025). *Haddock: software de gestión para bares y restaurantes.* https://www.haddock.app

Hostelería de España. (2023). *Informe anual de la hostelería española 2023.* Federación Española de Hostelería. https://www.hosteleria.com

Instituto Nacional de Estadística (INE). (2023). *Directorio Central de Empresas (DIRCE).* INE. https://www.ine.es

Instituto Nacional de Estadística (INE). (2024). *Directorio Central de Empresas (DIRCE): empresas por CNAE a 1 de enero de 2024.* INE. https://www.ine.es

Jefatura del Estado. (2022). *Ley 7/2022, de 8 de abril, de residuos y suelos contaminados para una economía circular.* Boletín Oficial del Estado, núm. 85, de 9 de abril de 2022. https://www.boe.es/buscar/act.php?id=BOE-A-2022-5809

Jefatura del Estado. (2025). *Ley 1/2025, de 1 de abril, de prevención de las pérdidas y el desperdicio alimentario.* Boletín Oficial del Estado, núm. 79, de 2 de abril de 2025. https://www.boe.es/buscar/act.php?id=BOE-A-2025-6597

MarketMan. (2024). *Software de gestión de inventario para restaurantes.* https://es.marketman.com

McKinsey & Company. (2022). *The digital future of hospitality.* McKinsey Global Institute.

Ministerio de Agricultura, Pesca y Alimentación. (2022). *Estrategia nacional de reducción de pérdidas y desperdicio alimentario.* Gobierno de España. https://www.mapa.gob.es

National Restaurant Association. (2023). *Restaurant industry 2023 state of the industry report.* National Restaurant Association Educational Foundation.

Osterwalder, A., & Pigneur, Y. (2010). *Business model generation: A handbook for visionaries, game changers, and challengers.* John Wiley & Sons.

Osterwalder, A., Pigneur, Y., Bernarda, G., & Smith, A. (2014). *Value proposition design: How to create products and services customers want.* John Wiley & Sons.

Programa de las Naciones Unidas para el Medio Ambiente (PNUMA). (2021). *Food waste index report 2021.* United Nations Environment Programme. https://www.unep.org/resources/report/unep-food-waste-index-report-2021

Sociedad Mercantil Estatal para la Gestión de la Innovación y las Tecnologías Turísticas (SEGITTUR). (2023). *Índice de madurez digital del sector turístico español.* Secretaría de Estado de Turismo, Gobierno de España. https://www.segittur.es

Taylor, J. W. (2008). A comparison of univariate time series methods for forecasting intraday arrivals at a call center. *Management Science, 54*(2), 253–265. https://doi.org/10.1287/mnsc.1070.0786

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician, 72*(1), 37–45. https://doi.org/10.1080/00031305.2017.1380080

Too Good To Go. (2023). *Impact report 2023.* https://www.toogoodtogo.com

Declaración de Uso de Herramientas de Inteligencia Artificial Generativa en Trabajos Fin de Grado

ADVERTENCIA: Desde la Universidad consideramos que ChatGPT u otras herramientas similares son herramientas muy útiles en la vida académica, aunque su uso queda siempre bajo la responsabilidad del alumno, puesto que las respuestas que proporciona pueden no ser veraces. En este sentido, NO está permitido su uso en la elaboración del Trabajo fin de Grado para generar código porque estas herramientas no son fiables en esa tarea. Aunque el código funcione, no hay garantías de que metodológicamente sea correcto, y es altamente probable que no lo sea.

