Facultad de Ciencias Económicas y Empresariales ICADE

Grado en Relaciones Internacionales y Business Analytics

BAG2GO: SIMULACIÓN DE ESCENARIOS OPERATIVOS PARA UNA IDEA DE NEGOCIO LOGÍSTICA

Clave: 202101749

Autor: Fernando Núñez Yñiguez.

Tutor: Carlos Bellón Núñez-Mera.

MADRID | Julio 2026

Contenido

1. Introducción 6
1.1. Contexto y problema de estudio 6
1.2. Justificación del trabajo 6
1.3. Objetivos y pregunta de investigación 7
1.4. Alcance y metodología general 7
2. Marco conceptual y planteamiento del servicio 9
2.1. Marco teórico aplicado a servicios logísticos urbanos 9
2.2. Descripción del servicio propuesto 10
2.3. Propuesta de valor y lógica operativa del modelo 11
2.4. Madrid como caso de estudio. 11
3. Metodología y construcción del modelo de simulación 13
3.1. Enfoque metodológico general 13
3.2. Unidad de análisis: la reserva operativa 14
3.3. Construcción de la base espacial de alojamientos 15
3.4. Validación de alojamientos y rutas hacia el aeropuerto 16
3.5. Parámetros operativos del modelo 16
3.5.1 Parámetros generales de simulación 17
3.5.2 Parámetros de demanda 17
3.5.3 Parámetros temporales del servicio 17
3.5.4 Parámetros operativos y de capacidad 18
3.6. Generación de la demanda diaria simulada 19
3.7. Parametrización económica y tratamiento de costes 20
3.8. KPIs para la comparación entre escenarios 21
4. Análisis de escenarios operativos en un día de referencia 22
4.1. Planteamiento del día de referencia para la comparación básica 22
4.2. Escenario 0: servicio directo individual como referencia 23
4.3. Escenario 1: flota limitada con rutas consolidadas 23
4.4. Escenario 2: hubs urbanos de consolidación 24
4.5. Escenario 3: hubs urbanos con almacén central de desbordamiento 25
4.6. Síntesis comparativa del día de referencia 26
5. Simulación Monte Carlo y estabilidad de los resultados 26
5.1. Objetivo y funcionamiento de la simulación Monte Carlo 27
5.2. Comparación del servicio y la eficiencia operativa 28
5.3. Comparación económica de los escenarios 29

5.4. Interpretación global de los resultados Monte Carlo 31
6. Conclusiones 32
7. Anexo 35
7.1. Anexo 1: Tabla de trazabilidad del código 35
7.2 Anexo 2: Obtención de datos con Overpass turbo. 57
7.3 Anexo 3: análisis de sensibilidad de capacidad en el escenario 2 58
8. Bibliografía. 61

Por la presente, yo, Fernando Núñez Yñiguez, estudiante de Relaciones Internacionales y Business Analytics de la Universidad Pontificia Comillas al presentar mi Trabajo Fin de Grado titulado "Bag2go: simulación de escenarios operativos para una idea de negocio logística. ", 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. Referencias: Usado conjuntamente con otras herramientas, como Science, para identificar referencias preliminares que luego he contrastado y validado.

2. Metodólogo: Para descubrir métodos aplicables a problemas específicos de investigación.

3. Interpretador y generador de código: Para ayudarme a crear la lógica programada (Anexo 1)

4. Corrector de estilo literario y de lenguaje: Para mejorar la calidad lingüística y estilística del texto.

5. Sintetizador y divulgador de libros complicados: Para resumir y comprender literatura compleja.

6. Revisor: Para recibir sugerencias sobre cómo mejorar y perfeccionar el trabajo con diferentes niveles de exigencia.

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: Fernando Núñez Yñiguez.

Resumen: este Trabajo de Fin de Grado analiza la viabilidad operativa de un servicio urbano de recogida, custodia y entrega de equipaje en el aeropuerto de Madrid. A partir de una necesidad habitual del turismo urbano: gestionar las maletas entre el check-out y la salida del vuelo. Para ello, se construye un modelo de simulación para comparar distintas formas de organizar la operación.

El modelo utiliza alojamientos turísticos reales de Madrid, tiempos y distancias hacia el aeropuerto, demanda diaria simulada, restricciones de capacidad, ventanas horarias y costes operativos. Se comparan cuatro escenarios: servicio directo individual, flota limitada con rutas consolidadas, hubs urbanos y hubs con almacén central de respaldo. Finalmente se ejecuta una simulación Monte Carlo para comprobar que configuración operativa se adapta mejor a las condiciones determinadas.

Palabras clave: Logística urbana, simulación operativa, Monte Carlo, hubs urbanos, gestión de equipaje, eficiencia logística.

Abstract: this Bachelor's Thesis analyzes the operational feasibility of an urban luggage collection, storage, and delivery service to Madrid Airport. It addresses a common need in urban tourism: managing luggage between hotel check-out and flight departure. To this end, a simulation model is developed to compare different ways of organizing the operation.

The model uses real tourist accommodation data from Madrid, travel times and distances to the airport, simulated daily demand, capacity constraints, time windows, and operating costs. Four scenarios are compared: individual direct service, a limited fleet with consolidated routes, urban hubs, and urban hubs with a central backup warehouse. Finally, a Monte Carlo simulation is carried out to assess which operational scenario performs best under the defined conditions.

Keywords: Urban logistics, operational simulation, Monte Carlo simulation, urban hubs, luggage management, logistics efficiency.

1. Introducción

1.1. Contexto y problema de estudio

En los últimos años, el turismo urbano ha evolucionado hacia un modelo en el que el viajero no solo valora el destino, sino también la comodidad con la que puede gestionar cada momento del viaje. La digitalización de reservas, por ejemplo, en transportes y alojamientos ha elevado el nivel de exigencia del usuario, que cada vez espera soluciones más ágiles y adaptadas a sus necesidades reales. En ese contexto, siguen existiendo puntos de fricción que, aun siendo cotidianos, afectan de forma clara a la experiencia de todo viajero.

Uno de esos puntos aparece en el último día de estancia. En muchos casos, los viajeros deben abandonar su alojamiento a media mañana, mientras que su vuelo no sale hasta varias horas después. Esta diferencia entre la hora de check-out y la salida efectiva del viaje genera una situación incómoda: el viajero debe cargar con su equipaje durante buena parte del día, modificar sus planes o recurrir a soluciones alternativas que no siempre resultan prácticas. Aunque existen opciones como dejar temporalmente las maletas en el propio alojamiento o utilizar consignas, estas alternativas no eliminan del todo el problema, ya que suelen obligar al viajero a volver sobre sus pasos, condicionando su movilidad o introduciendo rigideces en su planificación.

A partir de esta necesidad surge la idea de un servicio de recogida, custodia y entrega de equipaje que permita al cliente desprenderse de sus maletas al finalizar su estancia y recuperarlas más tarde en el propio aeropuerto. La propuesta responde a un problema real y reconocible, pero su interés no se limita a la oportunidad comercial que representa. Desde una perspectiva analítica, se trata de un servicio cuya viabilidad depende en gran medida de cómo se diseñe su operativa.

1.2. Justificación del trabajo

El interés de este trabajo radica en abordar un problema desde una perspectiva distinta a la habitual. En lugar de centrarse en validar una idea de negocio o en desarrollar su encaje comercial, el análisis se orienta a entender si dicha idea puede sostenerse desde el punto de vista operativo. Esto implica desplazar el foco hacia las decisiones que determinan su funcionamiento real: cómo se organiza el servicio, cómo se asignan los recursos y cómo se gestionan las restricciones del entorno.

El caso resulta especialmente adecuado porque permite transformar una idea de servicio en un problema operativo medible. El análisis no se limita a describir la necesidad del cliente, sino que utiliza datos, supuestos operativos y simulación para comparar alternativas de diseño y evaluar su impacto en variables clave como el coste, el tiempo y el nivel de servicio.

Además, el trabajo se sitúa en un tipo de problemática cada vez más relevante: el diseño de servicios urbanos bajo demanda. Este tipo de servicios, caracterizados por su dependencia de la localización, el tiempo y la capacidad disponible, requieren soluciones que vayan más allá de la intuición y que integren análisis cuantitativo en su diseño.

Por último, la elección de un único entorno de análisis permite centrar el trabajo y evitar dispersión. Esto facilita construir un modelo más consistente y extraer conclusiones más claras, manteniendo un equilibrio entre realismo y manejabilidad.

1.3. Objetivos y pregunta de investigación

Se busca estudiar qué configuración operativa resulta más adecuada para articular el servicio bajo unas condiciones realistas de funcionamiento, teniendo en cuenta variables que se analizarán más adelante.

El trabajo se apoya en los siguientes objetivos específicos. En primer lugar, definir de forma clara la lógica operativa del servicio y el flujo asociado a cada reserva. En segundo lugar, identificar las variables, restricciones y recursos que condicionan su funcionamiento. En tercer lugar, construir una base de análisis útil para representar el caso y comparar distintas alternativas de diseño. Por último, evaluar dichas alternativas a partir de criterios operativos concretos que permitan extraer una recomendación fundamentada. En coherencia con todo ello, surge la siguiente pregunta de investigación que servirá como guía del trabajo:

¿Qué configuración operativa permite prestar de forma más eficiente un servicio de recogida, custodia y entrega de equipaje en el aeropuerto, equilibrando coste, tiempo y nivel de servicio?

1.4. Alcance y metodología general

El estudio toma como unidad principal de análisis la reserva, entendida como la solicitud de recogida, custodia y posterior entrega del equipaje asociada a un cliente. A partir de esa unidad, se analizan las decisiones que condicionan el funcionamiento del servicio y

la forma en que distintas configuraciones operativas pueden responder a una misma demanda.

El alcance del TFG se limita a la dimensión operacional del modelo. Por tanto, no se pretende desarrollar un plan financiero completo ni profundizar en la validación comercial del negocio, ya que estos aspectos forman parte de los trabajos desarrollados por Manuel Recio Rubiato y Jorge Santander Ramos. No obstante, sus análisis servirán como referencia para contextualizar y justificar algunos de los supuestos utilizados en este trabajo.

En términos metodológicos, el trabajo combina una primera fase de definición del servicio, apoyada en el Business Model Canvas desarrollado en el marco conceptual, con una fase posterior de análisis aplicada. El Business Model Canvas (BMC) permite ordenar los elementos básicos de la propuesta, como la propuesta de valor, los clientes, los recursos necesarios y la lógica general de funcionamiento. Pero, no se busca validar este negocio con el BMC, sino tomar esa definición inicial como punto de partida para construir un modelo operativo.

Para ello, y como se ha mencionado previamente, se construye un modelo de simulación que permite representar la operación diaria del servicio, generar distintos escenarios de demanda y comparar varias configuraciones operativas mediante indicadores homogéneos de coste, tiempo, capacidad y nivel de servicio. De esta manera, la metodología adoptada responde a una lógica aplicada: utilizar la analítica para apoyar una decisión operativa concreta y dar respuesta a la pregunta de investigación planteada.

2. Marco conceptual y planteamiento del servicio

2.1. Marco teórico aplicado a servicios logísticos urbanos

En el ámbito logístico, el Business Analytics permite analizar decisiones relacionadas con la asignación de recursos, el diseño de redes, la capacidad disponible, los tiempos de servicio y la priorización de alternativas. En este sentido, la literatura reciente destaca que la analítica aplicada a supply chain management ha evolucionado desde usos más descriptivos hacia enfoques cada vez más orientados a la optimización y al soporte a decisiones complejas (Alsolbi et al., 2023).

Cuando este enfoque se traslada al entorno urbano, la complejidad se dispara. Los servicios logísticos urbanos operan en contextos marcados por la congestión, la variabilidad temporal, la presión sobre el espacio disponible y la necesidad de mantener niveles de servicio altos, bajo recursos limitados. Por ello, la literatura sobre smart urban logistics subraya que el reto no consiste solo en mover bienes o gestionar flujos, sino en integrar tecnología, datos y modelos de decisión para responder de forma más eficiente a las condiciones cambiantes de la ciudad (Büyüközkan & Ilıcak, 2022).

Dentro de estas decisiones, una de las más relevantes es la relacionada con la localización de infraestructuras o puntos de apoyo al servicio. La literatura clásica de logística y transporte ha mostrado que los problemas de localización ocupan un papel central en el diseño de redes, ya que determinan cómo se conectan los puntos de demanda con los recursos disponibles y condicionan de forma directa los costes, los tiempos y la flexibilidad del sistema. Además, estos problemas han ganado relevancia en los últimos años al combinarse con incertidumbre, sostenibilidad y decisiones basadas en datos, reforzándose su interés en servicios urbanos de nueva generación (Saldanha-da-Gama, 2022).

Otro aspecto especialmente relevante en logística urbana es el uso de estructuras intermedias de consolidación. La literatura reciente distingue entre urban consolidation centres, normalmente situados en zonas periféricas y con mayor capacidad, y micro-consolidation centres, ubicados más cerca de la zona de servicio y pensados para coberturas más localizadas. Ambos modelos responden a la necesidad de reorganizar flujos para reducir ineficiencias y adaptar la operación al entorno urbano, aunque su conveniencia depende de las características del servicio, la ciudad y la demanda. Distintos

trabajos muestran que la comparación entre configuraciones centralizadas, descentralizadas o multinivel requiere análisis específicos apoyados en criterios cuantitativos y operativos (Kovač et al., 2023; Gillström & Björklund, 2024).

La planificación operativa es otro bloque fundamental dentro del Business Analytics aplicado a logística urbana. En este plano, la simulación permite representar la llegada de solicitudes, la disponibilidad de recursos, las restricciones temporales y la asignación de tareas. La combinación de simulación y optimización ha demostrado ser especialmente valiosa en servicios urbanos bajo demanda, ya que permite modelizar el comportamiento del sistema y comparar decisiones alternativas. Un ejemplo reciente se observa en marcos de simulación-optimización aplicados a entregas instantáneas, donde el uso combinado de ambas técnicas mejora la capacidad de respuesta del sistema y reduce los tiempos totales de servicio (Jorge et al., 2024). Aunque el caso de este TFG no replica exactamente ese contexto, sí comparte su lógica de fondo: evaluar decisiones operativas bajo restricciones de tiempo, capacidad y coordinación de recursos.

2.2. Descripción del servicio propuesto

El servicio propuesto consiste en la recogida, custodia temporal y posterior entrega de equipaje para viajeros que necesitan gestionar sus maletas entre el final de su estancia y la salida de su vuelo. A efectos del modelo, el servicio se representa como una cadena operativa en la que cada reserva activa un flujo concreto desde un punto de origen hasta la entrega final en el aeropuerto.

Desde un punto de vista funcional, el servicio se articula a partir de una reserva previa realizada por el cliente. En dicha reserva se recogen los datos necesarios para ejecutar la operación, como el punto de recogida, la franja horaria prevista, el número de bultos y la referencia temporal asociada a la entrega final. Esta referencia puede apoyarse en la información del vuelo, ya que permite calcular el momento en el que el equipaje debe estar disponible en el aeropuerto. A partir de estos datos, el sistema organiza la recogida, custodia y entrega del equipaje conforme a la configuración operativa adoptada.

De forma general, el servicio puede dividirse en tres grandes fases. La primera corresponde a la recogida o recepción del equipaje, que se produce una vez el cliente abandona su alojamiento o deposita sus maletas en el punto establecido. La segunda comprende la custodia temporal y la gestión interna del equipaje, que puede implicar o no el paso por puntos intermedios en función del diseño operativo. La tercera fase consiste

en la entrega final en el aeropuerto, donde el cliente recupera su equipaje antes de la salida del vuelo.

2.3. Propuesta de valor y lógica operativa del modelo

La propuesta de valor del servicio se apoya en liberar al viajero de la gestión de su equipaje durante las horas comprendidas entre el final de la estancia y la salida del vuelo. El valor principal no reside únicamente en trasladar una maleta de un punto a otro, sino en ofrecer una solución que combine custodia, coordinación logística y entrega final en el aeropuerto.

Desde la perspectiva del usuario, esta propuesta puede resumirse en tres elementos principales: comodidad, seguridad y flexibilidad. La comodidad se relaciona con evitar que el cliente tenga que cargar con el equipaje o reorganizar su día en función de él. La seguridad implica que el servicio asume la custodia temporal de las maletas dentro de un sistema controlado. La flexibilidad se vincula con la capacidad de adaptar la operación a los horarios del viajero y a la entrega final antes del vuelo.

La lógica operativa del modelo surge precisamente de la necesidad de hacer viable esa propuesta de valor. Si el servicio promete esas tres características la operación debe estar diseñada para sostener esas tres dimensiones. La comodidad exige que la recogida o recepción del equipaje y su posterior entrega encajen con el itinerario del usuario. La seguridad requiere trazabilidad, control de custodia y protocolos claros sobre el equipaje. La flexibilidad exige una estructura capaz de absorber variaciones temporales y organizar el flujo de reservas sin deteriorar el nivel de servicio.

2.4. Madrid como caso de estudio.

La elección de Madrid como caso de estudio responde a la necesidad de situar el análisis en un entorno urbano con una intensidad turística elevada, una red consolidada de alojamientos y una infraestructura aeroportuaria de gran capacidad. Estas características hacen de Madrid un contexto especialmente adecuado para estudiar un servicio de gestión de equipaje cuya operativa depende, en gran medida, de la concentración espacial de la demanda, de los desplazamientos dentro de la ciudad y de la conexión con un nodo final claramente definido.

Desde el punto de vista turístico, Madrid destaca por su relevancia dentro del sistema turístico español. Los datos del Instituto Nacional de Estadística sitúan a Madrid entre los

puntos turísticos con mayor número de pernoctaciones hoteleras del país, lo que refleja tanto el volumen de visitantes que recibe como el peso de la estancia urbana en su actividad turística. A ello se suma la presencia de una oferta amplia y diversa de alojamientos, tanto hoteleros como de otro tipo, que refuerza la idoneidad de la ciudad como entorno para analizar servicios vinculados al tránsito de viajeros y a la gestión de equipaje (Instituto Nacional de Estadística [INE], 2026).

La elección de Madrid también se justifica por la relevancia del aeropuerto Adolfo Suárez Madrid-Barajas dentro de la red de transporte española y europea. Según Aena, el aeropuerto registró 68,2 millones de pasajeros en 2025 y se mantuvo como el principal aeropuerto español por tráfico de pasajeros, carga y número de operaciones (Aena, 2026).

Por tanto, Madrid no se selecciona únicamente por su atractivo turístico, sino por reunir las condiciones necesarias para estudiar el problema en un marco operativo creíble: una demanda potencial significativa, una estructura de alojamientos amplia y una infraestructura aeroportuaria con capacidad suficiente para actuar como punto central de destino.