Por la presente, yo, Diego Gutiérrez-Colomer, estudiante de Administración de Empresa y Business Analytics de la Universidad Pontificia Comillas al presentar mi Trabajo Fin de Grado titulado "ZIPO: Plataforma de optimización de inventarios para restaurantes mediante analítica de datos", declaro que he utilizado la herramienta de Inteligencia Artificial Generativa ChatGPT u otras similares de IAG de código sólo en el contexto de las actividades descritas a continuación:

1. **Brainstorming de ideas de investigación:** Utilizado para idear y esbozar posibles áreas de investigación.
2. **Crítico:** Para encontrar contra-argumentos a una tesis específica que pretendo defender.
3. **Referencias:** Usado conjuntamente con otras herramientas, como Science, para identificar referencias preliminares que luego he contrastado y validado.
4. **Metodólogo:** Para descubrir métodos aplicables a problemas específicos de investigación.
5. **Interpretador de código:** Para realizar análisis de datos preliminares.
6. **Constructor de plantillas:** Para diseñar formatos específicos para secciones del trabajo.
7. **Corrector de estilo literario y de lenguaje:** Para mejorar la calidad lingüística y estilística del texto.
8. **Generador de datos sintéticos de prueba:** Para la creación de conjuntos de datos ficticios.

9. **Generador previo de diagramas de flujo y contenido:** Para esbozar diagramas iniciales.
10. **Sintetizador y divulgador de libros complicados:** Para resumir y comprender literatura compleja.
11. **Revisor:** Para recibir sugerencias sobre cómo mejorar y perfeccionar el trabajo con diferentes niveles de exigencia.
12. **Traductor:** Para traducir textos de un lenguaje a otro.

Afirmo que toda la información y contenido presentados en este trabajo son producto de mi investigación y esfuerzo individual, excepto donde se ha indicado lo contrario y se han dado los créditos correspondientes (he incluido las referencias adecuadas en el TFG y he explicitado para que se ha usado ChatGPT u otras herramientas similares). Soy consciente de las implicaciones académicas y éticas de presentar un trabajo no original y acepto las consecuencias de cualquier violación a esta declaración.
Fecha: 16.06.2026
Firma: _DIEGOGC_____

Anexo I. Código del modelo de predicción (Prophet)

Función principal entrenar_prophet, implementada en Python, que entrena el modelo de predicción de demanda y devuelve la previsión a 7 días junto con su MAPE y el objeto del modelo entrenado. La función reserva los últimos 30 días del historial como hold-out para evaluar el modelo de forma honesta sobre datos que no ha visto durante el entrenamiento.

```
def entrenar_prophet(df_prod, periodos_pred=7, holdout=30):
    df = df_prod[['fecha','unidades_vendidas']]\
        .rename(columns={'fecha':'ds','unidades_vendidas':'y'})
    train, test = df.iloc[:-holdout], df.iloc[-holdout:]
    model = Prophet(weekly_seasonality=True,
                    seasonality_mode='additive', interval_width=0.80)
    model.fit(train)
    future   = model.make_future_dataframe(periods=holdout + periodos_pred)
    forecast = model.predict(future)
    merged = test.merge(forecast[['ds','yhat']], on='ds')
    mape = np.mean(np.abs((merged['y']-merged['yhat'])
                          / merged['y'].clip(lower=0.1))) * 100
    return forecast, round(mape, 1), model
```

Anexo II. Código del pipeline analítico completo (Python)

Se incluye a continuación el código completo del pipeline analítico desarrollado en Python (Jupyter Notebook), organizado por las mismas secciones descritas en el Capítulo 6. El código abarca desde la carga del dataset hasta la generación automática de la orden de compra.

1. Importación de librerías

```
!pip install prophet
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import warnings
from datetime import date, timedelta

# Prophet (instalación: pip install prophet)
from prophet import Prophet
from prophet.diagnostics import cross_validation, performance_metrics

warnings.filterwarnings('ignore')
plt.rcParams.update({'font.family': 'DejaVu Sans', 'figure.dpi': 120,
                     'axes.spines.top': False, 'axes.spines.right': False})

AZUL   = '#2E5FA3'
VERDE  = '#1D9E75'
CORAL  = '#D85A30'
AMBER  = '#BA7517'
GRIS   = '#888780'
print('Librerías cargadas correctamente.')
```

2. Carga y exploración del dataset

```
RUTA = 'Zipo_datos_restaurante.xlsx'

ventas  = pd.read_excel(RUTA, sheet_name='ventas_diarias', parse_dates=['fecha'])
stock   = pd.read_excel(RUTA, sheet_name='stock_actual')
params  = pd.read_excel(RUTA, sheet_name='parametros')

print(f'Ventas: {ventas.shape[0]:,} filas · {ventas["producto"].nunique()} productos
· '
      f'{ventas["fecha"].min().date()} → {ventas["fecha"].max().date()}')
ventas.head()
```

3. Análisis exploratorio

```
# Ventas totales por producto (ranking)
ranking = (ventas.groupby('producto')['unidades_vendidas']
           .sum().sort_values(ascending=False).reset_index())
ranking.columns = ['Producto', 'Total vendido (180 días)']
print(ranking.to_string(index=False))
# Patrón semanal: ventas medias por día de la semana

# Asegurar que fecha está en formato fecha
ventas['fecha'] = pd.to_datetime(ventas['fecha'], errors='coerce')

# Eliminar posibles fechas no válidas
ventas = ventas.dropna(subset=['fecha'])

# Crear día de la semana manualmente para evitar problemas de idioma / mayúsculas
mapa_dias = {
    0: 'lunes',
    1: 'martes',
    2: 'miércoles',
    3: 'jueves',
    4: 'viernes',
    5: 'sábado',
    6: 'domingo'
}

orden_dias = ['lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado',
'domingo']

ventas['dia_semana'] = ventas['fecha'].dt.weekday.map(mapa_dias)
```

```
# Calcular media por día y asegurar que todos los días existen
semanal = (
    ventas.groupby('dia_semana')['unidades_vendidas']
    .mean()
    .reindex(orden_dias)
    .fillna(0)
)

fig, ax = plt.subplots(figsize=(9, 4))

colors = [CORAL if d in ['viernes', 'sábado', 'domingo'] else AZUL for d in
semanal.index]

bars = ax.bar(
    semanal.index,
    semanal.values,
    color=colors,
    width=0.6,
    zorder=3
)

media_global = semanal.mean()

ax.axhline(
    media_global,
    color=GRIS,
    lw=1,
    ls='--',
    label='Media global'
)

ax.set_ylabel('Unidades medias vendidas (todos los productos)')
ax.set_title('Patrón de demanda por día de la semana', fontweight='bold')
ax.grid(axis='y', alpha=0.3, zorder=0)

p1 = mpatches.Patch(color=CORAL, label='Fin de semana')
p2 = mpatches.Patch(color=AZUL, label='Entre semana')

ax.legend(
    handles=[
        p1,
        p2,
        plt.Line2D([0], [0], color=GRIS, ls='--', label='Media')
    ],
    loc='upper left'
)

# Etiquetas encima de barras solo si el valor es válido
for bar in bars:
    height = bar.get_height()

    if np.isfinite(height):
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            height + 0.1,
            f'{height:.1f}',
            ha='center',
            va='bottom',
            fontsize=8
        )

plt.tight_layout()
plt.savefig('fig_patron_semanal.png', dpi=150, bbox_inches='tight')
plt.show()

# Ratio viernes/lunes evitando división por cero
if semanal['lunes'] != 0:
    print('Ratio viernes/lunes:', round(semanal['viernes'] / semanal['lunes'], 2))
```

```
else:
    print('No se puede calcular el ratio viernes/lunes porque lunes es 0')
# Evolución temporal de 4 productos representativos
prods_plot = ['Solomillo (kg)', 'Pollo (kg)', 'Cerveza (bot)', 'Tomate (kg)']
colores_prods = [AZUL, VERDE, CORAL, AMBER]

fig, axes = plt.subplots(2, 2, figsize=(12, 6), sharex=True)
axes = axes.flatten()

for i, (prod, color) in enumerate(zip(prods_plot, colores_prods)):
    serie = ventas[ventas['producto']==prod].set_index('fecha')['unidades_vendidas']
    media_movil = serie.rolling(7).mean()
    axes[i].fill_between(serie.index, serie, alpha=0.2, color=color)
    axes[i].plot(serie.index, serie, color=color, alpha=0.4, lw=0.8)
    axes[i].plot(media_movil.index, media_movil, color=color, lw=2, label='Media 7
días')
    axes[i].set_title(prod, fontweight='bold', fontsize=10)
    axes[i].set_ylabel('Unidades')
    axes[i].grid(axis='y', alpha=0.3)
    axes[i].legend(fontsize=8)

fig.suptitle('Evolución temporal de ventas — productos seleccionados',
fontweight='bold', y=1.01)
plt.tight_layout()
plt.savefig('fig_evolucion_temporal.png', dpi=150, bbox_inches='tight')
plt.show()
```