3. Metodología y construcción del modelo de simulación

3.1. Enfoque metodológico general

La metodología de este trabajo se basa en la construcción de un modelo cuantitativo de simulación operativa. El objetivo principal no es reproducir de forma exacta una operación real ya implantada, sino representar de manera estructurada el funcionamiento potencial de un servicio y comparar distintas configuraciones operativas. Todo el código que apoya el desarrollo de esta simulación se encuentra en el Anexo 1.

Esta decisión viene de la propia naturaleza del problema analizado. El servicio estudiado todavía no dispone de datos históricos reales de operación, ya que parte de una idea de negocio en fase de diseño. Por ello, no resulta posible construir un modelo predictivo basado en observaciones pasadas. En su lugar, se plantea una simulación que permite transformar una propuesta conceptual en un sistema operativo evaluable, incorporando localizaciones reales de alojamientos, tiempos de desplazamiento, demanda simulada, restricciones de capacidad, reglas de asignación y costes operativos. Se puede ver el flujo en el siguiente esquema.



Figura 1. Flujo general del modelo de simulación operativa utilizado para comparar los escenarios de servicio. Fuente: elaboración propia.

El modelo se centra en la operación diaria del servicio. Cada día simulado está formado por un conjunto de reservas, y cada reserva queda definida por una ubicación de origen,

un número de maletas, una hora de recogida, una hora de vuelo y una hora objetivo de llegada al aeropuerto del equipaje. A partir de estos elementos, la simulación evalúa si cada reserva puede ser atendida en función de la configuración operativa planteada en cada escenario.

La lógica del análisis es comparativa. Todos los escenarios se ejecutan sobre la misma demanda diaria simulada, lo que permite atribuir las diferencias de comportamiento al diseño operativo de cada configuración y no a variaciones en la demanda. De este modo, el modelo permite analizar cómo cambia el desempeño del servicio cuando se modifica la forma de organizar la operación: atención directa individual, flota limitada con rutas consolidadas, hubs urbanos de consolidación o hubs con apoyo de un almacén central.

La comparación entre escenarios se realiza mediante indicadores comunes de desempeño. Entre ellos se incluyen el número de reservas atendidas y rechazadas, el nivel de servicio, el número de viajes al aeropuerto, la ocupación media de las furgonetas, el coste operativo total y el coste medio por reserva. Esta normalización de indicadores es necesaria porque los escenarios tienen lógicas operativas distintas, pero deben poder evaluarse bajo una misma estructura analítica.

Además, el modelo incorpora una simulación Monte Carlo con 365 iteraciones, donde cada iteración representa un posible día de año natural. En cada una de ellas cambian la composición de la demanda, la asignación de reservas a alojamientos, el número de maletas y los horarios de recogida y vuelo. Esta ampliación permite evitar que la evaluación dependa de un único día simulado y facilita el análisis de la robustez de cada escenario ante variaciones razonables de la demanda.

3.2. Unidad de análisis: la reserva operativa

Cada reserva se define como un registro operativo que combina tres tipos de información: espacial, temporal y de capacidad. En primer lugar, la reserva se asocia a un alojamiento turístico de Madrid, que actúa como punto de origen y permite calcular distancias, tiempos de desplazamiento y, para los escenarios con puntos de recogida, permite la asignación al más cercano.

En segundo lugar, cada reserva incorpora un número de maletas, ya que la carga operativa del servicio no depende solo del número de clientes, sino del volumen físico que debe transportarse.

Por último, cada reserva incluye una hora de recogida, una hora de vuelo y una hora objetivo de llegada al aeropuerto. Esta última se calcula para asegurar que el equipaje esté disponible con margen suficiente antes de la salida del avión. Durante la simulación, la reserva puede quedar atendida, rechazada o, en el escenario con almacén central, derivada como desbordamiento, permitiendo comparar cómo distintas configuraciones operativas gestionan una misma demanda.

3.3. Construcción de la base espacial de alojamientos

La base espacial del modelo se construye a partir de un archivo GeoJSON con alojamientos turísticos de Madrid, extraído previamente de OpenStreetMap mediante Overpass Turbo. Estos alojamientos representan los posibles puntos de origen de las reservas simuladas, por lo que era necesario transformar la información geográfica en una tabla utilizable para el análisis.

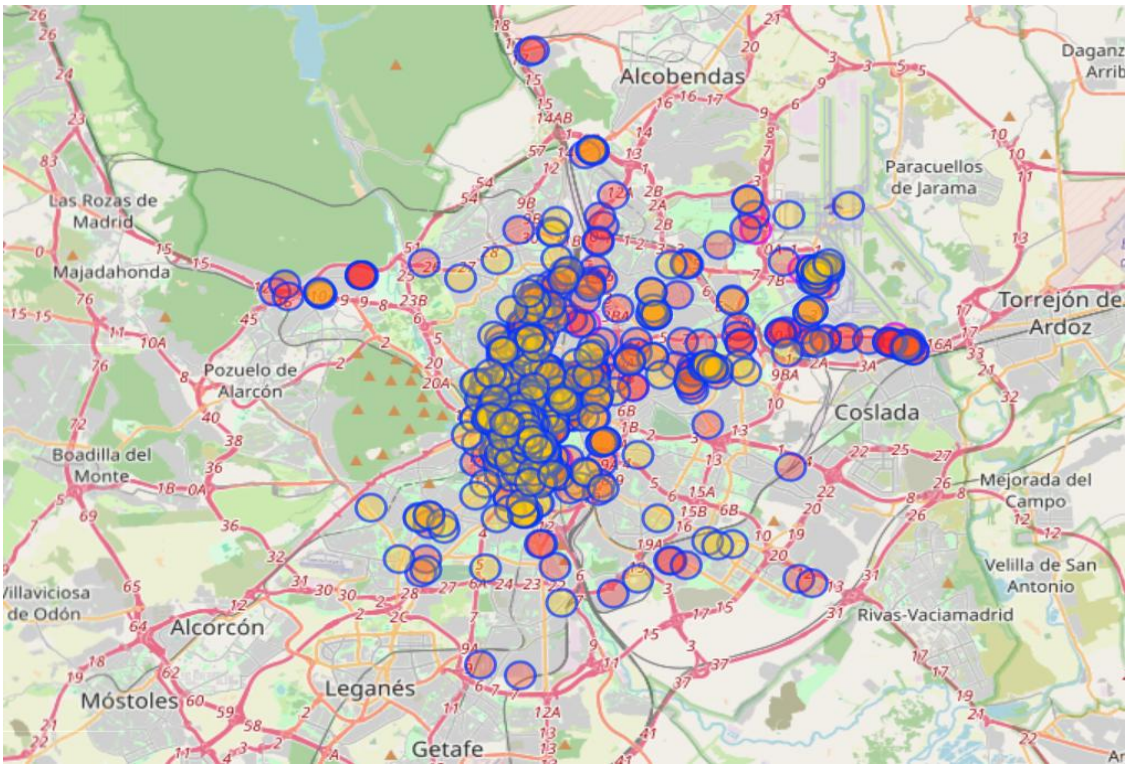


Figura 2. Distribución espacial de alojamientos turísticos identificados en Madrid mediante Overpass Turbo. Fuente: extraída de OpenStreetMap mediante Overpass Turbo.

Para ello, se conservan únicamente los registros con geometría de tipo punto, ya que el modelo necesita coordenadas concretas de latitud y longitud para calcular distancias, tiempos de desplazamiento y asignaciones a puntos de recogida. Para cada alojamiento

se extraen los principales campos disponibles, como el identificador, el nombre, la tipología, las coordenadas y, cuando existe, información básica de dirección.

Después se aplica una limpieza básica de la base. Se eliminan registros sin coordenadas válidas y duplicados identificados a partir del código de OpenStreetMap (Anexo 2). Además, se crea un identificador interno para cada alojamiento y, cuando el registro no dispone de nombre, se le genera uno para mantener la trazabilidad en el modelo. Tras este proceso, la base queda formada por 1.048 alojamientos válidos. Es importante recalcar que estos puntos no representan una demanda real, sino una base espacial sobre la que se distribuyen aleatoriamente las reservas simuladas. (Anexo 1).

3.4. Validación de alojamientos y rutas hacia el aeropuerto

Una vez obtenidos estos puntos, se valida su ruta hacia el aeropuerto Adolfo Suárez Madrid-Barajas, que se define como punto fijo de entrega final. A partir de sus coordenadas, se calcula para cada alojamiento tanto la distancia directa al aeropuerto como la ruta estimada por carretera.

Estas rutas se pueden obtener con OpenRouteService, que permite estimar el recorrido entre cada alojamiento y el aeropuerto. Los resultados se almacenan en una tabla intermedia, de manera que las rutas ya calculadas puedan reutilizarse sin necesidad de repetir todas las consultas a través de la API. (Anexo 1).

Después del cálculo de rutas, solo se conservan los alojamientos con tiempo y distancia al aeropuerto coherentes. Se eliminan, por tanto, aquellos casos en los que la ruta no haya podido calcularse correctamente. Tras el proceso, se conservan las 1.048 filas del conjunto de datos original.

Además, se calcula el ratio entre la distancia por carretera y la distancia directa, que nos permite obtener un factor de corrección que se explicará más adelante. Este indicador sirve como comprobación de coherencia, ya que permite revisar si las rutas estimadas presentan desviaciones razonables respecto a la distancia en línea recta. También se inspeccionan los alojamientos con mayores tiempos de desplazamiento para detectar posibles outliers. (Anexo 1).

3.5. Parámetros operativos del modelo

El modelo se apoya en una serie de parámetros que permiten convertir la demanda simulada en una operación concreta. Estos parámetros fijan las condiciones bajo las que

se evalúan los escenarios y permiten que todos ellos sean comparables. Los principales parámetros operativos divididos por bloques son los siguientes:

3.5.1 Parámetros generales de simulación

- Fecha de operación simulada: se fija el día 1 de enero de 2026 como fecha base del modelo. La elección de una fecha concreta permite trabajar con variables temporales en formato homogéneo.
- Semilla aleatoria: se utiliza una semilla global igual a 42. Sirve para que sea reproducible el código.
- Número de iteraciones Monte Carlo: se establecen 365 iteraciones. Esta configuración permite analizar la estabilidad de los escenarios a lo largo de un año simulado y no depender de un único día generado aleatoriamente.

3.5.2 Parámetros de demanda

- Demanda media diaria: se fija en 200 reservas. Esta cifra procede de redondear una estimación mensual de 5.350 clientes, equivalente a unas 178 reservas diarias. El redondeo a 200 introduce un margen de exigencia operativa y permite probar el comportamiento del sistema ante una demanda ligeramente superior a la estimación base. Esos 5350 clientes mensuales salen del SOM calculado por Manuel Recio Rubiato para el plan financiero (2026).
- Límites de demanda diaria: el número de reservas simuladas se acota entre 50 y 400.
- Distribución de maletas por reserva: se considera que una reserva puede incluir una, dos o tres maletas, con probabilidades del 60%, 30% y 10%, respectivamente. Esta hipótesis parte de la idea de que la reserva se realiza principalmente por persona. Permite traducir reservas a cantidad de maletas a gestionar.

3.5.3 Parámetros temporales del servicio

- Horario de recogida: las recogidas se generan entre las 9:30 y las 13:00, con una moda a las 11:00. Esta ventana responde a la lógica del servicio, ya que el cliente suele abandonar el alojamiento por la mañana y necesita dejar su equipaje antes de continuar con su día. Se establece este horario como la media según la guía de

estudio para el diploma de Hotel Management de la Confederation of Tourism and Hospitality (2011, pp. 94-95).

- Horario de vuelo: los vuelos se generan entre las 15:00 y las 23:30, con una moda a las 19:00. Esta hipótesis se apoya en una observación manual de horarios de salida del aeropuerto de Madrid-Barajas, donde se identifica una concentración relevante de vuelos entre las 18:00 y las 20:00. Para ello se ha utilizado la herramienta de Infovuelos de Aena (Aena, 2026)
- Margen de entrega en aeropuerto: el equipaje debe estar disponible en el aeropuerto 180 minutos antes de la salida del vuelo. Este margen se incorpora como compromiso de seguridad del servicio y busca reducir el riesgo operativo para el cliente.

3.5.4 Parámetros operativos y de capacidad

- Tiempo de retorno a ciudad: se considera un tiempo de retorno operativo de 25 minutos. Este parámetro permite reflejar que, después de una entrega, la furgoneta necesita reposicionarse para poder seguir operando.
- Capacidad de la furgoneta: se fija en 37 maletas. Este valor se estima a partir de la capacidad de carga de una Mercedes Vito Long, aproximada en 850 kg, dividida entre un peso medio de 20 kg por maleta y aplicando un margen de holgura del 10 %. La capacidad de la furgoneta actúa como una de las principales restricciones en los escenarios con consolidación. No se considera el volumen, ya que no es el limitante de capacidad en este caso.
- Tiempo de parada en recogida: se estima en 5 minutos por parada. Este valor representa el tiempo necesario para que la furgoneta se detenga, se identifique la reserva y se cargue el equipaje.
- Tiempo de descarga en aeropuerto: se estima en 6 minutos por viaje. Este tiempo aproxima la descarga de una furgoneta con varias maletas.
- Tiempo de manejo por maleta: se fija en 1,5 minutos. Este parámetro recoge el tiempo asociado a manipular, mover o clasificar cada bulto dentro de la operación.

- Tiempo de gestión en hub: se fija en 3 minutos por maleta. En los escenarios con hubs, este tiempo representa la recepción, identificación y ordenación del equipaje antes de su salida consolidada hacia el aeropuerto.
- Factores de retorno: se incorporan factores distintos según el tipo de operación posterior al viaje. El retorno directo se pondera con un factor de 0,75, mientras que el retorno a hub o almacén se mantiene en 1,00. Estos factores permiten ajustar el coste operativo del reposicionamiento según la lógica de cada escenario.

(Anexo 1).

3.6. Generación de la demanda diaria simulada

A partir de los parámetros definidos en el apartado anterior, el modelo genera una demanda diaria simulada que sirve como entrada común para todos los escenarios. Al no disponer de datos históricos reales del servicio, esta demanda no busca reproducir un día concreto de operación, sino construir un conjunto de reservas plausible y suficientemente estructurado para evaluar distintas configuraciones logísticas bajo las mismas condiciones.

El proceso comienza generando el número total de reservas del día mediante una distribución de Poisson, tomando como media la demanda diaria definida. Esta elección permite que el volumen de solicitudes varíe de un día a otro alrededor de un valor esperado. Una vez obtenido el número de reservas, cada una se asigna aleatoriamente a uno de los 1.048 alojamientos válidos de la base espacial. De esta forma, la demanda queda distribuida sobre localizaciones reales de Madrid, aunque sin ponderar unos alojamientos por encima de otros al no disponer de datos geográficos del uso del servicio.

Después, cada reserva recibe un número de maletas y dos referencias temporales: la hora de recogida y la hora de vuelo. Las maletas se generan a partir de la distribución definida previamente, mientras que las horas se obtienen mediante distribuciones triangulares dentro de las ventanas horarias establecidas. Este tipo de distribución permite incorporar una hora más probable sin asumir que todos los momentos de la ventana tienen la misma probabilidad.

Con estos datos, el modelo calcula la hora objetivo de llegada al aeropuerto, restando a la hora de vuelo el margen de entrega establecido. A continuación, se comprueba que la diferencia entre la hora de recogida y la hora objetivo sea suficiente para operar la reserva.

Por último, las horas generadas se redondean a intervalos de diez minutos para facilitar la agrupación de reservas en las rutas. El resultado es una tabla diaria en la que cada registro contiene el alojamiento de origen, el número de maletas, la hora de recogida, la hora de vuelo, la hora objetivo de llegada al aeropuerto, el margen operativo disponible y los tiempos y distancias asociados al alojamiento. (Anexo 1).

3.7. Parametrización económica y tratamiento de costes

Además de los parámetros operativos, el modelo incorpora una serie de parámetros económicos para estimar el coste asociado a cada escenario. El objetivo de estos parámetros es disponer de una métrica que permita comparar la eficiencia operativa y la necesidad de recursos de las distintas configuraciones.

El modelo se centra en los costes directamente relacionados con la operación diaria: desplazamiento, manipulación de equipaje, tiempos de parada, gestión en puntos de recogida y disponibilidad de flota. Los principales parámetros económicos utilizados son los siguientes y se derivan de la estructura de costes del análisis financiero elaborado por Manuel Recio Rubiato:

- Coste de combustible por kilómetro: se calcula a partir de un consumo estimado de 6,7 litros cada 100 kilómetros y un precio de combustible de 1,80 euros por litro. Con estos valores, el coste por KM es de 0,12 euros por kilómetro. Este parámetro permite imputar el coste variable asociado a los desplazamientos realizados por las furgonetas.
- Coste operativo por hora: se deriva de la estructura salarial prevista en el plan financiero. Se parte de 210.000 euros anuales en sueldos, 6 trabajadores y 1.920 horas anuales por trabajador, calculadas como 40 horas semanales durante 48 semanas. Con estos supuestos, el coste operativo resultante es de aproximadamente 18,23 euros por hora.
- Coste operativo por minuto: a partir del coste horario anterior, se obtiene un coste operativo aproximado de 0,304 euros por minuto. Este valor permite transformar tiempos de operación en costes económicos comparables.
- Coste de manejo por maleta: se calcula multiplicando el tiempo de manejo por maleta, fijado en 1,5 minutos, por el coste operativo por minuto. El resultado es un coste aproximado de 0,46 euros por maleta.

- Coste de parada en recogida: se obtiene multiplicando los 5 minutos estimados por parada por el coste operativo por minuto. El coste resultante es de 1,52 euros por parada. Este parámetro recoge el tiempo necesario para detenerse, identificar la reserva y cargar el equipaje.
- Coste de descarga en aeropuerto: se calcula a partir de un tiempo de descarga de 6 minutos por viaje. Aplicando el coste operativo por minuto, el coste de descarga se sitúa en torno a 1,82 euros por viaje. Este coste se imputa cada vez que una furgoneta realiza una entrega en el aeropuerto.
- Coste de gestión en hub por maleta: en los escenarios con hubs, cada maleta requiere un tiempo adicional de recepción, identificación y ordenación. Este tiempo se fija en 3 minutos por maleta. Al multiplicarlo por el coste operativo por minuto, se obtiene un coste aproximado de 0,91 euros por maleta.
- Coste de disponibilidad de vehículo: el alquiler anual de cada vehículo se estima en 12.000 euros. Considerando 360 días comerciales, el coste diario de disponibilidad por vehículo es de 33,33 euros. Este coste no depende del número de viajes realizados, sino de tener la furgoneta disponible para operar durante el día.