4. Modelado de la demanda con Prophet

```
def entrenar_prophet(df_prod, periodos_pred=7, holdout=30):
    """
    Entrena un modelo Prophet para un producto y devuelve:
    - forecast: previsión para los próximos `periodos_pred` días
    - mape: error porcentual absoluto medio en el periodo de hold-out
    - model: modelo entrenado
    """
    df = df_prod[['fecha','unidades_vendidas']].rename(
        columns={'fecha':'ds','unidades_vendidas':'y'})
    df = df.sort_values('ds').reset_index(drop=True)

    train = df.iloc[:-holdout]
    test  = df.iloc[-holdout:]

    model = Prophet(
        yearly_seasonality=False,
        weekly_seasonality=True,
        daily_seasonality=False,
        seasonality_mode='additive',
        interval_width=0.80
    )
    model.fit(train, iter=300)

    # Previsión hold-out + próximos 7 días
    future = model.make_future_dataframe(periods=holdout + periodos_pred)
    forecast = model.predict(future)

    # MAPE en hold-out
    pred_test = forecast[forecast['ds'].isin(test['ds'])][['ds','yhat']]
    merged = test.merge(pred_test, on='ds')
    mape = np.mean(np.abs((merged['y'] - merged['yhat']) /
merged['y'].clip(lower=0.1))) * 100

    return forecast, round(mape, 1), model

print('Función entrenar_prophet() definida.')
# Entrenamos el modelo para todos los productos
# (puede tardar 1-2 minutos)
productos_lista = ventas['producto'].unique()
resultados = {}
```

```
for prod in productos_lista:
    df_prod = ventas[ventas['producto'] == prod].copy()
    forecast, mape, model = entrenar_prophet(df_prod)
    resultados[prod] = {'forecast': forecast, 'mape': mape, 'model': model}
    print(f'  {prod:<22} MAPE = {mape:.1f}%')

print(f'\nMAPE medio: {np.mean([r["mape"] for r in resultados.values()]):.1f}%')
```

5. Visualización de previsiones

```
def plot_previsiones(prod, ax, color):
    fc = resultados[prod]['forecast']
    hist = ventas[ventas['producto']==prod][['fecha','unidades_vendidas']]
    hist = hist.rename(columns={'fecha':'ds','unidades_vendidas':'y'})

    # Separar histórico de previsión futura
    ultimo_dia_hist = hist['ds'].max()
    fc_hist  = fc[fc['ds'] <= ultimo_dia_hist]
    fc_futuro= fc[fc['ds'] >  ultimo_dia_hist]

    ax.plot(hist['ds'], hist['y'], color=GRIS, lw=0.8, alpha=0.5, label='Real')
    ax.plot(fc_hist['ds'], fc_hist['yhat'], color=color, lw=1.2, label='Ajuste
modelo')
    ax.fill_between(fc_hist['ds'], fc_hist['yhat_lower'], fc_hist['yhat_upper'],
                    color=color, alpha=0.1)
    ax.fill_between(fc_futuro['ds'], fc_futuro['yhat_lower'],
fc_futuro['yhat_upper'],
                    color=color, alpha=0.25, label='IC 80%')
    ax.plot(fc_futuro['ds'], fc_futuro['yhat'], color=color, lw=2,
            ls='--', marker='o', ms=5, label='Previsión 7d')
    ax.axvline(ultimo_dia_hist, color='black', lw=0.8, ls=':', alpha=0.6)
    ax.set_title(f'{prod}  (MAPE={resultados[prod]["mape"]}%)',
                 fontweight='bold', fontsize=9)
    ax.set_ylabel('Unidades')
    ax.grid(axis='y', alpha=0.3)
    ax.legend(fontsize=7, loc='upper left')

# Mostramos 6 productos en una cuadrícula 2×3
prods_vis = ['Solomillo (kg)','Pollo (kg)','Lubina (kg)',
             'Tomate (kg)','Vino tinto (bot)','Cerveza (bot)']
colores_vis = [AZUL, VERDE, CORAL, AMBER, '#534AB7', '#0F6E56']

fig, axes = plt.subplots(3, 2, figsize=(13, 11))
axes = axes.flatten()
for i, (prod, color) in enumerate(zip(prods_vis, colores_vis)):
    plot_previsiones(prod, axes[i], color)

fig.suptitle('Previsión de demanda — próximos 7 días (línea vertical = hoy)',
             fontweight='bold', fontsize=12)
plt.tight_layout()
plt.savefig('fig_previsiones.png', dpi=150, bbox_inches='tight')
plt.show()
```

6. Descomposición del modelo

```
from prophet.plot import plot_components

prod_ejemplo = 'Solomillo (kg)'
model = resultados[prod_ejemplo]['model']
fc    = resultados[prod_ejemplo]['forecast']

fig = plot_components(model, fc)
fig.suptitle(f'Descomposición del modelo — {prod_ejemplo}', fontweight='bold',
y=1.01)
fig.set_size_inches(10, 5)
plt.savefig('fig_componentes.png', dpi=150, bbox_inches='tight')
plt.show()
```

```
print('Tendencia y estacionalidad semanal visualizadas.')
```

7. Sistema de alertas de stock

```
HORIZONTE_DIAS = 7   # días de previsión que consideramos
fecha_hoy = ventas['fecha'].max()  # último día del historial = 'hoy'

def calcular_alertas(resultados, stock, params, horizonte=7):
    """
    Para cada producto calcula:
    - demanda_prevista: suma de yhat para los próximos `horizonte` días
    - stock_necesario: demanda_prevista + margen de seguridad
    - deficit: stock_necesario - stock_actual (negativo = sin alerta)
    - nivel: OK / Atención / Crítico
    - cantidad_pedir: cuánto hay que pedir (0 si no hace falta)
    """
    filas = []
    for prod in resultados:
        fc = resultados[prod]['forecast']
        mape = resultados[prod]['mape']

        # Previsión de los próximos `horizonte` días
        fut = fc[fc['ds'] > fecha_hoy].head(horizonte)
        demanda_prevista = max(0, fut['yhat'].sum())
        # Intervalo de confianza superior (escenario pesimista)
        demanda_alta = max(0, fut['yhat_upper'].sum())

        # Stock actual
        stock_row = stock[stock['producto'] == prod]
        if stock_row.empty:
            continue
        stock_actual = float(stock_row['stock_actual'].values[0])
        coste = float(stock_row['coste_unitario_eur'].values[0])
        unidad = stock_row['unidad'].values[0]

        # Parámetros
        param_row = params[params['producto'] == prod]
        min_dias  = int(param_row['stock_minimo_dias'].values[0])
        plazo     = int(param_row['plazo_entrega_dias'].values[0])
        proveedor = param_row['proveedor'].values[0]

        # Consumo diario medio previsto
        consumo_diario = demanda_prevista / horizonte
        # Margen de seguridad = consumo_diario * min_dias
        margen = consumo_diario * min_dias
        # Stock necesario = demanda prevista + margen
        stock_necesario = demanda_prevista + margen

        deficit = stock_necesario - stock_actual
        dias_stock = stock_actual / consumo_diario if consumo_diario > 0 else 999

        # Nivel de alerta
        if stock_actual >= stock_necesario:
            nivel = 'OK'
        elif stock_actual >= demanda_prevista:
            nivel = 'Atención'
        else:
            nivel = 'Crítico'

        cantidad_pedir = max(0, round(deficit, 2)) if deficit > 0 else 0
        valor_pedido = round(cantidad_pedir * coste, 2)

        filas.append({
            'Producto': prod,
            'Unidad': unidad,
            'Stock actual': round(stock_actual, 2),
            'Demanda prevista 7d': round(demanda_prevista, 2),
            'Stock necesario': round(stock_necesario, 2),
            'Días de stock': round(dias_stock, 1),
```

```
            'Alerta': nivel,
            'Cantidad a pedir': cantidad_pedir,
            'Valor pedido (€)': valor_pedido,
            'Proveedor': proveedor,
            'Plazo entrega (d)': plazo,
            'MAPE modelo (%)': mape
        })

    df_alertas = pd.DataFrame(filas)
    orden_nivel = {'Crítico': 0, 'Atención': 1, 'OK': 2}
    df_alertas['_ord'] = df_alertas['Alerta'].map(orden_nivel)
    df_alertas =
df_alertas.sort_values('_ord').drop(columns='_ord').reset_index(drop=True)
    return df_alertas

df_alertas = calcular_alertas(resultados, stock, params, HORIZONTE_DIAS)
print(f'Sistema de alertas calculado para {len(df_alertas)} productos.')
df_alertas
```