A partir de estos parámetros, el modelo calcula dos grandes tipos de coste. Por un lado, el coste variable operativo, que depende de los kilómetros recorridos, los tiempos de servicio, las paradas, la manipulación de maletas y la gestión en hubs. Por otro lado, el coste de disponibilidad de flota, que representa el coste diario de contar con los vehículos necesarios para operar cada escenario. La suma de ambas partidas da lugar al coste total simulado. (Anexo 1).

3.8. KPIs para la comparación entre escenarios

Para comparar los escenarios de forma homogénea, todos los resultados se transforman en una misma estructura de KPIs, que nos permite comparar entre escenarios, a pesar de ser bastante diferentes entre sí. Estas métricas comparativas son las siguientes:

- Reservas totales: número total de reservas generadas en un día.
- Reservas atendidas: número de reservas que el escenario consigue gestionar correctamente. Se mide así la capacidad del escenario de absorber la demanda.
- Reservas rechazadas: número de reservas que no pueden ser atendidas.

- Reservas desbordadas al almacén: número de reservas que, en el escenario con almacén, se derivan desde los hubs por falta de capacidad disponible. Permite medir el uso del almacén como mecanismo de respaldo.
- Tasa de atención: porcentaje de reservas atendidas sobre el total de reservas generadas.
- Nivel de servicio total: porcentaje de reservas entregadas dentro del margen establecido.
- Nivel de servicio sobre reservas atendidas: porcentaje de reservas atendidas que cumplen el margen de entrega. Sirve para diferenciar entre no atender una reserva y atenderla fuera de plazo.
- Maletas totales: número total de maletas.
- Viajes al aeropuerto: número de desplazamientos realizados hasta el aeropuerto.
- Ocupación media: porcentaje medio de uso de la capacidad de las furgonetas.
- Número de vehículos operativos: número de vehículos necesarios o disponibles para ejecutar el escenario.
- Coste variable total: coste asociado directamente a la operación.
- Coste de disponibilidad de flota: coste fijo asociado al número de vehículos necesarios.
- Coste total: suma del coste variable total y del coste de disponibilidad de flota.
- Coste medio por reserva atendida: coste total dividido entre las reservas que se finalizan.
- Coste medio por reserva total: coste total dividido entre todas las reservas generadas.

(Anexo 1).

4. Análisis de escenarios operativos en un día de referencia

4.1. Planteamiento del día de referencia para la comparación básica

Para comparar los escenarios se utiliza una misma demanda diaria simulada. En el día de referencia, se han generado 212 reservas y 317 maletas, que se emplean como entrada común para todas las configuraciones analizadas.

Todos los escenarios reciben, por tanto, las mismas reservas, con los mismos alojamientos de origen, horarios de recogida, horarios de vuelo, maletas y tiempos de desplazamiento

asociados. Permite comparar los resultados bajo las mismas condiciones y observar las diferencias entre escenarios.

4.2. Escenario 0: servicio directo individual como referencia

El escenario 0 representa la forma más simple de operar el servicio. Cada reserva se atiende de manera individual desde el alojamiento hasta el aeropuerto. Por ello, este escenario funciona como referencia basal para valorar el efecto de las estrategias de consolidación planteadas en los escenarios posteriores.

Al no aplicar restricciones de agrupación ni de capacidad, el escenario atiende toda la demanda del día y no genera rechazos. También mantiene un nivel de servicio del 100 %, ya que todas las entregas se realizan dentro del margen previsto.

El principal inconveniente aparece en la intensidad operativa. Al realizar un viaje individual por reserva, el número de desplazamientos al aeropuerto es muy elevado y la ocupación media de las furgonetas se mantiene en solo un 4,04 %.

Desde el punto de vista económico, el coste total asciende a 5.539,57 euros, con un coste operativo medio de 26,13 euros por reserva atendida. El escenario directo garantiza el servicio, pero lo hace con una estructura muy poco eficiente. (Anexo 1).

4.3. Escenario 1: flota limitada con rutas consolidadas

El escenario 1 introduce una operación con flota limitada y rutas consolidadas. En este caso, las reservas pueden agruparse en una misma ruta siempre que sean compatibles en términos de capacidad, horario de recogida y margen de llegada al aeropuerto. La flota se fija en tres vehículos para hacer la comparación más equilibrada con el escenario 2, donde también se dispone de tres furgonetas, una asociada a cada hub urbano.

Para este escenario, se añaden una serie de supuestos operativos razonables que permiten una construcción sólida de este escenario. Estos parámetros permiten calcular distancias y tiempos entre dos puntos de la ruta, que por una limitación de la API de ORS no podemos hacerlo para nuestras rutas. Estos son:

- Máximas paradas por ruta: se establece un número máximo de paradas para una ruta en 6. Más paradas podrían comprometer la llegada a tiempo al aeropuerto.
- Ventana de recogida: permite agrupar reservas hechas en una franja de 45 minutos.

- Velocidad media urbana: se fija en 24 km/h, tomando como referencia los valores observados en una visualización de velocidades medias por tramo en Madrid basada en datos del ayuntamiento (Antizar, 2020).
- Factor de corrección de Haversine: ayuda al cálculo de distancias y se fija en 1,45. Se calcula como la media del ratio de carretera de la tabla mencionada en el apartado 3.4.
- Tiempo medio de retorno: 22 minutos y es calculada como la media de tiempo mínimo observada en esa misma tabla.

A diferencia del escenario directo, el modelo puede rechazar reservas cuando no es posible incorporarlas a una ruta viable. En el día de referencia, este escenario atiende 59 reservas y rechaza 153. Estos rechazos no representan un fallo del modelo, sino la consecuencia de trabajar con una limitación de vehículos de transporte y con restricciones temporales exigentes.

El nivel de servicio sobre las reservas atendidas se mantiene en el 100 %, lo que indica que las reservas aceptadas llegan dentro del margen previsto. Sin embargo, el nivel de servicio total se reduce al 27,83 %, ya que una parte importante de la demanda no puede ser absorbida por la configuración operativa. La agrupación permite reducir los desplazamientos al aeropuerto hasta 27 viajes. No obstante, la ocupación media de las furgonetas sigue siendo baja, con un 8,81 %.

Desde el punto de vista económico, el coste total asciende a 383,19 euros, con un coste medio de 6,49 euros por reserva atendida. Aunque el coste total no es elevado, debe interpretarse junto con la baja tasa de atención. (Anexo 1).

4.4. Escenario 2: hubs urbanos de consolidación

El escenario 2 modifica la lógica operativa del servicio al introducir hubs urbanos como puntos de consolidación. En este caso, el cliente deposita la maleta en el hub asignado y la empresa se encarga únicamente del transporte hasta el aeropuerto.

Para este caso, se han creado tres posibles centros de recogida, uno en el centro de Madrid, otro en el barrio de Salamanca y un último cerca de Atocha, con una restricción de capacidad de 60, 40 y 40 respectivamente. Esta limitación de la capacidad se debe a la búsqueda de locales pequeños que no supongan un alto alquiler (Aunque no se esté imputando como un coste en este escenario). Además, se ha observado que estas

capacidades son suficientes con la demanda contemplada (análisis de sensibilidad que lo demuestra en Anexo 3). Finalmente, se añaden dos parámetros adicionales: tiempo operativo de llegada al hub, situado en 5 minutos, para la recepción y clasificación del equipaje, y el umbral de salida, limitado en 0,7 y que pretende que el hub no se sature.

En el código, cada reserva se asigna a uno de los centros de recogida definidos y se agrupan las salidas hacia el aeropuerto en función de la capacidad disponible y de los horarios objetivo de entrega. La flota se fija en tres vehículos, uno asociado a cada hub.

Este escenario atiende toda la demanda y no genera rechazos. También mantiene un nivel de servicio del 100 %, ya que todas las reservas llegan al aeropuerto dentro del margen previsto. La principal mejora aparece en la agrupación de los desplazamientos. El número de viajes al aeropuerto se reduce hasta 12 y la ocupación media de las furgonetas aumenta hasta el 71,40 %. Esto muestra que los hubs permiten aprovechar mucho mejor la capacidad disponible, al concentrar varias reservas en un mismo punto de salida.

Desde el punto de vista económico, el coste total asciende a 627,88 euros, con un coste medio de 2,96 euros por reserva atendida. (Anexo 1).

4.5. Escenario 3: hubs urbanos con almacén central de desbordamiento

El escenario 3 mantiene la lógica de hubs urbanos, pero incorpora un almacén central como mecanismo de respaldo, situado en el entorno de Coslada-Barajas. En este caso, las reservas se asignan inicialmente al hub correspondiente y, cuando la capacidad del hub no es suficiente, pueden derivarse al almacén central, siempre que no sea necesario enviarlas directamente al aeropuerto por restricciones temporales.

La lógica utilizada permite diferenciar entre reservas atendidas directamente desde los hubs y reservas desbordadas al almacén. El objetivo del escenario no es asumir que el almacén central mejora automáticamente la operación, sino comprobar si permite absorber demanda cuando los hubs tienen una capacidad más limitada.

En este caso, el escenario atiende 210 reservas y rechaza 2. Además, 118 reservas se derivan al almacén central, lo que muestra que los hubs presentan una tensión de capacidad relevante bajo esta configuración. El nivel de servicio sobre las reservas atendidas se mantiene en el 100%, aunque el nivel de servicio total se sitúa en el 99,06% por las reservas rechazadas.

La incorporación del almacén permite mantener una cobertura muy elevada, pero aumenta la complejidad operativa. Los viajes al aeropuerto ascienden a 13 y la ocupación media se sitúa en el 65,49 %, ligeramente por debajo del escenario 2. El coste total operativo asciende a 1.393,37 euros, con un coste medio de 6,64 euros por reserva atendida. (Anexo 1).

4.6. Síntesis comparativa del día de referencia

En la siguiente tabla se recoge un resumen de los resultados de cada uno de los cuatro escenarios en el día de referencia, manteniendo constante la misma demanda de entrada en 212 y 317 maletas a gestionar.

Escenario	Reservas atendidas	Reservas desbordadas	Tasa de atención	Viajes al aeropuerto	Ocupación media	Coste total	Coste medio por reserva atendida
Escenario 0	212	0	100%	212	4,04%	5.539,57 €	26,13 €
Escenario 1	59	0	27,83%	27	8,81%	383,19 €	6,49 €
Escenario 2	212	0	100%	12	71,40%	627,88 €	2,96 €
Escenario 3	210	118	99,06%	13	65,49%	1.393,37 €	6,64 €

Tabla 1. Comparación de escenarios operativos en el día de referencia, para parámetros absolutos. Fuente: elaboración propia a partir de los resultados del modelo de simulación. Las reservas desbordada representan la demanda derivada al almacén central al alcanzarse el umbral operativo de saturación del hub

El escenario 1 reduce los desplazamientos y el coste total, aunque rechaza una parte importante de la demanda. Por su parte, el escenario de hubs urbanos presenta el mejor equilibrio del día analizado, ya que atiende todas las reservas, reduce los viajes al aeropuerto, mejora la ocupación de la flota y mantiene el menor coste medio por reserva atendida. El escenario con almacén central conserva una cobertura elevada, pero introduce mayor complejidad y un coste superior. (Anexo 1).

5. Simulación Monte Carlo y estabilidad de los resultados

El análisis de un único día de referencia permite comparar los escenarios bajo unas mismas condiciones, pero no es suficiente para comprobar si las conclusiones se mantienen cuando cambia la composición de la demanda. En un servicio como el planteado, el número de reservas, la ubicación de los alojamientos, los horarios de

recogida, las horas de vuelo y el número de maletas pueden variar de un día a otro. Además, al tratarse de un servicio nuevo, no existen datos históricos reales que permitan observar directamente ese comportamiento.

Por este motivo, se incorpora una simulación Monte Carlo. Esta técnica permite repetir el modelo bajo múltiples combinaciones posibles de entrada y observar cómo varían los resultados ante distintos días simulados. En lugar de trabajar únicamente con un resultado determinista, la simulación permite analizar una distribución de resultados y evaluar la estabilidad de cada escenario ante la incertidumbre asociada a la demanda y a las condiciones operativas (Ross, 2012).

5.1. Objetivo y funcionamiento de la simulación Monte Carlo

Una vez comparados los escenarios en un día de referencia, se utiliza Monte Carlo para analizar su comportamiento ante distintos días posibles de operación. La finalidad es evitar que la evaluación dependa de una única demanda diaria simulada y comprobar si los resultados se mantienen cuando cambian las reservas, los alojamientos seleccionados, las maletas y los horarios.

En el código, esta simulación se implementa mediante la función ejecutar_montecarlo. El proceso consiste en repetir varias veces la generación de una demanda diaria y ejecutar, para cada una de ellas, los cuatro escenarios definidos en el modelo. Cada iteración representa un posible día operativo distinto.

El modelo toma una semilla inicial y la va modificando en cada iteración, por lo que cada día simulado es diferente, pero reproducible. Después, para cada semilla, se genera una nueva demanda diaria mediante la función generar_demanda_diaria, utilizando la base de alojamientos y los parámetros del modelo. A continuación, esa misma demanda se introduce en la función ejecutar_escenarios_dia, que calcula los resultados de los cuatro escenarios bajo las mismas condiciones.

De esta forma, en cada iteración todos los escenarios compiten contra la misma demanda diaria, por lo que los resultados permiten contrastar los resultados de manera justa: las diferencias entre escenarios no se deben a que uno reciba una demanda más favorable que otro, sino a la forma en que cada configuración organiza la operación.

Para cada iteración, el modelo guarda los KPIs obtenidos por escenario e incorpora dos campos adicionales: la semilla utilizada y el número de iteración. Todos los resultados se

consolidan en una única tabla, df_montecarlo, que recoge el comportamiento de cada escenario a lo largo de todos los días simulados.

Posteriormente, el código agrupa esta tabla por escenario y calcula indicadores medios de demanda, tasa de atención, nivel de servicio, viajes al aeropuerto, ocupación y costes. Además, se calculan percentiles del coste medio por reserva total, lo que permite observar no solo el coste promedio, sino también cómo varía el resultado en días más o menos favorables. (Anexo 1).

5.2. Comparación del servicio y la eficiencia operativa

Los resultados de la simulación Monte Carlo permiten analizar cómo se comportan los escenarios cuando la operación se repite sobre distintos días de demanda simulada. En promedio, cada iteración genera alrededor de 199,5 reservas, por lo que el análisis se realiza sobre un volumen diario similar al utilizado en el día de referencia.

El escenario 0 mantiene una tasa de atención del 100 % y un nivel de servicio medio del 99,9 %. Este resultado es coherente con su lógica operativa, ya que cada reserva se atiende de forma individual y no existen restricciones de consolidación. Sin embargo, esta estabilidad exige una operación muy intensiva: el escenario registra una media de 199,482 viajes al aeropuerto por día simulado y una ocupación media de solo el 4,0 %. Esto confirma que el servicio directo garantiza la cobertura, pero apenas aprovecha la capacidad disponible de las furgonetas.

El escenario 1 presenta una tasa de atención media del 27,6 %, igual que su nivel de servicio medio. La principal limitación no está en la puntualidad de las reservas aceptadas, sino en la capacidad del modelo para incorporarlas a rutas viables. Aunque la agrupación reduce los viajes medios al aeropuerto hasta 27,652, la ocupación media sigue siendo baja, con un 8,0 %. Esto indica que la consolidación mediante recogidas en alojamientos mejora el número de desplazamientos respecto al servicio directo, pero sigue condicionada por las ventanas horarias, la capacidad de los vehículos y la construcción de rutas compatibles.

El escenario 2 obtiene el comportamiento más equilibrado. La tasa de atención media alcanza el 99,6 % y el nivel de servicio medio también se sitúa en el 99,6 %. Además, reduce los viajes medios al aeropuerto hasta 11,337 y eleva la ocupación media de las furgonetas al 71,3 %. La explicación está en la propia lógica de hubs: al concentrar el

equipaje en puntos urbanos definidos, la operación puede organizar salidas más cargadas y reducir la dispersión asociada a las recogidas individuales.

El escenario 3 también mantiene una cobertura elevada, con una tasa de atención media y un nivel de servicio medio del 99,1 %. En términos operativos, registra 12,189 viajes medios al aeropuerto y una ocupación media del 66,0 %. La incorporación del almacén central permite mantener una operación consolidada, aunque introduce una capa adicional de gestión que reduce ligeramente el aprovechamiento de la capacidad frente al escenario basado únicamente en hubs.

En la siguiente imagen se puede observar una síntesis de los principales resultados medios obtenidos en la simulación Monte Carlo. Se observan: la capacidad de atención de cada escenario, la intensidad de viajes al aeropuerto, el aprovechamiento de la flota y el coste medio por reserva. (Anexo 1).

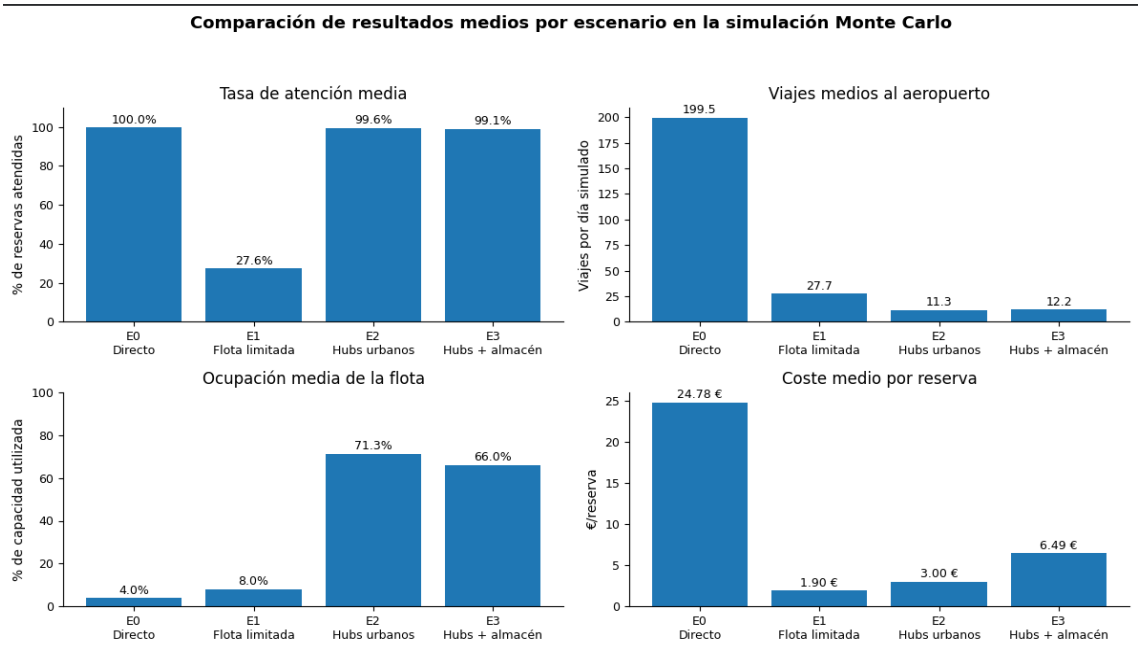


Figura 3. Comparación de resultados medios por escenario en la simulación Monte Carlo. Fuente: elaboración propia a partir de los resultados del modelo de simulación.