8. Visualización del panel de alertas

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# --- Panel izquierdo: días de stock disponible ---
colores_alerta = {'OK': VERDE, 'Atención': AMBER, 'Crítico': CORAL}
bar_colors = [colores_alerta[n] for n in df_alertas['Alerta']]
prods_cortos = [p.split('(')[0].strip() for p in df_alertas['Producto']]

bars = ax1.barh(prods_cortos, df_alertas['Días de stock'], color=bar_colors,
                height=0.6, zorder=3)
ax1.axvline(7, color='black', lw=1, ls='--', alpha=0.5, label='Horizonte 7 días')
ax1.set_xlabel('Días de stock disponible')
ax1.set_title('Días de stock por producto', fontweight='bold')
ax1.grid(axis='x', alpha=0.3, zorder=0)
for bar, val in zip(bars, df_alertas['Días de stock']):
    ax1.text(val+0.1, bar.get_y()+bar.get_height()/2,
             f'{val:.1f}d', va='center', fontsize=8)
p_ok  = mpatches.Patch(color=VERDE, label='OK')
p_at  = mpatches.Patch(color=AMBER, label='Atención')
p_cr  = mpatches.Patch(color=CORAL, label='Crítico')
ax1.legend(handles=[p_ok, p_at, p_cr], loc='lower right')

# --- Panel derecho: stock actual vs necesario ---
x = np.arange(len(df_alertas))
w = 0.35
ax2.bar(x-w/2, df_alertas['Stock actual'],    width=w, color=AZUL,  label='Stock
actual',    zorder=3)
ax2.bar(x+w/2, df_alertas['Stock necesario'], width=w, color=CORAL, label='Stock
necesario', zorder=3, alpha=0.8)
ax2.set_xticks(x)
ax2.set_xticklabels(prods_cortos, rotation=35, ha='right', fontsize=8)
ax2.set_ylabel('Unidades')
ax2.set_title('Stock actual vs. necesario (7 días)', fontweight='bold')
ax2.grid(axis='y', alpha=0.3, zorder=0)
ax2.legend()

plt.suptitle('Panel de alertas de stock — Restaurante El Mirador',
fontweight='bold', fontsize=12)
plt.tight_layout()
plt.savefig('fig_panel_alertas.png', dpi=150, bbox_inches='tight')
plt.show()
```

9. Generación automática de la orden de compra

```
pedido = df_alertas[df_alertas['Cantidad a pedir'] > 0].copy()

print(f'Productos que necesitan reposición: {len(pedido)} de {len(df_alertas)}')
print(f'Valor total del pedido: {pedido["Valor pedido (€)"].sum():,.2f} €\n')
```