5.3. Comparación económica de los escenarios

Desde el punto de vista económico, la simulación Monte Carlo permite comparar el coste medio de operar cada configuración a lo largo de los distintos días simulados. Para ello, se analizan el coste total medio y el coste medio por reserva total, incorporando también los percentiles 5 y 95 para observar la variabilidad del coste entre iteraciones. La Figura

4 sintetiza esta comparación y permite visualizar de forma conjunta tanto el coste medio de cada escenario.

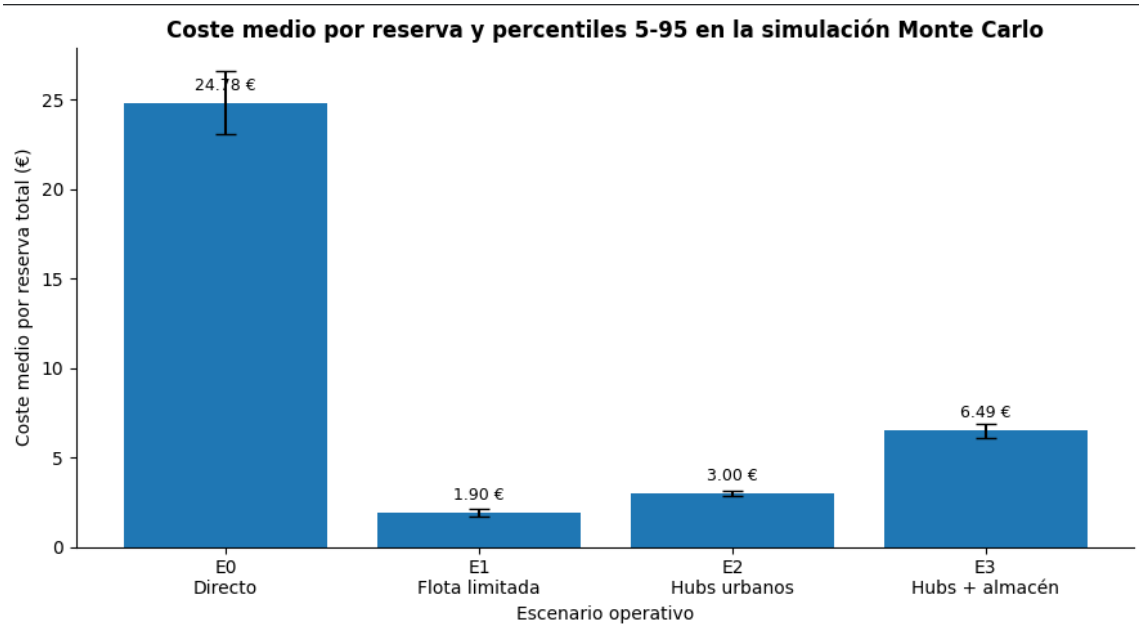


Figura 4. Coste medio por reserva total y percentiles 5-95 en la simulación Monte Carlo. Fuente: elaboración propia a partir de los resultados del modelo de simulación.

El escenario 0 presenta el mayor coste total medio, con 4.944,01 euros por día simulado. También registra el coste medio por reserva más elevado, con 24,78 euros. Este resultado es coherente con su lógica operativa, ya que cada reserva exige un desplazamiento individual al aeropuerto, lo que incrementa de forma directa los costes asociados a kilómetros, tiempos de operación y disponibilidad de vehículos. Además, el coste medio se mueve entre 23,03 y 26,57 euros en los percentiles 5 y 95, demostrando que el escenario directo mantiene un coste elevado de manera recurrente.

El escenario 1 muestra el menor coste total medio, con 377,60 euros, y un coste medio por reserva total de 1,90 euros. No obstante, al operar con una flota restringida y rechazar una parte importante de la demanda, el coste total se reduce, pero no necesariamente refleja una operación económicamente mejor desde el punto de vista del servicio completo. El percentil 5 se sitúa en 1,66€ y el 95 en 2,15€

El escenario 2 presenta un coste total medio de 597,66 euros y un coste medio por reserva total de 3,00 euros. Este resultado lo sitúa como la alternativa más equilibrada desde el punto de vista económico, ya que consigue mantener un coste unitario reducido sin depender de una baja cobertura operativa. Además, los percentiles 5 y 95 del coste medio

se sitúan entre 2,86 y 3,15 euros, lo que indica una variabilidad baja entre los días simulados.

El escenario 3 alcanza un coste total medio de 1.296,62 euros y un coste medio por reserva total de 6,49 euros. Este mayor coste se explica por la incorporación del almacén central, que añade una capa adicional de gestión y disponibilidad operativa. Aunque el escenario mantiene una función de respaldo, su estructura resulta más cara que la basada únicamente en hubs urbanos. El rango entre los percentiles 5 y 95, de 6,08 a 6,89 euros, muestra que esta diferencia económica se mantiene de forma bastante estable en la simulación. (Anexo 1).

5.4. Interpretación global de los resultados Monte Carlo

Los resultados confirman que la forma de organizar la operación tiene un impacto directo en la capacidad de servicio, la eficiencia de los viajes y el coste medio.

El escenario 1 reduce viajes y costes, aunque no consigue absorber una parte suficiente de la demanda. La flota limitada y las restricciones de capacidad, horarios y construcción de rutas hacen que muchas reservas no puedan incorporarse a trayectos viables y quedan como rechazadas.

Entre las alternativas con consolidación, el escenario basado en hubs urbanos presenta el comportamiento más equilibrado. Mantiene una tasa de atención prácticamente completa, reduce de forma significativa los desplazamientos al aeropuerto, alcanza una ocupación media elevada y conserva un coste medio bajo y estable. Esto indica que concentrar el equipaje en puntos urbanos definidos permite transformar una demanda dispersa en una operación más eficiente.

El escenario con almacén central también ofrece una cobertura elevada, pero su mayor complejidad se traduce en más coste y una ligera pérdida de eficiencia operativa frente al modelo de hubs. Por ello, el almacén puede interpretarse como un mecanismo de respaldo útil en contextos de saturación, pero no como la configuración más eficiente en condiciones normales. (Anexo 1).

6. Conclusiones

El objetivo de este trabajo ha sido analizar, desde una perspectiva operativa, qué configuración permite prestar de forma más eficiente un servicio urbano de recogida, custodia y entrega de equipaje en el aeropuerto de Madrid. La pregunta de investigación planteaba cual era el diseño operativo capaz de equilibrar coste, tiempo y nivel de servicio en un contexto urbano condicionado por la dispersión de alojamientos, las restricciones horarias, la capacidad de las furgonetas y la necesidad de entregar el equipaje con suficiente margen antes del vuelo.

A partir del modelo construido, la respuesta a la pregunta de investigación es clara: bajo las condiciones simuladas, la configuración basada en centros urbanos de consolidación es la alternativa más sólida. Este escenario permite mantener un nivel de servicio prácticamente completo, reducir de forma significativa los desplazamientos al aeropuerto, aprovechar mejor la capacidad de las furgonetas y mantener un coste medio reducido y estable. Los resultados sugieren que la ventaja principal del modelo de hubs está en concentrar una demanda inicialmente dispersa en puntos urbanos desde los que se pueden organizar salidas más eficientes hacia el aeropuerto.

El escenario directo individual cumple una función útil dentro del análisis, ya que actúa como referencia para medir el efecto de las alternativas de consolidación. Su principal ventaja es que permite atender toda la demanda y mantener el nivel de servicio, pero lo hace a costa de una estructura operativa poco eficiente. En el día de referencia, este escenario realiza un viaje por reserva, alcanza una ocupación media de solo el 4,04 % y registra un coste medio de 26,13 euros por reserva atendida. La simulación Monte Carlo confirma esta conclusión: aunque mantiene una tasa de atención del 100 %, presenta el mayor coste medio y una ocupación muy baja.

El escenario de flota limitada con rutas consolidadas mejora algunos aspectos respecto al servicio directo, especialmente en la reducción del número de viajes y del coste total. Sin embargo, su principal limitación es la baja capacidad de absorción de la demanda. En el día de referencia atiende 59 reservas y rechaza 153, manteniendo la puntualidad de las reservas aceptadas, pero reduciendo el nivel de servicio total al 27,83 %. La simulación Monte Carlo refuerza esta idea, con una tasa de atención media del 27,6 %. En la práctica, el escenario 1 reduce los viajes, pero con tres vehículos no consigue absorber suficiente

demanda, porque muchas reservas no encajan en rutas compatibles por horario, ubicación o capacidad.

Frente a estas dos alternativas, el escenario de hubs urbanos presenta el mejor equilibrio. En el día de referencia atiende toda la demanda, no genera rechazos, reduce los viajes al aeropuerto a 12 y eleva la ocupación media de las furgonetas hasta el 71,40 %. Además, obtiene un coste medio de 2,96 euros por reserva atendida. Los resultados del Monte Carlo confirman que este comportamiento no depende de un único día favorable: el escenario mantiene una tasa de atención media del 99,6 %, un nivel de servicio medio del 99,6 %, una ocupación media del 71,3 % y un coste medio por reserva total de 3,00 euros. También presenta una variabilidad reducida, con un rango de coste medio entre 2,86 y 3,15 euros en los percentiles 5 y 95.

El escenario con almacén central de desbordamiento también ofrece una cobertura elevada, pero no mejora la eficiencia del modelo de hubs urbanos. Su aportación principal es actuar como mecanismo de respaldo cuando la capacidad de los hubs se tensiona. En el día de referencia atiende 210 reservas, rechaza únicamente 2 y deriva 118 reservas al almacén central. Sin embargo, esta mayor capacidad de respaldo introduce más complejidad operativa, aumenta los costes y reduce ligeramente la ocupación media respecto al escenario de hubs. La simulación Monte Carlo confirma esta interpretación: el escenario mantiene una tasa de atención media del 99,1 %, pero con un coste medio por reserva total de 6,49 euros, superior al del modelo basado únicamente en hubs. Por tanto, el almacén central puede ser útil en situaciones de saturación o como mecanismo de contingencia, pero no aparece como la configuración más eficiente en condiciones normales de operación.

No obstante, las conclusiones deben interpretarse teniendo en cuenta las limitaciones del modelo. La principal limitación procede de la propia naturaleza del servicio analizado: al tratarse de una propuesta nueva y no de una operación ya implantada, no existen datos históricos reales sobre reservas, horarios de uso, distribución geográfica de la demanda, número de maletas por cliente o comportamiento efectivo de los usuarios. Por ello, el modelo no puede apoyarse en una base empírica completa de funcionamiento pasado, sino que debe inferir parte de los datos y construir la demanda mediante simulaciones basadas en supuestos razonables.

Esta limitación afecta especialmente a la generación de la demanda diaria y a la parametrización operativa. Aunque las localizaciones de origen se apoyan en alojamientos turísticos reales de Madrid, la asignación de reservas a esos alojamientos se realiza de forma aleatoria y no pondera la probabilidad de uso por zona, tipo de alojamiento o perfil de cliente. Del mismo modo, variables como las ventanas de recogida, la distribución de maletas, los tiempos de parada, los tiempos de gestión, las capacidades de hubs o determinados costes operativos se han definido a partir de criterios razonables y de la información disponible, pero podrían variar en una operación real.

Estas limitaciones acotan el alcance del modelo. Los resultados no deben interpretarse como una predicción exacta de una operación real, sino como una comparación estructurada entre alternativas bajo los mismos supuestos. En este sentido, el trabajo cumple su propósito: transformar una idea de servicio en un problema operativo medible, identificar las variables que condicionan su funcionamiento y evaluar alternativas mediante indicadores comparables.

Como líneas futuras de mejora, sería recomendable incorporar datos reales de demanda si el servicio llegara a probarse mediante un piloto, especialmente sobre distribución geográfica de clientes, número de maletas y disposición a utilizar los puntos de recogida de equipaje. También sería útil optimizar la localización exacta de los hubs introduciendo costes inmobiliarios detallados y estudiar escenarios de demanda estacional o de mayor saturación. Otra ampliación relevante consistiría en incorporar modelos de ruteo más avanzados. Finalmente, contando con unos datos históricos relevantes y la incorporación de datos de tráfico en tiempo real, se podría diseñar un algoritmo para optimizar al máximo el flujo del equipaje hasta el aeropuerto, permitiendo reducir el tamaño de los locales de recogida y de entrega en aeropuerto, reduciendo costes de alquiler.

7. Anexo

7.1. Anexo 1: Tabla de trazabilidad del código

Apartado del TFG	**Descripción**	**Bloques necesarios para su ejecución**
3.3	Construcción y limpieza de la base espacial de alojamientos	Bloques 1-2
3.4	Cálculo de distancias y validación de rutas hacia el aeropuerto	Bloques 1-5
3.5	Definición de parámetros operativos del modelo	Bloque 6
3.7	Parametrización económica y tratamiento de costes	Bloques 6-7
3.6	Generación de la demanda diaria simulada	Bloques 6-9
4.2	Escenario 0: servicio directo individual	Bloques 6-10
4.3	Escenario 1: flota limitada con rutas consolidadas	Bloques 6-7 y 11-12
4.4	Escenario 2: hubs urbanos de consolidación	Bloques 6-7 y 13-16
4.5	Escenario 3: hubs urbanos con almacén central	Bloques 6-7 y 13-19
3.8 y 4.6	Cálculo de KPIs y comparación homogénea de escenarios	Bloque 20
5.1 a 5.4	Simulación Monte Carlo y resumen de resultados	Bloques 20-21

```
# BLOQUE 1. Librerías y configuración inicial

from pathlib import Path
import json
import math
import time
import numpy as np
import pandas as pd
import requests
import math

pd.set_option("display.max_columns", 100)
pd.set_option("display.width", 180)

# BLOQUE 2. Carga y limpieza de la base espacial de alojamientos

PATH_GEOJSON = DATA_DIR / "export (2).geojson"

if not PATH_GEOJSON.exists():
    raise FileNotFoundError(f"No se encuentra el archivo: {PATH_GEOJSON}")

with open(PATH_GEOJSON, "r", encoding="utf-8") as f:
    geojson_data = json.load(f)

features = geojson_data["features"]

def extraer_alojamientos_desde_geojson(features): # Convertimos Geojson en una tabla, conserva
puntos validos (Coordenadas lógicas) y da un nombre si no tiene.

    registros = []

    for feature in features:
        props = feature.get("properties", {}) or {}
        geom = feature.get("geometry", {}) or {}

        if geom.get("type") != "Point":
            continue

        coords = geom.get("coordinates", None)

        if coords is None or len(coords) < 2:
            continue

        lon = coords[0]
        lat = coords[1]

        osm_id = props.get("@id")
        name = props.get("name") or props.get("brand") or props.get("short_name")

        tourism = props.get("tourism")
        building = props.get("building")

        if tourism in ["hotel", "hostel", "guest_house", "apartment", "motel"]:
            tipo = tourism
        elif building == "hotel":
            tipo = "hotel"
        else:
            tipo = tourism or building or "alojamiento"

        registros.append({
            "osm_id": osm_id,
            "name": name,
            "type": tipo,
            "tourism": tourism,
            "building": building,
            "lat": lat,
            "lon": lon,
            "addr_street": props.get("addr:street"),
            "addr_housenumber": props.get("addr:housenumber")
        })

    df = pd.DataFrame(registros)

    # Elimina registros sin coordenadas
    df = df.dropna(subset=["lat", "lon"]).copy()

    # Elimina duplicados de OSM
    df = df.drop_duplicates(subset=["osm_id"]).copy()

    # identificador propio
    df = df.reset_index(drop=True)
    df["alojamiento_id"] = np.arange(1, len(df) + 1)
```

```
    # Nombra si no tiene
    df["alojamiento_nombre"] = df["name"]

    mask_sin_nombre = df["alojamiento_nombre"].isna()

    df.loc[mask_sin_nombre, "alojamiento_nombre"] = (
        df.loc[mask_sin_nombre, "type"]
        .fillna("alojamiento")
        .str.title()
        + "_"
        + df.loc[mask_sin_nombre, "alojamiento_id"].astype(str).str.zfill(4)
    )

    columnas_finales = [
        "alojamiento_id",
        "osm_id",
        "alojamiento_nombre",
        "type",
        "tourism",
        "building",
        "lat",
        "lon",
        "addr_street",
        "addr_housenumber"
    ]

    return df[columnas_finales].copy()

df_alojamientos = extraer_alojamientos_desde_geojson(features)

print("Alojamientos limpios:", len(df_alojamientos))
print("Alojamientos sin nombre original:", df_alojamientos["alojamiento_nombre"].str.contains("_",
regex=False).sum())

display(df_alojamientos.head())
display(df_alojamientos["type"].value_counts())

# BLOQUE 3. Cálculo de distancia directa al aeropuerto

AEROPUERTO = {
    "nombre": "Aeropuerto Adolfo Suárez Madrid-Barajas",
    "lat": 40.4983,
    "lon": -3.5676
}

def haversine_km(lat1, lon1, lat2, lon2): #Calculamos distancias entre puntos según el metodo de
Haversine.

    R = 6371  # radio de la tierra

#convertimos a radianes
    lat1_rad = np.radians(lat1)
    lon1_rad = np.radians(lon1)
    lat2_rad = np.radians(lat2)
    lon2_rad = np.radians(lon2)

    dlat = lat2_rad - lat1_rad
    dlon = lon2_rad - lon1_rad
#aplicamos la formula
    a = (
        np.sin(dlat / 2) ** 2
        + np.cos(lat1_rad) * np.cos(lat2_rad) * np.sin(dlon / 2) ** 2
    )

    c = 2 * np.arcsin(np.sqrt(a))

    return R * c

df_alojamientos["distancia_directa_aeropuerto_km"] = haversine_km(
    df_alojamientos["lat"],
    df_alojamientos["lon"],
    AEROPUERTO["lat"],
    AEROPUERTO["lon"]
)

display(df_alojamientos[["distancia_directa_aeropuerto_km"]].describe())

# Vemos los alojamientos que están más lejos.
display(
    df_alojamientos.sort_values("distancia_directa_aeropuerto_km", ascending=False)
    .head(10)
)

# BLOQUE 4. Configuración de OpenRouteService y función de rutas
```

```
import openrouteservice
from getpass import getpass

ORS_API_KEY = getpass("Pega aquí API key: ").strip()

client = openrouteservice.Client(
    key=ORS_API_KEY,
    timeout=60
)

import requests
import time
import numpy as np

def calcular_ruta_ors_requests( #Calculamos la eltiempo y la longitud del viaje según ORS
    lat_origen,
    lon_origen,
    lat_destino,
    lon_destino,
    api_key,
    radio_origen=1000,
    radio_destino=1000,
    max_reintentos=3
):

    url = "https://api.openrouteservice.org/v2/directions/driving-car/geojson"

    headers = {
        "Authorization": api_key,
        "Content-Type": "application/json"
    }

    body = {
        "coordinates": [
            [float(lon_origen), float(lat_origen)],
            [float(lon_destino), float(lat_destino)]
        ],
        "radiuses": [radio_origen, radio_destino]
    }

    ultimo_error = None

    for intento in range(max_reintentos):
        try:
            response = requests.post(url, headers=headers, json=body, timeout=60)

            if response.status_code == 200:
                data = response.json()
                props = data["features"][0]["properties"]

                if "summary" in props:
                    distancia_m = props["summary"]["distance"]
                    duracion_s = props["summary"]["duration"]
                else:
                    distancia_m = props["segments"][0]["distance"]
                    duracion_s = props["segments"][0]["duration"]

                return distancia_m / 1000, duracion_s / 60, None

            elif response.status_code in [429, 500, 502, 503, 504]:
                ultimo_error = f"{response.status_code}: {response.text[:300]}"
                espera = 10 * (intento + 1)
                print(f"Error temporal {response.status_code}. Reintentando en {espera}
segundos...")
                time.sleep(espera)
                continue

            else:
                ultimo_error = f"{response.status_code}: {response.text[:300]}"
                return np.nan, np.nan, ultimo_error

        except Exception as e:
            ultimo_error = str(e)
            espera = 10 * (intento + 1)
            print(f"Excepción en intento {intento + 1}: {ultimo_error}. Reintentando en {espera}
segundos...")
            time.sleep(espera)

    return np.nan, np.nan, ultimo_error

# BLOQUE 5. Cálculo y validación de rutas hacia el aeropuerto

PATH_CACHE_TIEMPOS = DATA_DIR / "tiempos_alojamientos_aeropuerto_ampliado_T4.xlsx"
```

```
if PATH_CACHE_TIEMPOS.exists():
    df_cache = pd.read_excel(PATH_CACHE_TIEMPOS)

    df_cache_validas = df_cache.dropna(subset=[
        "distancia_aeropuerto_km",
        "tiempo_aeropuerto_min"
    ]).copy()

    df_cache_validas = df_cache_validas[
        (df_cache_validas["distancia_aeropuerto_km"] > 0)
        & (df_cache_validas["tiempo_aeropuerto_min"] > 0)
    ].copy()

    ids_calculados = set(df_cache_validas["alojamiento_id"].tolist())

    print("Caché existente cargada.")

else:
    df_cache_validas = pd.DataFrame()
    ids_calculados = set()
    print("No existe caché válida previa. Se calculará desde cero.")

resultados_nuevos = []

total = len(df_alojamientos)

for idx, row in df_alojamientos.iterrows():

    alojamiento_id = row["alojamiento_id"]

    if alojamiento_id in ids_calculados:
        continue

    distancia_km, duracion_min, error = calcular_ruta_ors_requests(
        lat_origen=row["lat"],
        lon_origen=row["lon"],
        lat_destino=AEROPUERTO["lat"],
        lon_destino=AEROPUERTO["lon"],
        api_key=ORS_API_KEY
    )

    resultados_nuevos.append({
        "alojamiento_id": alojamiento_id,
        "osm_id": row.get("osm_id", None),
        "alojamiento_nombre": row["alojamiento_nombre"],
        "tipo": row["type"],
        "lat": row["lat"],
        "lon": row["lon"],
        "distancia_directa_aeropuerto_km": row["distancia_directa_aeropuerto_km"],
        "distancia_aeropuerto_km": distancia_km,
        "tiempo_aeropuerto_min": duracion_min,
        "error_ors": error
    })

    if len(resultados_nuevos) % 25 == 0:
        df_temp = pd.concat(
            [df_cache_validas, pd.DataFrame(resultados_nuevos)],
            ignore_index=True
        )

        df_temp.to_excel(PATH_CACHE_TIEMPOS, index=False)

        rutas_validas = df_temp["tiempo_aeropuerto_min"].notna().sum()
        errores = df_temp["error_ors"].notna().sum()

        print(
            f"Guardado parcial: {len(df_temp)} filas | "
            f"Rutas válidas: {rutas_validas} | "
            f"Errores: {errores}"
        )

        time.sleep(1)

df_tiempos_ampliado = pd.concat(
    [df_cache_validas, pd.DataFrame(resultados_nuevos)],
    ignore_index=True
)

df_tiempos_ampliado.to_excel(PATH_CACHE_TIEMPOS, index=False)

print("Cálculo finalizado.")
print("Total filas guardadas:", len(df_tiempos_ampliado))
print("Rutas válidas:", df_tiempos_ampliado["tiempo_aeropuerto_min"].notna().sum())
print("Errores:", df_tiempos_ampliado["error_ors"].notna().sum())
```

```
display(df_tiempos_ampliado.head())

df_base = df_tiempos_ampliado.copy()

# Filtrar rutas válidas
df_base = df_base.dropna(subset=[
    "tiempo_aeropuerto_min",
    "distancia_aeropuerto_km"
]).copy()

df_base = df_base[df_base["tiempo_aeropuerto_min"] > 0].copy()
df_base = df_base[df_base["distancia_aeropuerto_km"] > 0].copy()

df_base = df_base.reset_index(drop=True)

print("Base final válida para el modelo:", len(df_base))

display(
    df_base[[
        "distancia_directa_aeropuerto_km",
        "distancia_aeropuerto_km",
        "tiempo_aeropuerto_min"
    ]].describe()
)

# Ratio carretera / distancia directa: coeficiente que mide si la distancia de ORS es superior a la
linea recta que sacamos por Harvesine
df_base["ratio_carretera_directa"] = (
    df_base["distancia_aeropuerto_km"]
    / df_base["distancia_directa_aeropuerto_km"]
)

display(df_base[["ratio_carretera_directa"]].describe())

# Posibles valores raros
display(
    df_base.sort_values("tiempo_aeropuerto_min", ascending=False)
    .head(10)
)

# BLOQUE 6. Parámetros operativos del modelo

SEED_GLOBAL = 42

# Parámetros derivados del modelo financiero de Manuel Recio Rubiato

SUELDOS_ANUALES = 210000
NUM_TRABAJADORES = 6
HORAS_ANUALES_POR_TRABAJADOR = 1920 #resultado de multiplicar 40 horas semanales por 48 semanas

COSTE_OPERATIVO_HORA = (
    SUELDOS_ANUALES
    / (NUM_TRABAJADORES * HORAS_ANUALES_POR_TRABAJADOR)
)

COSTE_OPERATIVO_MINUTO = COSTE_OPERATIVO_HORA / 60

# Tiempos utilizados para transformar operación en coste: No tenemos una referencia exacta, inferido
de experiencia personal en situaciones comparables que ahora detallo
TIEMPO_PARADA_RECOGIDA_MIN = 5 # tiempo de carga observado en Uber. Desde que la app marcaba que
había llegado, hasta que arrancó
TIEMPO_DESCARGA_AEROPUERTO_MIN = 6 #descarga de una furgoneta llena de maletas. Resultado de
multiplicar 25 maletas por una descarga de 15 segundo por maleta (redondeo)
TIEMPO_MANEJO_MALETA_MIN = 1.5
TIEMPO_GESTION_HUB_MALETA_MIN = 3 #recepción y orden de las maletas

PARAMS = {
    # Simulación
    "fecha_operacion": "2026-01-01",
    "num_iteraciones_montecarlo": 365, #Lo establecemos para un año

    # Demanda
    "demanda_media_reservas": 200, #Basado en el SOM de Manuel y el estudio de demanda de Jorge
Santander Ramos (5350 clientes al mes, la estimación base sería de 178 reservas diarias, pero
redondeamos por tener margen operativo)
    "demanda_minima": 50, #Margen para montecarlo
    "demanda_maxima": 400, #Margen para montecarlo

    # Distribución de maletas por reserva: Probabilidad asignada de esa manera al entender de que la
reserva se hace por persona y no por grupo
    "prob_maletas": {
        1: 0.60,
        2: 0.30,
        3: 0.10
    },
```

```
    # Horarios de recogida, en minutos desde medianoche (Por la configuración del servicio, solo se
recoge en la mañana y se entrega por la tarde)
    "recogida_min": 9 * 60 + 30,
    "recogida_moda": 11 * 60, # moda por ser horario de check out común.
    "recogida_max": 13 * 60,

    # Horarios de vuelo, en minutos desde medianoche sale de aquí la moda:
https://www.skyscanner.es/vuelos/llegadas-salidas/mad/madrid-barajas-llegadas-salidas . Observación
Manual de dos semanas, situan entra las 18.00 y las 20.00
    "vuelo_min": 15 * 60,
    "vuelo_moda": 19 * 60,
    "vuelo_max": 23 * 60 + 30,

    # Servicio
    "margen_entrega_aeropuerto_min": 180, #Por seguridad de nuestro servicio y como compromiso con
cliente, las maletas deben de estar en aeruopuerto al menos 3 horas antes de que salga el avión
    "buffer_minimo_operativo_min": 30, #por las restricciones temporales de la política de empresa,
no será posible que haya una solicitud de menos de 30 minutos.
    "tiempo_retorno_ciudad_min": 25,

    # Vehículos
    "capacidad_furgoneta_maletas": 37, #resultado redondeado de coger carga de una mercedes vito
long (850 kilos) entre peso de una maleta mediana-grande llena (20 kilos), menos un margen del 10%,
para que haya holgura

    # Tiempos operativos
    "tiempo_parada_recogida_min": TIEMPO_PARADA_RECOGIDA_MIN,
    "tiempo_descarga_aeropuerto_min": TIEMPO_DESCARGA_AEROPUERTO_MIN,
    "tiempo_manejo_maleta_min": TIEMPO_MANEJO_MALETA_MIN,
    "tiempo_gestion_hub_maleta_min": TIEMPO_GESTION_HUB_MALETA_MIN,

    # Costes variables de combustible: datos del TFG de Manuel Recio Rubiato
    "consumo_litros_100km": 6.7,
    "precio_combustible_litro": 1.80,
    "coste_por_km": 6.7 / 100 * 1.80,

    # Coste operativo imputado a partir del modelo financiero
    "sueldos_anuales": SUELDOS_ANUALES,
    "num_trabajadores": NUM_TRABAJADORES,
    "horas_anuales_por_trabajador": HORAS_ANUALES_POR_TRABAJADOR,
    "coste_operativo_hora": COSTE_OPERATIVO_HORA,
    "coste_operativo_minuto": COSTE_OPERATIVO_MINUTO,

    # Costes unitarios derivados de tiempos operativos
    "coste_manejo_por_maleta": TIEMPO_MANEJO_MALETA_MIN * COSTE_OPERATIVO_MINUTO,
    "coste_parada_recogida": TIEMPO_PARADA_RECOGIDA_MIN * COSTE_OPERATIVO_MINUTO,
    "coste_descarga_aeropuerto": TIEMPO_DESCARGA_AEROPUERTO_MIN * COSTE_OPERATIVO_MINUTO,
    "coste_gestion_hub_por_maleta": TIEMPO_GESTION_HUB_MALETA_MIN * COSTE_OPERATIVO_MINUTO,

    # Factores de retorno (para computar el coste operativo del retorno de una furgoneta). Existen 3
posibilidades a otro punto de la ciudad, al hub o al almacen
    "factor_retorno_directo": 0.75,
    "factor_retorno_hub": 1.00,
    "factor_retorno_almacen": 1.00,

    # Coste de disponibilidad de flota. Extraido del TFG de Manuel Recio Rubiato
    "alquiler_vehiculo_anual": 12000,
    "dias_operacion_anio": 360,
    "coste_disponibilidad_vehiculo_dia": 12000 / 360
}

print("Parámetros cargados.")
print("Demanda media diaria:", PARAMS["demanda_media_reservas"])
print("Coste combustible por km:", round(PARAMS["coste_por_km"], 4))
print("Coste combustible trayecto referencia 36 km:", round(36 * PARAMS["coste_por_km"], 2))
print("Coste operativo hora:", round(PARAMS["coste_operativo_hora"], 2))
print("Coste operativo minuto:", round(PARAMS["coste_operativo_minuto"], 3))
print("Coste manejo por maleta:", round(PARAMS["coste_manejo_por_maleta"], 2))
print("Coste parada recogida:", round(PARAMS["coste_parada_recogida"], 2))
print("Coste descarga aeropuerto:", round(PARAMS["coste_descarga_aeropuerto"], 2))
print("Coste gestión hub por maleta:", round(PARAMS["coste_gestion_hub_por_maleta"], 2))
print("Coste disponibilidad vehículo/día:", round(PARAMS["coste_disponibilidad_vehiculo_dia"], 2))

# BLOQUE 7. Funciones de coste

def calcular_coste_viaje( #Esta función de coste contempla los distintos escenarios operativos que
estudiamos, teniendolos en cuenta la realidad de que son distintos y pueden tener costes extras
según la estructura
    distancia_km,
    duracion_min=None,
    params=PARAMS,
    num_maletas=0,
    num_paradas=0,
    factor_retorno=0.0,
```

```
    pasa_por_hub=False,
    incluye_entrega_aeropuerto=True
):
    distancia_operativa = distancia_km * (1 + factor_retorno)

    coste = distancia_operativa * params["coste_por_km"]
    coste += num_maletas * params["coste_manejo_por_maleta"]
    coste += num_paradas * params["coste_parada_recogida"]

    if incluye_entrega_aeropuerto:
        coste += params["coste_descarga_aeropuerto"]

    if pasa_por_hub:
        coste += num_maletas * params["coste_gestion_hub_por_maleta"]

    return coste

def calcular_coste_disponibilidad_flota(num_vehiculos, params=PARAMS): # teniendo en consideración
las furgonetas disponibles según el coste de la misma calculado por Manuel Recio
    return num_vehiculos * params["coste_disponibilidad_vehiculo_dia"]

def estimar_vehiculos_por_concurrencia(df, inicio_col, fin_col): #calculamos el numero de vehiculos
que podríamos llegar a necesitar (importante en escenario 0)
    eventos = []

    for _, row in df.iterrows():
        if pd.notna(row[inicio_col]) and pd.notna(row[fin_col]):
            eventos.append((row[inicio_col], 1))
            eventos.append((row[fin_col], -1))

    eventos = sorted(eventos, key=lambda x: (x[0], x[1]))

    activos = 0
    max_activos = 0

    for _, cambio in eventos:
        activos += cambio
        max_activos = max(max_activos, activos)

    return max_activos

# BLOQUE 8. Generación de demanda diaria simulada

def minutos_a_datetime(fecha_base, minutos): #conversion de tiempos
    return pd.to_datetime(fecha_base) + pd.to_timedelta(minutos, unit="m")

def redondear_minutos(minutos, intervalo=10):
    return int(round(minutos / intervalo) * intervalo)

def generar_demanda_diaria(df_base, seed, params=PARAMS, demanda_media=None): #Generamos reservas
simuladas, acotadas a nuestros parametros operativos.
    rng = np.random.default_rng(seed)

    if demanda_media is None:
        demanda_media = params["demanda_media_reservas"]

    n_reservas = rng.poisson(demanda_media) #Utilizamos una distribución de Poisson para modelar la
cantidad de reservas que pueden entrar en un día, con la media hemos calculado.
    n_reservas = max(params["demanda_minima"], n_reservas)
    n_reservas = min(params["demanda_maxima"], n_reservas)

    fecha = params["fecha_operacion"]
    maletas_valores = np.array(list(params["prob_maletas"].keys()))
    maletas_probs = np.array(list(params["prob_maletas"].values()))

    reservas = []
    intentos = 0
    max_intentos = n_reservas * 30

    while len(reservas) < n_reservas and intentos < max_intentos:
        intentos += 1

        alojamiento = df_base.iloc[rng.integers(0, len(df_base))] # Seleccionamos aleatoriamente un
alojamiento. En ausencia de datos historicos de reservas se asume misma probabilidad.
        maletas = int(rng.choice(maletas_valores, p=maletas_probs)) #Seleccionamos una cantidad de
maletas aleatoria, pero delimitada con pesos.

        # Generamos la hora de recogida y de vuelo una distribución triangular, con parametros
definidos por el horario operativo y la moda observada de salidas de vuelos del Aeropuerto de Madrid
        recogida_min = redondear_minutos(
            rng.triangular(params["recogida_min"], params["recogida_moda"], params["recogida_max"])
        )

        vuelo_min = redondear_minutos(
            rng.triangular(params["vuelo_min"], params["vuelo_moda"], params["vuelo_max"])
```

```
        )

        hora_recogida = minutos_a_datetime(fecha, recogida_min)
        hora_vuelo = minutos_a_datetime(fecha, vuelo_min)
        hora_objetivo = hora_vuelo - pd.to_timedelta(params["margen_entrega_aeropuerto_min"],
unit="m")
        margen = (hora_objetivo - hora_recogida).total_seconds() / 60

        if margen < params["buffer_minimo_operativo_min"]:
            continue

        reservas.append({
            "reserva_id": len(reservas) + 1,
            "alojamiento_id": alojamiento["alojamiento_id"],
            "alojamiento_nombre": alojamiento["alojamiento_nombre"],
            "tipo": alojamiento["tipo"],
            "lat": alojamiento["lat"],