```
# Agrupar por proveedor
for prov in pedido['Proveedor'].unique():
    sub = pedido[pedido['Proveedor']==prov][['Producto','Unidad','Cantidad a pedir',
                                             'Valor pedido (€)','Plazo entrega
(d)']]
    total_prov = sub['Valor pedido (€)'].sum()
    print(f'── {prov} ──────────────────────────────')
    print(sub.to_string(index=False))
    print(f'   Subtotal: {total_prov:,.2f} €')
    print()
# Resumen ejecutivo del pipeline
n_ok      = (df_alertas['Alerta']=='OK').sum()
n_aten    = (df_alertas['Alerta']=='Atención').sum()
n_crit    = (df_alertas['Alerta']=='Crítico').sum()
mape_med  = df_alertas['MAPE modelo (%)'].mean()
valor_ped = pedido['Valor pedido (€)'].sum()

print('=' * 50)
print('RESUMEN EJECUTIVO — ZIPO ANALYTICS PIPELINE')
print('=' * 50)
print(f'Fecha de análisis:    {fecha_hoy.date()}')
print(f'Horizonte previsión:  {HORIZONTE_DIAS} días')
print(f'Productos analizados: {len(df_alertas)}')
print(f'  ✅ Stock OK:          {n_ok}')
print(f'  ⚠️  Atención:         {n_aten}')
print(f'  🔴 Crítico:           {n_crit}')
print(f'Precisión media MAPE: {mape_med:.1f}%')
print(f'Valor pedido sugerido:{valor_ped:,.2f} €')
print('=' * 50)
```

Anexo III. Script de generación del dataset sintético (Python)

Se incluye el script de Python que genera el dataset sintético del Restaurante El Mirador descrito en el apartado 6.1. El script parte de un nivel de ventas base por producto y aplica el patrón semanal, la estacionalidad anual y una componente aleatoria, y exporta las tres hojas del conjunto de datos (ventas diarias, stock actual y parámetros).

```
"""
Generación del dataset sintético - Restaurante El Mirador
============================================================
Script que genera el conjunto de datos sintético utilizado en el TFG de Zipo.
Simula 180 días de operación (1 jul – 27 dic 2024) de un restaurante ficticio
con 11 productos repartidos en cinco categorías.

El procedimiento, para cada producto, parte de un nivel de ventas base y le
aplica tres efectos sucesivos:
  1. Patrón semanal  -> más ventas en fin de semana, mínimo los lunes.
  2. Estacionalidad   -> mayor demanda en verano, descenso hacia otoño.
  3. Ruido aleatorio  -> variación natural de las ventas diarias.

Genera tres hojas: ventas_diarias, stock_actual y parametros.
"""

import numpy as np
import pandas as pd
from datetime import date, timedelta

# Semilla para reproducibilidad
np.random.seed(42)
```

```
# ------------------------------------------------------------------------------
# 1. Definición de productos: nivel base diario y categoría
# ------------------------------------------------------------------------------
productos = {
    # producto            nivel_base  categoria
    'Solomillo (kg)':     (6.2,  'carne'),
    'Pollo (kg)':         (10.5, 'carne'),
    'Lubina (kg)':        (4.0,  'pescado'),
    'Salmón (kg)':        (3.3,  'pescado'),
    'Tomate (kg)':        (7.9,  'verdura'),
    'Lechuga (ud)':       (6.6,  'verdura'),
    'Patata (kg)':        (11.8, 'verdura'),
    'Nata (l)':           (4.7,  'lacteo'),
    'Queso (kg)':         (2.3,  'lacteo'),
    'Vino tinto (bot)':   (9.4,  'bebida'),
    'Cerveza (bot)':      (16.1, 'bebida'),
}

# ------------------------------------------------------------------------------
# 2. Patrón semanal (multiplicador por día: lunes=0 ... domingo=6)
#    Más ventas viernes y sábado, mínimo el lunes.
# ------------------------------------------------------------------------------
patron_semanal = {
    0: 0.62,   # lunes
    1: 0.85,   # martes
    2: 0.86,   # miércoles
    3: 0.86,   # jueves
    4: 1.36,   # viernes
    5: 1.36,   # sábado
    6: 1.10,   # domingo
}