            "lon": alojamiento["lon"],
            "maletas": maletas,
            "hora_recogida": hora_recogida,
            "hora_vuelo": hora_vuelo,
            "hora_objetivo_aeropuerto": hora_objetivo,
            "margen_operativo_min": margen,
            "distancia_aeropuerto_km": alojamiento["distancia_aeropuerto_km"],
            "tiempo_aeropuerto_min": alojamiento["tiempo_aeropuerto_min"]
        })

    if len(reservas) < n_reservas:
        raise RuntimeError("No se pudieron generar suficientes reservas válidas.")

    return pd.DataFrame(reservas)

# BLOQUE 9. Día de referencia

df_demanda_test = generar_demanda_diaria(
    df_base=df_base,
    seed=SEED_GLOBAL,
    params=PARAMS,
    demanda_media=PARAMS["demanda_media_reservas"]
)

print("Reservas:", len(df_demanda_test))
print("Maletas:", df_demanda_test["maletas"].sum())
display(df_demanda_test.head())

# BLOQUE 10. Escenario 0: servicio directo individual

def simular_escenario_directo(df_demanda, params=PARAMS):
    df = df_demanda.copy()

    df["tiempo_total_servicio_min"] = (
        params["tiempo_parada_recogida_min"]
        + df["tiempo_aeropuerto_min"]
        + params["tiempo_descarga_aeropuerto_min"]
    )

    df["hora_entrega_estimada"] = df["hora_recogida"] + pd.to_timedelta(
        df["tiempo_total_servicio_min"], unit="m"
    )

    df["hora_fin_operativa"] = df["hora_entrega_estimada"] + pd.to_timedelta(
        params["tiempo_retorno_ciudad_min"], unit="m"
    )

    df["entregada_en_plazo"] = df["hora_entrega_estimada"] <= df["hora_objetivo_aeropuerto"]
    df["estado"] = np.where(df["entregada_en_plazo"], "entregada_en_plazo", "fuera_de_plazo")

    df["coste_reserva"] = df.apply(
        lambda row: calcular_coste_viaje(
            distancia_km=row["distancia_aeropuerto_km"],
            duracion_min=row["tiempo_total_servicio_min"],
            params=params,
            num_maletas=row["maletas"],
            num_paradas=1,
            factor_retorno=params["factor_retorno_directo"]
        ),
        axis=1
    )

    df["viaje_id"] = df["reserva_id"]
    df["ocupacion_viaje"] = df["maletas"] / params["capacidad_furgoneta_maletas"]

    reservas_totales = len(df)
    reservas_atendidas = len(df)
    reservas_en_plazo = df["entregada_en_plazo"].sum()
```

```
    coste_variable = df["coste_reserva"].sum()
    num_vehiculos = estimar_vehiculos_por_concurrencia(df, "hora_recogida", "hora_fin_operativa")
    coste_flota = calcular_coste_disponibilidad_flota(num_vehiculos, params)
    coste_total = coste_variable + coste_flota

    kpis = {
        "escenario": "E0_Directo_individual",
        "reservas_totales": reservas_totales,
        "reservas_atendidas": reservas_atendidas,
        "reservas_rechazadas": 0,
        "reservas_desbordadas_almacen": 0,
        "tasa_atencion": 1.0 if reservas_totales > 0 else np.nan,
        "nivel_servicio_total": reservas_en_plazo / reservas_totales if reservas_totales > 0 else
np.nan,
        "nivel_servicio_atendidas": reservas_en_plazo / reservas_atendidas if reservas_atendidas > 0
else np.nan,
        "maletas_totales": df["maletas"].sum(),
        "viajes_aeropuerto": len(df),
        "ocupacion_media": df["ocupacion_viaje"].mean(),
        "num_vehiculos_operativos": num_vehiculos,
        "coste_variable_total": coste_variable,
        "coste_disponibilidad_flota": coste_flota,
        "coste_total": coste_total,
        "coste_medio_reserva_atendida": coste_total / reservas_atendidas if reservas_atendidas > 0
else np.nan,
        "coste_medio_reserva_total": coste_total / reservas_totales if reservas_totales > 0 else
np.nan
    }

    return df, pd.DataFrame([kpis])

# BLOQUE 11. Escenario 1: reglas de ruta consolidada

#Supuestos operativos razonables para la construcción de un primer modelo
PARAMS.update({
    "max_paradas_por_ruta": 6, #evitar rutas excesivamente largas que comprometan la llegada puntual
al aeropuerto
    "ventana_recogida_ruta_min": 45, # permite agrupar clientes que reserven dentro de un rango
horario parecido
    "velocidad_media_urbana_kmh": 24, # Extraido de aqui:
https://app.atlas.co/shared/sk0OIALcXCHfeyiq8cfP?loc=-
3.6709%2C40.4389%2C15.5728z&public=true&trk=public_post_comment-text
    "factor_correccion_haversine": 1.45, #Calculado como la media de Ratio de carretera de esta
tabla: base_modelo_alojamientos_T4_ORS
    "tiempo_retorno_ciudad_min": 22 #Media de tiempo minimo de tabla:
base_modelo_alojamientos_T4_ORS
})

def distancia_haversine_km(lat1, lon1, lat2, lon2): #determinar distancia en linea recta
    R = 6371
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
    return R * 2 * np.arcsin(np.sqrt(a))

def ordenar_paradas_hacia_aeropuerto(df_ruta): #Ordenar la parada de mayor a menos distancia
    return df_ruta.sort_values("distancia_aeropuerto_km", ascending=False).copy()

def estimar_ruta_consolidada(df_ruta, hora_inicio, params=PARAMS): # Sacar el tiempo de llegada
según la ruta establecida.
    df_orden = ordenar_paradas_hacia_aeropuerto(df_ruta)

    tiempo_actual = hora_inicio
    distancia_total = 0
    tiempo_total = 0
    lat_ant = None
    lon_ant = None

    for _, row in df_orden.iterrows():
        if lat_ant is not None:
            distancia = distancia_haversine_km(lat_ant, lon_ant, row["lat"], row["lon"])
            distancia *= params["factor_correccion_haversine"]
            tiempo = distancia / params["velocidad_media_urbana_kmh"] * 60
            distancia_total += distancia
            tiempo_total += tiempo
            tiempo_actual += pd.to_timedelta(tiempo, unit="m")

        if tiempo_actual < row["hora_recogida"]:
            espera = (row["hora_recogida"] - tiempo_actual).total_seconds() / 60
            tiempo_total += espera
            tiempo_actual = row["hora_recogida"]

        tiempo_total += params["tiempo_parada_recogida_min"]
```

```
        tiempo_actual += pd.to_timedelta(params["tiempo_parada_recogida_min"], unit="m")

        lat_ant = row["lat"]
        lon_ant = row["lon"]

    ultima = df_orden.iloc[-1]
    distancia_total += ultima["distancia_aeropuerto_km"]
    tiempo_total += ultima["tiempo_aeropuerto_min"] + params["tiempo_descarga_aeropuerto_min"]
    tiempo_actual += pd.to_timedelta(
        ultima["tiempo_aeropuerto_min"] + params["tiempo_descarga_aeropuerto_min"],
        unit="m"
    )

    return {
        "hora_entrega_estimada": tiempo_actual,
        "distancia_total_km": distancia_total,
        "tiempo_total_ruta_min": tiempo_total,
        "num_paradas": len(df_orden),
        "maletas_ruta": df_orden["maletas"].sum()
    }

def construir_ruta_factible(pendientes, hora_inicio_ruta, params=PARAMS): #Diseña que reservas se
añade para una ruta.
    if pendientes.empty:
        return None, None, None

    # Establece que reservas son candidatas según el horario que hemos establecido
    fin_ventana = hora_inicio_ruta + pd.to_timedelta(params["ventana_recogida_ruta_min"], unit="m")
    elegibles = pendientes[pendientes["hora_recogida"] <= fin_ventana].copy()

    if elegibles.empty:
        return None, None, "sin_reservas_listas"

    #Prioriza reserva  que deba de llegar más urgente al aeropuerto
    elegibles = elegibles.sort_values(["hora_objetivo_aeropuerto", "hora_recogida"])
    idx_inicial = elegibles.index[0]
    df_inicial = pendientes.loc[[idx_inicial]].copy()

    #Observa si tiene sentido coger la reserva
    ruta_inicial = estimar_ruta_consolidada(df_inicial, hora_inicio_ruta, params)
    if ruta_inicial["hora_entrega_estimada"] > df_inicial["hora_objetivo_aeropuerto"].iloc[0]:
        return None, idx_inicial, "deadline_imposible_con_flota"

    indices = [idx_inicial]
    carga = int(df_inicial["maletas"].sum())

    # Viabilidad de añadir más maletas a la ruta, conociendo las limitaciones
    for idx, row in elegibles.drop(index=idx_inicial).iterrows():
        if len(indices) >= params["max_paradas_por_ruta"]:
            break

        nueva_carga = carga + int(row["maletas"])
        if nueva_carga > params["capacidad_furgoneta_maletas"]:
            continue

        df_prueba = pendientes.loc[indices + [idx]].copy()
        ruta_prueba = estimar_ruta_consolidada(df_prueba, hora_inicio_ruta, params)

        if (ruta_prueba["hora_entrega_estimada"] <= df_prueba["hora_objetivo_aeropuerto"]).all():
            indices.append(idx)
            carga = nueva_carga

    return pendientes.loc[indices].copy(), None, None

# BLOQUE 12. Escenario 1: simulación de flota limitada

def simular_escenario_flota_limitada(df_demanda, num_furgonetas, params=PARAMS): #Simula el
escenario 1, conocidas las restricciones que este supone y estudia como funcionaria el sistema con
2-4 furgonetas. Se establece este rango por viabilidad del primer año de la empresa.
    pendientes = df_demanda.copy().sort_values(
        ["hora_objetivo_aeropuerto", "hora_recogida"]
    ).reset_index(drop=True)

    hora_inicio = pendientes["hora_recogida"].min()
    disponibilidad = {i: hora_inicio for i in range(1, num_furgonetas + 1)}

    rutas = []
    atendidas = []
    rechazadas = []
    ruta_id = 1

    while len(pendientes) > 0:
        furgoneta_id = min(disponibilidad, key=disponibilidad.get)
        hora_inicio_ruta = max(disponibilidad[furgoneta_id], pendientes["hora_recogida"].min())
```

```
        df_ruta, idx_rechazo, motivo = construir_ruta_factible(pendientes, hora_inicio_ruta, params)

        if motivo == "sin_reservas_listas":
            disponibilidad[furgoneta_id] = pendientes["hora_recogida"].min()
            continue

        if idx_rechazo is not None:
            rechazo = pendientes.loc[[idx_rechazo]].copy()
            rechazo["estado"] = "rechazada"
            rechazo["motivo_rechazo"] = motivo
            rechazadas.append(rechazo)
            pendientes = pendientes.drop(index=idx_rechazo).copy()
            continue

        ruta = estimar_ruta_consolidada(df_ruta, hora_inicio_ruta, params)
        hora_entrega = ruta["hora_entrega_estimada"]
        hora_fin = hora_entrega + pd.to_timedelta(params["tiempo_retorno_ciudad_min"], unit="m")
        disponibilidad[furgoneta_id] = hora_fin

        coste_ruta = calcular_coste_viaje(
            distancia_km=ruta["distancia_total_km"],
            duracion_min=ruta["tiempo_total_ruta_min"],
            params=params,
            num_maletas=ruta["maletas_ruta"],
            num_paradas=ruta["num_paradas"],
            factor_retorno=params["factor_retorno_directo"]
        )

        ocupacion = ruta["maletas_ruta"] / params["capacidad_furgoneta_maletas"]

        rutas.append({
            "ruta_id": ruta_id,
            "furgoneta_id": furgoneta_id,
            "hora_inicio_ruta": hora_inicio_ruta,
            "hora_entrega_estimada": hora_entrega,
            "hora_fin_operativa": hora_fin,
            "num_reservas": len(df_ruta),
            "maletas_ruta": ruta["maletas_ruta"],
            "num_paradas": ruta["num_paradas"],
            "distancia_total_km": ruta["distancia_total_km"],
            "tiempo_total_ruta_min": ruta["tiempo_total_ruta_min"],
            "coste_ruta": coste_ruta,
            "ocupacion_ruta": ocupacion
        })

        df_ruta = df_ruta.copy()
        df_ruta["ruta_id"] = ruta_id
        df_ruta["furgoneta_id"] = furgoneta_id
        df_ruta["hora_entrega_estimada"] = hora_entrega
        df_ruta["hora_fin_operativa"] = hora_fin
        df_ruta["coste_ruta"] = coste_ruta
        df_ruta["ocupacion_ruta"] = ocupacion
        atendidas.append(df_ruta)

        pendientes = pendientes.drop(index=df_ruta.index).copy()
        ruta_id += 1

    df_atendidas = pd.concat(atendidas, ignore_index=True) if atendidas else pd.DataFrame()
    df_rutas = pd.DataFrame(rutas)
    df_rechazadas = pd.concat(rechazadas, ignore_index=True) if rechazadas else pd.DataFrame()

    reservas_totales = len(df_demanda)
    reservas_atendidas = len(df_atendidas)
    reservas_rechazadas = len(df_rechazadas)

    if reservas_atendidas > 0:
        df_atendidas["entregada_en_plazo"] = df_atendidas["hora_entrega_estimada"] <=
df_atendidas["hora_objetivo_aeropuerto"]
        df_atendidas["estado"] = np.where(df_atendidas["entregada_en_plazo"], "entregada_en_plazo",
"fuera_de_plazo")
        maletas_ruta = df_atendidas.groupby("ruta_id")["maletas"].transform("sum")
        df_atendidas["coste_reserva"] = df_atendidas["coste_ruta"] * df_atendidas["maletas"] /
maletas_ruta
        reservas_en_plazo = df_atendidas["entregada_en_plazo"].sum()
    else:
        reservas_en_plazo = 0

    coste_variable = df_rutas["coste_ruta"].sum() if len(df_rutas) > 0 else 0
    coste_flota = calcular_coste_disponibilidad_flota(num_furgonetas, params)
    coste_total = coste_variable + coste_flota

    kpis = {
        "escenario": f"E1_Flota_limitada_{num_furgonetas}_vehiculos",
        "reservas_totales": reservas_totales,
        "reservas_atendidas": reservas_atendidas,
```

```
        "reservas_rechazadas": reservas_rechazadas,
        "reservas_desbordadas_almacen": 0,
        "tasa_atencion": reservas_atendidas / reservas_totales if reservas_totales > 0 else np.nan,
        "nivel_servicio_total": reservas_en_plazo / reservas_totales if reservas_totales > 0 else
np.nan,
        "nivel_servicio_atendidas": reservas_en_plazo / reservas_atendidas if reservas_atendidas > 0
else np.nan,
        "maletas_totales": df_demanda["maletas"].sum(),
        "viajes_aeropuerto": len(df_rutas),
        "ocupacion_media": df_rutas["ocupacion_ruta"].mean() if len(df_rutas) > 0 else 0,
        "num_vehiculos_operativos": num_furgonetas,
        "coste_variable_total": coste_variable,
        "coste_disponibilidad_flota": coste_flota,
        "coste_total": coste_total,
        "coste_medio_reserva_atendida": coste_total / reservas_atendidas if reservas_atendidas > 0
else np.nan,
        "coste_medio_reserva_total": coste_total / reservas_totales if reservas_totales > 0 else
np.nan
    }

    return df_atendidas, df_rutas, df_rechazadas, pd.DataFrame([kpis])

# BLOQUE 13. Escenario 2: definición de hubs y rutas hub-aeropuerto

#Establecemos tres puntos de recogida: 1 en el centro de Madrid (mayor densidad turística), 1 en el
Barrio Salamanca y uno por atocha (cerca de la estación de tren). La restricción de maletas viene de
buscar locales pequeños, del tamaño de los locker rooms, consiguiendo asi reducir costes de alquiler
hubs = pd.DataFrame([
    {
        "hub_id": 1,
        "hub_nombre": "Hub Centro",
        "lat": 40.4168,
        "lon": -3.7038,
        "capacidad_maletas": 60
    },
    {
        "hub_id": 2,
        "hub_nombre": "Hub Atocha-Sur",
        "lat": 40.4073,
        "lon": -3.6909,
        "capacidad_maletas": 40
    },
    {
        "hub_id": 3,
        "hub_nombre": "Hub Salamanca-Chamartin",
        "lat": 40.4467,
        "lon": -3.6780,
        "capacidad_maletas": 40
    }
])

display(hubs)

#Establece los tiempos de distancias de los hubs al aeropuerto según ORS
resultados_hubs = []

for _, hub in hubs.iterrows():

    distancia_km, duracion_min, error = calcular_ruta_ors_requests(
        lat_origen=hub["lat"],
        lon_origen=hub["lon"],
        lat_destino=AEROPUERTO["lat"],
        lon_destino=AEROPUERTO["lon"],
        api_key=ORS_API_KEY
    )

    resultados_hubs.append({
        "hub_id": hub["hub_id"],
        "hub_nombre": hub["hub_nombre"],
        "distancia_hub_aeropuerto_km": distancia_km,
        "tiempo_hub_aeropuerto_min": duracion_min,
        "error_ors": error
    })

df_hubs_rutas = pd.DataFrame(resultados_hubs)

hubs = hubs.merge(df_hubs_rutas, on=["hub_id", "hub_nombre"], how="left")

display(hubs)

# BLOQUE 14. Escenario 2: asignación de reservas a hubs y capacidades

PARAMS.update({
    "tiempo_operativo_llegada_hub_min": 5, #Recepción, Registro y clasificación inical
    "umbral_salida_hub": 0.70 #Para darnos margen y no saturar el hub.
```

```
})

def asignar_hub_mas_cercano(df_demanda, hubs, params=PARAMS):
    registros = []

    for _, reserva in df_demanda.iterrows():
        opciones = []

        for _, hub in hubs.iterrows():
            distancia = distancia_haversine_km(
                reserva["lat"], reserva["lon"],
                hub["lat"], hub["lon"]
            )

            opciones.append({
                "hub_id": hub["hub_id"],
                "hub_nombre": hub["hub_nombre"],
                "distancia_al_hub_km": distancia,
                "tiempo_al_hub_min": 0,
                "capacidad_hub_maletas": hub["capacidad_maletas"],
                "distancia_hub_aeropuerto_km": hub["distancia_hub_aeropuerto_km"],
                "tiempo_hub_aeropuerto_min": hub["tiempo_hub_aeropuerto_min"]
            })

        hub_asignado = min(opciones, key=lambda x: x["distancia_al_hub_km"])

        fila = reserva.to_dict()
        fila.update(hub_asignado)

        # La hora de recogida es el momento en el que el cliente entrega la maleta.
        fila["hora_llegada_hub"] = (
            fila["hora_recogida"]
            + pd.to_timedelta(params["tiempo_operativo_llegada_hub_min"], unit="m")
        )

        registros.append(fila)

    return pd.DataFrame(registros)

def crear_hubs_con_capacidad(hubs_base, capacidad_centro, capacidad_atocha, capacidad_salamanca):
    hubs_mod = hubs_base.copy()
    hubs_mod.loc[hubs_mod["hub_nombre"] == "Hub Centro", "capacidad_maletas"] = capacidad_centro
    hubs_mod.loc[hubs_mod["hub_nombre"] == "Hub Atocha-Sur", "capacidad_maletas"] = capacidad_atocha
    hubs_mod.loc[hubs_mod["hub_nombre"] == "Hub Salamanca-Chamartin", "capacidad_maletas"] =
capacidad_salamanca
    return hubs_mod

hubs_modelo = crear_hubs_con_capacidad(hubs, 60, 40, 40)
display(hubs_modelo)

# BLOQUE 15. Escenario 2: salidas consolidadas desde hubs

def simular_salidas_hub_con_capacidad(
    df_hub,
    params=PARAMS
):

    df_hub = df_hub.sort_values("hora_llegada_hub").reset_index(drop=True)

    capacidad_hub = int(df_hub["capacidad_hub_maletas"].iloc[0])
    hub_id = df_hub["hub_id"].iloc[0]
    hub_nombre = df_hub["hub_nombre"].iloc[0]
    distancia_hub_aeropuerto = df_hub["distancia_hub_aeropuerto_km"].iloc[0]
    tiempo_hub_aeropuerto = df_hub["tiempo_hub_aeropuerto_min"].iloc[0]

    pendientes = df_hub.copy()
    i = 0

    stock = []
    atendidas = []
    rechazadas = []
    viajes = []

    viaje_id = 1
    max_stock_maletas = 0

    if len(pendientes) == 0:
        return pd.DataFrame(), pd.DataFrame(), pd.DataFrame(), {
            "hub_id": hub_id,
            "hub_nombre": hub_nombre,
            "max_stock_maletas": 0,
            "saturacion_maxima": 0
        }

    tiempo_actual = pendientes["hora_llegada_hub"].min()
```

```
    while i < len(pendientes) or len(stock) > 0:

        if len(stock) == 0 and i < len(pendientes):
            tiempo_actual = max(tiempo_actual, pendientes.loc[i, "hora_llegada_hub"])

        while i < len(pendientes) and pendientes.loc[i, "hora_llegada_hub"] <= tiempo_actual:

            fila = pendientes.loc[i].to_dict()
            carga_actual = sum(x["maletas"] for x in stock)

            if carga_actual + fila["maletas"] <= capacidad_hub:
                stock.append(fila)
                carga_actual += fila["maletas"]
                max_stock_maletas = max(max_stock_maletas, carga_actual)
            else:
                rechazo = pd.DataFrame([fila])
                rechazo["estado"] = "rechazada"
                rechazo["motivo_rechazo"] = "hub_sin_capacidad"
                rechazadas.append(rechazo)

            i += 1

        if len(stock) == 0:
            continue

        df_stock = pd.DataFrame(stock)

        df_stock["ultima_salida_posible"] = (
            df_stock["hora_objetivo_aeropuerto"]
            - pd.to_timedelta(tiempo_hub_aeropuerto, unit="m")
            - pd.to_timedelta(params["tiempo_descarga_aeropuerto_min"], unit="m")
        )

        hora_limite_mas_urgente = df_stock["ultima_salida_posible"].min()
        carga_stock = df_stock["maletas"].sum()

        condicion_salida = (
            carga_stock >= params["capacidad_furgoneta_maletas"]
            or tiempo_actual >= hora_limite_mas_urgente
            or carga_stock >= capacidad_hub * params["umbral_salida_hub"]
        )

        if not condicion_salida:
            siguiente_llegada = (
                pendientes.loc[i, "hora_llegada_hub"]
                if i < len(pendientes)
                else pd.Timestamp.max
            )

            tiempo_actual = min(siguiente_llegada, hora_limite_mas_urgente)
            continue

        df_stock = df_stock.sort_values(["hora_objetivo_aeropuerto", "hora_llegada_hub"])

        seleccionadas = []
        carga_viaje = 0

        for _, row in df_stock.iterrows():

            if carga_viaje + row["maletas"] <= params["capacidad_furgoneta_maletas"]:
                hora_entrega_estimada = (
                    tiempo_actual
                    + pd.to_timedelta(tiempo_hub_aeropuerto, unit="m")
                    + pd.to_timedelta(params["tiempo_descarga_aeropuerto_min"], unit="m")
                )

                if hora_entrega_estimada <= row["hora_objetivo_aeropuerto"]:
                    seleccionadas.append(row["reserva_id"])
                    carga_viaje += row["maletas"]

        if len(seleccionadas) == 0:
            reserva_rechazada = df_stock.iloc[[0]].copy()
            reserva_rechazada["estado"] = "rechazada"
            reserva_rechazada["motivo_rechazo"] = "no_sale_a_tiempo_desde_hub"
            rechazadas.append(reserva_rechazada)

            stock = [
                x for x in stock
                if x["reserva_id"] != reserva_rechazada["reserva_id"].iloc[0]
            ]
            continue

        df_viaje = df_stock[df_stock["reserva_id"].isin(seleccionadas)].copy()
```

```
        hora_entrega_estimada = (
            tiempo_actual
            + pd.to_timedelta(tiempo_hub_aeropuerto, unit="m")
            + pd.to_timedelta(params["tiempo_descarga_aeropuerto_min"], unit="m")
        )

        coste_linehaul = calcular_coste_viaje(
            distancia_km=distancia_hub_aeropuerto,
            duracion_min=tiempo_hub_aeropuerto + params["tiempo_descarga_aeropuerto_min"],
            params=params,
            num_maletas=df_viaje["maletas"].sum(),
            num_paradas=0,
            factor_retorno=params["factor_retorno_hub"],
            pasa_por_hub=True,
            incluye_entrega_aeropuerto=True
        )

        coste_transferencia_almacen_total = (
            df_viaje["coste_transferencia_almacen"].sum()
            if "coste_transferencia_almacen" in df_viaje.columns
            else 0
        )

        coste_total_viaje = coste_linehaul + coste_transferencia_almacen_total

        viajes.append({
            "viaje_id": viaje_id,
            "hub_id": hub_id,
            "hub_nombre": hub_nombre,
            "hora_salida_hub": tiempo_actual,
            "hora_entrega_estimada": hora_entrega_estimada,
            "num_reservas": len(df_viaje),
            "maletas_viaje": df_viaje["maletas"].sum(),
            "distancia_hub_aeropuerto_km": distancia_hub_aeropuerto,
            "tiempo_hub_aeropuerto_min": tiempo_hub_aeropuerto,
            "coste_transferencia_almacen": coste_transferencia_almacen_total,
            "coste_viaje": coste_total_viaje,
            "ocupacion_viaje": df_viaje["maletas"].sum() / params["capacidad_furgoneta_maletas"]
        })

        df_viaje["viaje_id"] = viaje_id
        df_viaje["hora_salida_hub"] = tiempo_actual
        df_viaje["hora_entrega_estimada"] = hora_entrega_estimada
        df_viaje["estado"] = "entregada_en_plazo"
        df_viaje["coste_viaje"] = coste_total_viaje

        atendidas.append(df_viaje)

        ids_seleccionadas = set(seleccionadas)
        stock = [
            x for x in stock
            if x["reserva_id"] not in ids_seleccionadas
        ]

        viaje_id += 1

    df_atendidas = pd.concat(atendidas, ignore_index=True) if atendidas else pd.DataFrame()
    df_viajes = pd.DataFrame(viajes)
    df_rechazadas = pd.concat(rechazadas, ignore_index=True) if rechazadas else pd.DataFrame()

    resumen_stock = {
        "hub_id": hub_id,
        "hub_nombre": hub_nombre,
        "max_stock_maletas": max_stock_maletas,
        "saturacion_maxima": max_stock_maletas / capacidad_hub
    }

    return df_atendidas, df_viajes, df_rechazadas, resumen_stock

# BLOQUE 16. Escenario 2: simulación completa de hubs urbanos

#Simula el escenario de los hubs. Podemos jugar con las capacidades y ver si afectan al desarrollo
del modelo.
def simular_escenario_hubs_capacidad(
    df_demanda,
    hubs,
    params=PARAMS,
    nombre_escenario="E2_Hubs_capacidad_limitada"
):

    df_demanda_hubs = asignar_hub_mas_cercano(
        df_demanda=df_demanda,
        hubs=hubs,
        params=params
    )
```

```
    atendidas_total = []
    viajes_total = []
    rechazos_total = []
    resumen_stock_total = []

    for hub_id, df_hub in df_demanda_hubs.groupby("hub_id"):

        df_atendidas_hub, df_viajes_hub, df_rechazadas_hub, resumen_stock =
simular_salidas_hub_con_capacidad(
            df_hub=df_hub,
            params=params
        )

        if len(df_atendidas_hub) > 0:
            atendidas_total.append(df_atendidas_hub)

        if len(df_viajes_hub) > 0:
            viajes_total.append(df_viajes_hub)

        if len(df_rechazadas_hub) > 0:
            rechazos_total.append(df_rechazadas_hub)

        resumen_stock_total.append(resumen_stock)

    df_atendidas = pd.concat(atendidas_total, ignore_index=True) if atendidas_total else
pd.DataFrame()
    df_viajes = pd.concat(viajes_total, ignore_index=True) if viajes_total else pd.DataFrame()
    df_rechazadas = pd.concat(rechazos_total, ignore_index=True) if rechazos_total else
pd.DataFrame()
    df_stock = pd.DataFrame(resumen_stock_total)

    reservas_totales = len(df_demanda)
    reservas_atendidas = len(df_atendidas)
    reservas_rechazadas = len(df_rechazadas)

    coste_variable_total = df_viajes["coste_viaje"].sum() if len(df_viajes) > 0 else 0

    num_vehiculos_operativos = len(hubs)

    coste_disponibilidad_flota = calcular_coste_disponibilidad_flota(num_vehiculos_operativos,
params)

    coste_total = coste_variable_total + coste_disponibilidad_flota

    kpis = {
        "escenario": nombre_escenario,
        "reservas_totales": reservas_totales,
        "reservas_atendidas": reservas_atendidas,
        "reservas_rechazadas": reservas_rechazadas,
        "tasa_atencion": reservas_atendidas / reservas_totales,
        "nivel_servicio_total": reservas_atendidas / reservas_totales,
        "nivel_servicio_atendidas": 1.0 if reservas_atendidas > 0 else 0,
        "maletas_totales": df_demanda["maletas"].sum(),
        "viajes_aeropuerto": len(df_viajes),
        "ocupacion_media": df_viajes["ocupacion_viaje"].mean() if len(df_viajes) > 0 else 0,
        "coste_total": coste_total,
        "coste_medio_reserva_atendida": coste_total / reservas_atendidas if reservas_atendidas > 0
else np.nan,
        "coste_medio_reserva_total": coste_total / reservas_totales if reservas_totales > 0 else
np.nan,
        "num_vehiculos_operativos": num_vehiculos_operativos,
        "coste_variable_total": coste_variable_total,
        "coste_disponibilidad_flota": coste_disponibilidad_flota,
        "saturacion_maxima_hubs": df_stock["saturacion_maxima"].max() if len(df_stock) > 0 else 0
    }

    return df_atendidas, df_viajes, df_rechazadas, pd.DataFrame([kpis]), df_stock, df_demanda_hubs

# BLOQUE 17. Escenario 3: almacén central y distancias hub-almacén

#Con esto, podemos plantear un sistema mixto de hubs con almacenamiento centralizado y comprobar si
este escenario operativo mejora las métricas del segundo.
almacen_central = {
    "hub_id": 99,
    "hub_nombre": "Almacen central Coslada-Barajas",
    "lat": 40.4500,
    "lon": -3.5600,
    "capacidad_maletas": 500
}

distancia_almacen_aeropuerto_km, tiempo_almacen_aeropuerto_min, error_almacen =
calcular_ruta_ors_requests(
    lat_origen=almacen_central["lat"],
    lon_origen=almacen_central["lon"],
```

```
    lat_destino=AEROPUERTO["lat"],
    lon_destino=AEROPUERTO["lon"],
    api_key=ORS_API_KEY
)

almacen_central["distancia_hub_aeropuerto_km"] = distancia_almacen_aeropuerto_km
almacen_central["tiempo_hub_aeropuerto_min"] = tiempo_almacen_aeropuerto_min
almacen_central["error_ors"] = error_almacen

almacen_central

#Calculamos las distancias de los hubs al almacen, según ORS
def preparar_hubs_para_almacen(
    hubs_config,
    almacen_central,
    api_key
):
    hubs_e3 = hubs_config.copy()
    distancias_almacen = []

    for _, hub in hubs_e3.iterrows():

        distancia_km, duracion_min, error = calcular_ruta_ors_requests(
            lat_origen=hub["lat"],
            lon_origen=hub["lon"],
            lat_destino=almacen_central["lat"],
            lon_destino=almacen_central["lon"],
            api_key=api_key
        )

        distancias_almacen.append({
            "hub_id": hub["hub_id"],
            "distancia_hub_almacen_km": distancia_km,
            "tiempo_hub_almacen_min": duracion_min,
            "error_hub_almacen": error
        })

    df_hub_almacen = pd.DataFrame(distancias_almacen)

    hubs_e3 = hubs_e3.merge(
        df_hub_almacen,
        on="hub_id",
        how="left"
    )

    return hubs_e3

hubs_e3 = preparar_hubs_para_almacen(
    hubs_config=hubs_modelo,
    almacen_central=almacen_central,
    api_key=ORS_API_KEY
)

display(hubs_e3)

# BLOQUE 18. Escenario 3: salidas desde hubs con desbordamiento

def simular_salidas_hub_con_desborde(
    df_hub,
    almacen_central,
    params=PARAMS
):

    df_hub = df_hub.sort_values("hora_llegada_hub").reset_index(drop=True)

    if len(df_hub) == 0:
        return (
            pd.DataFrame(),
            pd.DataFrame(),
            pd.DataFrame(),
            pd.DataFrame(),
            {
                "hub_id": None,
                "hub_nombre": None,
                "max_stock_maletas": 0,
                "saturacion_maxima": 0
            }
        )

    capacidad_hub = int(df_hub["capacidad_hub_maletas"].iloc[0])
    hub_id = df_hub["hub_id"].iloc[0]
    hub_nombre = df_hub["hub_nombre"].iloc[0]

    distancia_hub_almacen = df_hub["distancia_hub_almacen_km"].iloc[0]
    tiempo_hub_almacen = df_hub["tiempo_hub_almacen_min"].iloc[0]
```

```
    reservas_hub = []
    reservas_desbordadas = []

    carga_acumulada_hub = 0
    max_stock_maletas = 0

    for _, row in df_hub.iterrows():

        fila = row.to_dict()
        maletas_reserva = fila["maletas"]

        # Si la reserva cabe en el hub, se queda en el hub
        if carga_acumulada_hub + maletas_reserva <= capacidad_hub:

            reservas_hub.append(fila)

            carga_acumulada_hub += maletas_reserva
            max_stock_maletas = max(max_stock_maletas, carga_acumulada_hub)

        # Si no cabe, se deriva al almacén central, añadiendo costes extras.
        else:

            fila["hub_id_original"] = fila["hub_id"]
            fila["hub_nombre_original"] = fila["hub_nombre"]

            fila["hub_id"] = almacen_central["hub_id"]
            fila["hub_nombre"] = almacen_central["hub_nombre"]
            fila["capacidad_hub_maletas"] = almacen_central["capacidad_maletas"]
            fila["distancia_hub_aeropuerto_km"] = almacen_central["distancia_hub_aeropuerto_km"]
            fila["tiempo_hub_aeropuerto_min"] = almacen_central["tiempo_hub_aeropuerto_min"]

            fila["hora_llegada_hub"] = (
                fila["hora_llegada_hub"]
                + pd.to_timedelta(tiempo_hub_almacen, unit="m")
                + pd.to_timedelta(10, unit="m")
            )

            coste_transferencia = calcular_coste_viaje(
                distancia_km=distancia_hub_almacen,
                duracion_min=tiempo_hub_almacen,
                params=params,
                num_maletas=fila["maletas"],
                num_paradas=0,
                factor_retorno=params["factor_retorno_almacen"],
                pasa_por_hub=False,
                incluye_entrega_aeropuerto=False
            )

            fila["coste_transferencia_almacen"] = coste_transferencia
            fila["estado_desborde"] = "derivada_a_almacen"

            reservas_desbordadas.append(fila)

    df_reservas_hub = pd.DataFrame(reservas_hub)
    df_desbordadas = pd.DataFrame(reservas_desbordadas)

    # Simulamos las salidas al aeropuerto solo con las reservas que sí caben en el hub
    if len(df_reservas_hub) > 0:

        df_atendidas, df_viajes, df_rechazadas, resumen_stock = simular_salidas_hub_con_capacidad(
            df_hub=df_reservas_hub,
            params=params
        )

    else:

        df_atendidas = pd.DataFrame()
        df_viajes = pd.DataFrame()
        df_rechazadas = pd.DataFrame()

        resumen_stock = {
            "hub_id": hub_id,
            "hub_nombre": hub_nombre,
            "max_stock_maletas": 0,
            "saturacion_maxima": 0
        }

    # Añadimos una medida sencilla de saturación máxima del hub
    resumen_stock["max_stock_maletas"] = max_stock_maletas
    resumen_stock["saturacion_maxima"] = max_stock_maletas / capacidad_hub if capacidad_hub > 0 else
0

    return df_atendidas, df_viajes, df_rechazadas, df_desbordadas, resumen_stock
```

```
# BLOQUE 19. Escenario 3: simulación completa con almacén central

def simular_escenario_hubs_almacen(
    df_demanda,
    hubs_e3,
    almacen_central,
    params=PARAMS,
    nombre_escenario="E3_Hubs_con_almacen"
):

    df_demanda_hubs = asignar_hub_mas_cercano(
        df_demanda=df_demanda,
        hubs=hubs_e3,
        params=params
    )

    df_demanda_hubs = df_demanda_hubs.merge(
        hubs_e3[[
            "hub_id",
            "distancia_hub_almacen_km",
            "tiempo_hub_almacen_min"
        ]],
        on="hub_id",
        how="left"
    )

    atendidas_total = []
    viajes_total = []
    rechazos_total = []
    desbordadas_total = []
    resumen_stock_total = []

    for hub_id, df_hub in df_demanda_hubs.groupby("hub_id"):

        df_atendidas_hub, df_viajes_hub, df_rechazadas_hub, df_desbordadas_hub, resumen_stock =
simular_salidas_hub_con_desborde(
            df_hub=df_hub,
            almacen_central=almacen_central,
            params=params
        )

        if len(df_atendidas_hub) > 0:
            atendidas_total.append(df_atendidas_hub)

        if len(df_viajes_hub) > 0:
            df_viajes_hub["origen_operativo"] = "hub"
            viajes_total.append(df_viajes_hub)

        if len(df_rechazadas_hub) > 0:
            rechazos_total.append(df_rechazadas_hub)

        if len(df_desbordadas_hub) > 0:
            desbordadas_total.append(df_desbordadas_hub)

        resumen_stock_total.append(resumen_stock)

    if len(desbordadas_total) > 0:
        df_desbordadas = pd.concat(desbordadas_total, ignore_index=True)
    else:
        df_desbordadas = pd.DataFrame()

    if len(df_desbordadas) > 0:

        df_atendidas_almacen, df_viajes_almacen, df_rechazos_almacen, resumen_stock_almacen =
simular_salidas_hub_con_capacidad(
            df_hub=df_desbordadas,
            params=params
        )

        if len(df_atendidas_almacen) > 0:
            atendidas_total.append(df_atendidas_almacen)

        if len(df_viajes_almacen) > 0:
            df_viajes_almacen["origen_operativo"] = "almacen_central"
            viajes_total.append(df_viajes_almacen)

        if len(df_rechazos_almacen) > 0:
            rechazos_total.append(df_rechazos_almacen)

        resumen_stock_total.append(resumen_stock_almacen)

    if len(atendidas_total) > 0:
        df_atendidas = pd.concat(atendidas_total, ignore_index=True)
    else:
        df_atendidas = pd.DataFrame()
```

```
    if len(viajes_total) > 0:
        df_viajes = pd.concat(viajes_total, ignore_index=True)
    else:
        df_viajes = pd.DataFrame()

    if len(rechazos_total) > 0:
        df_rechazadas = pd.concat(rechazos_total, ignore_index=True)
    else:
        df_rechazadas = pd.DataFrame()

    df_stock = pd.DataFrame(resumen_stock_total)

    reservas_totales = len(df_demanda)
    reservas_atendidas = len(df_atendidas)
    reservas_rechazadas = len(df_rechazadas)
    reservas_desbordadas = len(df_desbordadas)

    coste_variable_total = df_viajes["coste_viaje"].sum() if len(df_viajes) > 0 else 0

    num_vehiculos_operativos = len(hubs_e3) + 1

    coste_disponibilidad_flota = calcular_coste_disponibilidad_flota(num_vehiculos_operativos,
params)

    coste_total = coste_variable_total + coste_disponibilidad_flota

    kpis = {
        "escenario": nombre_escenario,
        "reservas_totales": reservas_totales,
        "reservas_atendidas": reservas_atendidas,
        "reservas_rechazadas": reservas_rechazadas,
        "reservas_desbordadas_almacen": reservas_desbordadas,
        "tasa_atencion": reservas_atendidas / reservas_totales if reservas_totales > 0 else 0,
        "nivel_servicio_total": reservas_atendidas / reservas_totales if reservas_totales > 0 else
0,
        "nivel_servicio_atendidas": 1.0 if reservas_atendidas > 0 else 0,
        "maletas_totales": df_demanda["maletas"].sum(),
        "viajes_aeropuerto": len(df_viajes),
        "ocupacion_media": df_viajes["ocupacion_viaje"].mean() if len(df_viajes) > 0 else 0,
        "coste_total": coste_total,
        "coste_medio_reserva_atendida": coste_total / reservas_atendidas if reservas_atendidas > 0
else np.nan,
        "coste_medio_reserva_total": coste_total / reservas_totales if reservas_totales > 0 else
np.nan,
        "num_vehiculos_operativos": num_vehiculos_operativos,
        "coste_variable_total": coste_variable_total,
        "coste_disponibilidad_flota": coste_disponibilidad_flota,
        "saturacion_maxima_hubs": df_stock["saturacion_maxima"].max() if len(df_stock) > 0 else 0
    }

    return df_atendidas, df_viajes, df_rechazadas, df_desbordadas, pd.DataFrame([kpis]), df_stock

# BLOQUE 20. Comparación final, KPIs y ejecución de escenarios

#Comparamos los 3 escenarios observado.
NUM_FURGONETAS_E1 = 3

COLUMNAS_KPI = [
    "escenario",
    "reservas_totales",
    "reservas_atendidas",
    "reservas_rechazadas",
    "reservas_desbordadas_almacen",
    "tasa_atencion",
    "nivel_servicio_total",
    "nivel_servicio_atendidas",
    "maletas_totales",
    "viajes_aeropuerto",
    "ocupacion_media",
    "num_vehiculos_operativos",
    "coste_variable_total",
    "coste_disponibilidad_flota",
    "coste_total",
    "coste_medio_reserva_atendida",
    "coste_medio_reserva_total"
]

def normalizar_kpis(df):
    df = df.copy()
    for col in COLUMNAS_KPI:
        if col not in df.columns:
            df[col] = 0
    return df[COLUMNAS_KPI]
```

```
def ejecutar_escenarios_dia(df_demanda, params=PARAMS):
    res_e0, kpi_e0 = simular_escenario_directo(df_demanda, params)

    res_e1, rutas_e1, rechazos_e1, kpi_e1 = simular_escenario_flota_limitada(
        df_demanda=df_demanda,
        num_furgonetas=NUM_FURGONETAS_E1,
        params=params
    )

    res_e2, viajes_e2, rechazos_e2, kpi_e2, stock_e2, demanda_hubs_e2 =
simular_escenario_hubs_capacidad(
        df_demanda=df_demanda,
        hubs=hubs_modelo,
        params=params,
        nombre_escenario="E2_Hubs_capacidad_limitada"
    )

    res_e3, viajes_e3, rechazos_e3, desbordadas_e3, kpi_e3, stock_e3 =
simular_escenario_hubs_almacen(
        df_demanda=df_demanda,
        hubs_e3=hubs_e3,
        almacen_central=almacen_central,
        params=params,
        nombre_escenario="E3_Hubs_con_almacen"
    )

    detalle = {
        "E0": res_e0,
        "E1": res_e1,
        "E2": res_e2,
        "E3": res_e3,
        "rutas_E1": rutas_e1,
        "viajes_E2": viajes_e2,
        "viajes_E3": viajes_e3,
        "rechazos_E1": rechazos_e1,
        "rechazos_E2": rechazos_e2,
        "rechazos_E3": rechazos_e3,
        "desbordadas_E3": desbordadas_e3,
        "stock_E2": stock_e2,
        "stock_E3": stock_e3,
        "demanda_hubs_E2": demanda_hubs_e2
    }

    comparacion = pd.concat(
        [
            normalizar_kpis(kpi_e0),
            normalizar_kpis(kpi_e1),
            normalizar_kpis(kpi_e2),
            normalizar_kpis(kpi_e3)
        ],
        ignore_index=True
    )

    return comparacion, detalle

df_comparacion_final, detalle_dia_base = ejecutar_escenarios_dia(df_demanda_test, PARAMS)

cols_pct = ["tasa_atencion", "nivel_servicio_total", "nivel_servicio_atendidas", "ocupacion_media"]
cols_coste = ["coste_variable_total", "coste_disponibilidad_flota", "coste_total",
"coste_medio_reserva_atendida", "coste_medio_reserva_total"]

df_comparacion_final[cols_pct] = df_comparacion_final[cols_pct].round(4)
df_comparacion_final[cols_coste] = df_comparacion_final[cols_coste].round(2)

display(df_comparacion_final)

# BLOQUE 21. Simulación Monte Carlo y resumen de resultados

def ejecutar_montecarlo(df_base, n_iteraciones, demanda_media, seed_inicial=1000, params=PARAMS):
    resultados = []

    print(f"Iniciando Monte Carlo con {n_iteraciones} iteraciones...")

    for i in range(n_iteraciones):
        seed = seed_inicial + i

        df_demanda = generar_demanda_diaria(df_base, seed, params, demanda_media)
        comparacion, _ = ejecutar_escenarios_dia(df_demanda, params)

        comparacion["seed"] = seed
        comparacion["iteracion"] = i + 1

        resultados.append(comparacion)

        # Log cada 10 iteraciones
```

```
        if (i + 1) % 10 == 0:
            print(f"Iteración {i + 1}/{n_iteraciones} completada")

    print("Monte Carlo finalizado.")

    return pd.concat(resultados, ignore_index=True)

df_montecarlo = ejecutar_montecarlo(
    df_base=df_base,
    n_iteraciones=PARAMS["num_iteraciones_montecarlo"],
    demanda_media=PARAMS["demanda_media_reservas"],
    seed_inicial=1000,
    params=PARAMS
)

df_resumen_montecarlo = (
    df_montecarlo
    .groupby("escenario")
    .agg(
        reservas_medias=("reservas_totales", "mean"),
        tasa_atencion_media=("tasa_atencion", "mean"),
        nivel_servicio_medio=("nivel_servicio_total", "mean"),
        viajes_medios=("viajes_aeropuerto", "mean"),
        ocupacion_media=("ocupacion_media", "mean"),
        coste_total_medio=("coste_total", "mean"),
        coste_medio_reserva=("coste_medio_reserva_total", "mean"),
        coste_medio_p5=("coste_medio_reserva_total", lambda x: x.quantile(0.05)),
        coste_medio_p95=("coste_medio_reserva_total", lambda x: x.quantile(0.95))
    )
    .reset_index()
    .round(3)
)

display(df_resumen_montecarlo)
```

7.2 Anexo 2: Obtención de datos con Overpass turbo.

Información extraída de este link: https://overpass-turbo.eu/

```
[out:json][timeout:90];

area
  ["name"="Madrid"]
  ["boundary"="administrative"]
  ["admin_level"="8"]
  ->.searchArea;

(
  node["tourism"~"hotel|hostel|guest_house|apartment|motel"](area.searchArea);
  way["tourism"~"hotel|hostel|guest_house|apartment|motel"](area.searchArea);
  relation["tourism"~"hotel|hostel|guest_house|apartment|motel"](area.searchArea);

  node["building"="hotel"](area.searchArea);
  way["building"="hotel"](area.searchArea);
  relation["building"="hotel"](area.searchArea);
);

out center tags;
```

7.3 Anexo 3: análisis de sensibilidad de capacidad en el escenario 2

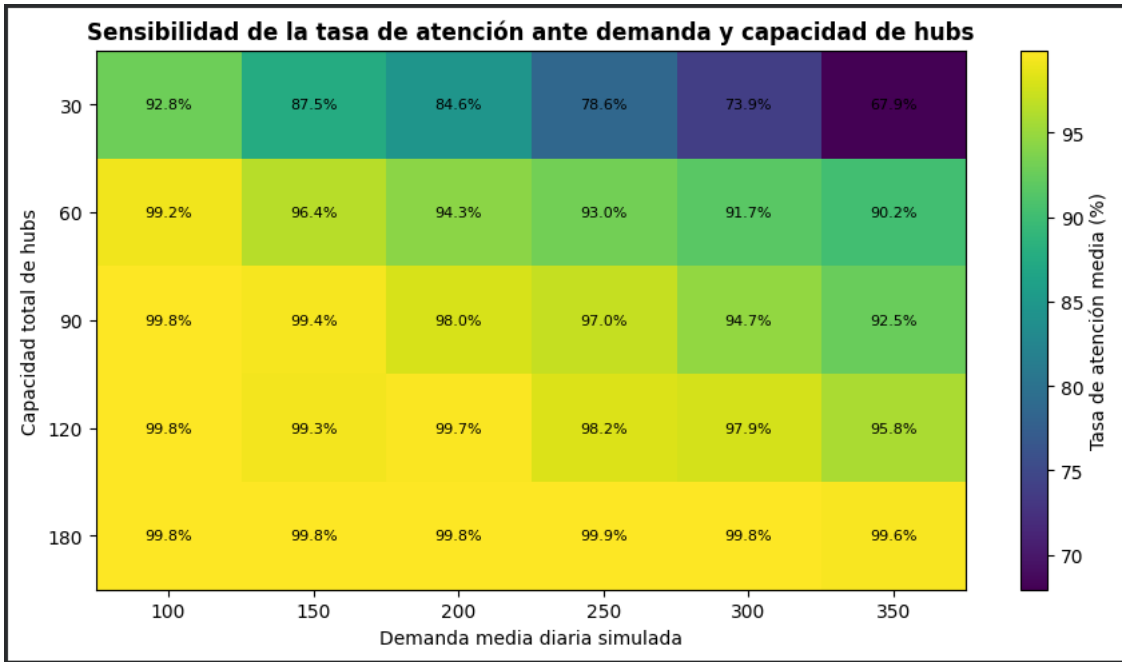


Figura 5. Sensibilidad de la tasa de atención ante distintos niveles de demanda media diaria y capacidad total de hubs. Fuente: elaboración propia a partir del modelo de simulación.

```
def crear_hubs_por_capacidad_total(hubs_base, capacidad_total):

    capacidad_centro = round(capacidad_total * 3 / 7)
    capacidad_atocha = round(capacidad_total * 2 / 7)
    capacidad_salamanca = capacidad_total - capacidad_centro - capacidad_atocha

    return crear_hubs_con_capacidad(
        hubs_base=hubs_base,
        capacidad_centro=capacidad_centro,
        capacidad_atocha=capacidad_atocha,
        capacidad_salamanca=capacidad_salamanca
    )

def sensibilidad_demanda_capacidad_hubs(
    df_base,
    hubs_base,
    demandas_medias,
    capacidades_totales,
    n_iteraciones=20,
    seed_inicial=3000,
    params=PARAMS
):

    resultados = []

    for capacidad_total in capacidades_totales:

        hubs_sensibilidad = crear_hubs_por_capacidad_total(
            hubs_base=hubs_base,
            capacidad_total=capacidad_total
        )

        for demanda_media in demandas_medias:

            tasas = []

            print(
                f"Ejecutando sensibilidad | "
                f"Capacidad total hubs: {capacidad_total} | "
                f"Demanda media: {demanda_media}"
            )

            for i in range(n_iteraciones):

                seed = seed_inicial + i

                df_demanda = generar_demanda_diaria(
```

```
                    df_base=df_base,
                    seed=seed,
                    params=params,
                    demanda_media=demanda_media
                )

                _, _, _, kpis_e2, _, _ = simular_escenario_hubs_capacidad(
                    df_demanda=df_demanda,
                    hubs=hubs_sensibilidad,
                    params=params,
                    nombre_escenario="E2_Hubs_sensibilidad"
                )

                tasas.append(kpis_e2["tasa_atencion"].iloc[0])

            resultados.append({
                "capacidad_total_hubs": capacidad_total,
                "demanda_media": demanda_media,
                "tasa_atencion_media": np.mean(tasas)
            })

    return pd.DataFrame(resultados)


# Valores a analizar
demandas_medias = [100, 150, 200, 250, 300, 350]
capacidades_totales = [30, 60, 90, 120, 180]

df_sensibilidad_hubs = sensibilidad_demanda_capacidad_hubs(
    df_base=df_base,
    hubs_base=hubs,
    demandas_medias=demandas_medias,
    capacidades_totales=capacidades_totales,
    n_iteraciones=20,
    seed_inicial=3000,
    params=PARAMS
)

display(df_sensibilidad_hubs)

# Crear matriz para el heatmap
matriz_heatmap = df_sensibilidad_hubs.pivot(
    index="capacidad_total_hubs",
    columns="demanda_media",
    values="tasa_atencion_media"
)

# Convertir a porcentaje
matriz_pct = matriz_heatmap * 100

plt.figure(figsize=(9, 5))

plt.imshow(
    matriz_pct,
    aspect="auto"
)

plt.title(
    "Sensibilidad de la tasa de atención ante demanda y capacidad de hubs",
    fontsize=12,
    fontweight="bold"
)

plt.xlabel("Demanda media diaria simulada")
plt.ylabel("Capacidad total de hubs")

plt.xticks(
    ticks=np.arange(len(matriz_pct.columns)),
    labels=matriz_pct.columns
)

plt.yticks(
    ticks=np.arange(len(matriz_pct.index)),
    labels=matriz_pct.index
)

# Añadir valores dentro de cada celda
for i in range(len(matriz_pct.index)):
    for j in range(len(matriz_pct.columns)):
        valor = matriz_pct.iloc[i, j]
        plt.text(
            j,
            i,
            f"{valor:.1f}%",
            ha="center",
```

```
            va="center",
            fontsize=8
        )

# Barra lateral
cbar = plt.colorbar()
cbar.set_label("Tasa de atención media (%)")

plt.tight_layout()

plt.savefig("Figura_5_heatmap_sensibilidad_demanda_capacidad.png", dpi=300, bbox_inches="tight")

plt.show()
```

8. Bibliografía.

Aena. (2026). *Business data on Adolfo Suárez Madrid-Barajas Airport*. Recuperado de https://www.aena.es/en/airlines/airports-and-destinations/our-airports/adolfo-suarez-madrid-barajas.html

Alsolbi, I., Hosseinnia Shavaki, F., Agarwal, R., Bharathy, G. K., Prakash, S., & Prasad, M. (2023). Big data optimisation and management in supply chain management: A systematic literature review. *Artificial Intelligence Review, 56*, 253-284.

Antizar, D. (2020). *Velocidad calles de Madrid* [Mapa interactivo]. Atlas. Datos procedentes del Portal de Datos Abiertos del Ayuntamiento de Madrid. Recuperado de https://app.atlas.co/shared/sk0OIALcXCHfeyiq8cfP?loc=-3.6709%2C40.4389%2C15.5728z&public=true&trk=public_post_comment-text

Ayuntamiento de Madrid. (2025). *Aforos de tráfico de la ciudad de Madrid no permanentes*. Portal de Datos Abiertos del Ayuntamiento de Madrid. Recuperado de https://datos.madrid.es/dataset/300209-0-aforos-no-permanentes

Büyüközkan, G., & Ilıcak, Ö. (2022). Smart urban logistics: Literature review and future directions. *Socio-Economic Planning Sciences, 81*, 101197.

Confederation of Tourism and Hospitality. (2011). *Front office operations*. CTH Resources.

Gillström, H., & Björklund, M. (2024). Assessing the benefits of urban consolidation centres: An overview based on a systematic literature review. *Transport Reviews, 44*(5), 972-991.

Instituto Nacional de Estadística. (2026). *Hotel tourism short-term trends (HOS/HPI/IPHS)*. Recuperado de https://www.ine.es/dyngs/INEbase/en/operacion.htm?c=Estadistica_C&cid=1254736177015&idp=1254735576863&menu=ultiDatos

Jorge, D., Rocha, T., & Ramos, T. R. P. (2024). A time-driven simulation-optimization framework for the dynamic heterogeneous order-courier assignment problem for instant deliveries. *Transportation Research Part E: Logistics and Transportation Review, 192*, 103783.

Kovač, M., Tadić, S., Krstić, M., & Veljović, M. (2023). A methodology for planning city logistics concepts based on city-dry port micro-consolidation centres. *Mathematics, 11*(15), 3347.

Recio, M. (2026) *Evaluación económica y estrategias de monetización del startup logístico Bag2Go en Madrid.* Trabajo Fin de Grado.

OpenRouteService. (s. f.). *OpenRouteService API*. Recuperado de: https://api.openrouteservice.org/

Overpass Turbo. (s. f.). *Overpass Turbo*. Recuperado de: https://overpass-turbo.eu/

Ross, S. M. (2012). *Simulation* (5th ed.). Academic Press.

Saldanha-da-Gama, F. (2022). Facility location in logistics and transportation: An enduring relationship. *Transportation Research Part E: Logistics and Transportation Review, 166*, 102903.