# ------------------------------------------------------------------------------
# 3. Estacionalidad anual (multiplicador por mes)
#    Verano alto, descenso en otoño, ligera recuperación en diciembre.
# ------------------------------------------------------------------------------
estacionalidad = {
    7:  1.08,   # julio
    8:  1.20,   # agosto
    9:  1.02,   # septiembre
    10: 0.94,   # octubre
    11: 0.78,   # noviembre
    12: 1.00,   # diciembre
}

# ------------------------------------------------------------------------------
# 4. Generación de la serie de ventas diarias
# ------------------------------------------------------------------------------
fecha_inicio = date(2024, 7, 1)
n_dias = 180

filas = []
for prod, (base, categoria) in productos.items():
    for d in range(n_dias):
        fecha = fecha_inicio + timedelta(days=d)
        # Nivel base × patrón semanal × estacionalidad
        nivel = base * patron_semanal[fecha.weekday()] * estacionalidad[fecha.month]
        # Componente aleatoria (ruido gaussiano del 15%)
        ruido = np.random.normal(1.0, 0.15)
        unidades = max(0, nivel * ruido)
        filas.append({
            'fecha': pd.Timestamp(fecha),
            'producto': prod,
            'unidades_vendidas': round(unidades, 2),
            'unidad': prod.split('(')[1].rstrip(')'),
        })

ventas_diarias = pd.DataFrame(filas)
```

```
# ------------------------------------------------------------------------------
# 5. Hoja de stock actual (nivel de inventario en el momento del análisis)
# ------------------------------------------------------------------------------
stock_actual = pd.DataFrame([
    ('Solomillo (kg)',   12,   'kg',  28.0),
    ('Pollo (kg)',        8,   'kg',   5.5),
    ('Lubina (kg)',       6,   'kg',  18.0),
    ('Salmón (kg)',       8,   'kg',  14.0),
    ('Tomate (kg)',      20,   'kg',   1.8),
    ('Lechuga (ud)',     18,   'ud',   0.9),
    ('Patata (kg)',      30,   'kg',   0.7),
    ('Nata (l)',         10,   'l',    2.2),
    ('Queso (kg)',        7.5, 'kg',   9.5),
    ('Vino tinto (bot)', 30,   'bot',  6.0),
    ('Cerveza (bot)',   100,   'bot',  0.8),
], columns=['producto', 'stock_actual', 'unidad', 'coste_unitario_eur'])

# ------------------------------------------------------------------------------
# 6. Hoja de parámetros (proveedor, plazo de entrega, stock mínimo en días)
# ------------------------------------------------------------------------------
parametros = pd.DataFrame([
    ('Solomillo (kg)',   3, 'Cárnicas López',    2),
    ('Pollo (kg)',       3, 'Cárnicas López',    2),
    ('Lubina (kg)',      2, 'Pescados Mar',      1),
    ('Salmón (kg)',      2, 'Pescados Mar',      1),
    ('Tomate (kg)',      3, 'Verduras Huerta',   1),
    ('Lechuga (ud)',     3, 'Verduras Huerta',   1),
    ('Patata (kg)',      4, 'Verduras Huerta',   2),
    ('Nata (l)',         3, 'Lácteos Norte',     2),
    ('Queso (kg)',       4, 'Lácteos Norte',     2),
    ('Vino tinto (bot)', 5, 'Bodegas Rioja',     3),
    ('Cerveza (bot)',    5, 'Distribuidora Sur', 3),
], columns=['producto', 'stock_minimo_dias', 'proveedor', 'plazo_entrega_dias'])

# ------------------------------------------------------------------------------
# 7. Exportación a Excel (tres hojas)
# ------------------------------------------------------------------------------
with pd.ExcelWriter('Zipo_datos_restaurante.xlsx') as writer:
    ventas_diarias.to_excel(writer, sheet_name='ventas_diarias', index=False)
    stock_actual.to_excel(writer, sheet_name='stock_actual', index=False)
    parametros.to_excel(writer, sheet_name='parametros', index=False)

print(f'Dataset generado: {len(ventas_diarias)} registros de ventas '
      f'({ventas_diarias["producto"].nunique()} productos × {n_dias} días).')
```