Universidades de Burgos, León y Valladolid

Máster universitario

Inteligencia de Negocio y Big Data en Entornos Seguros

TFM del Máster Inteligencia de Negocio y Big Data en Entornos Seguros

Exploración de tecnologías Python para la creación de cuadros de mandos e implementación para un caso de estudio dentro del entorno aeronáutico

Presentado por Laura Isabel López Naves
en Universidad de Valladolid — Septiembre de 2019
Tutor: Anibal Bregón Bregón, Miguel Ángel Martínez Prieto

"El objetivo último de la visualización
no es hacer visibles los datos ante nuestros ojos,
sino ante nuestro entendimiento."
-Yusef Hassan

Resumen

La gestión del tráfico aéreo (Air Traffic Management: ATM) se lleva a cabo en un entorno muy complejo en el que coexisten diferentes tipos de actores (como aeropuertos, aerolíneas, aeronaves, etc.) y en el que influyen, entre otros, aspectos como la meteorología, la estructura de los espacios aéreos alrededor de los aeropuertos, o los procedimientos de control aplicados. Además, los pronósticos invitan a pensar que esta complejidad seguirá aumentando en los próximos años, tanto por el número de variables que seguirán sumándose a la ecuación, como por el incremento previsto en el tráfico aéreo. Una de las tareas más importantes a la hora de tratar toda esa cantidad de información relacionada con la gestión del tráfico aéreo, es la creación de cuadros de mando (dashboards) capaces de representar de una forma clara y concisa los aspectos más destacados de la situación del espacio aéreo.

En este Trabajo Fin de Máster se llevará a cabo un estudio de las distintas alternativas que Python ofrece para la creación de dashboards, y la implementación, como caso de estudio, de un dashboard integrado, que ofrezca diferentes herramientas visuales para exploración y reporting de datos provenientes de la gestión de tráfico aéreo.

Descriptores

Cuadro de mandos, Python, ATM, ADS-B, Dash, Jupyter, Plotly, Visualización

Abstract

Air Traffic Management (ATM) is carried out in a very complex environment in which different types of actors coexist (such as airports, airlines, aircraft, etc.) and in which aspects such as meteorology, the structure of airspace around airports, or the control procedures applied, among others, influence. Furthermore, forecasts suggest that this complexity will continue to increase in the coming years, both in terms of the number of variables that will continue to be added to the equation, and in terms of the expected increase in air traffic. One of the most important tasks when using all this information related to air traffic management is the creation of dashboards capable of representing in a clear and concise way the most important aspects of the airspace situation.

In this Master's Thesis, a study will be carried out of the different alternatives that Python offers for the creation of dashboards, and the implementation, as a case study, of an integrated dashboard that offers different visual tools for exploration and reporting of data coming from air traffic management.

Keywords

Dashboard, Python, ATM, ADS-B, Dash, Jupyter, Plotly, Visualization

Índice general

Índice de figuras

Índice de tablas

Introducción

Este primer capítulo, surge de la necesidad de ofrecer al lector una visión global, del contexto en el que se enmarca este trabajo. En él, se llevará a cabo una breve introducción sobre la gestión actual del tráfico aéreo, la necesidad de una infraestructura Big Data debido al gran volumen de datos a tratar, y de un cuadro de mandos, imprescindible para poder extraer conclusiones estratégicas, a partir de la información generada. Además, dentro de este marco, se describirán la motivación y el problema que pone de manifiesto la necesidad de este trabajo, así como los objetivos que se pretenden alcanzar. Finalmente, se realizará una descripción, en forma de guía, de la estructura de este documento.

1.1. Introducción

En la actualidad, el transporte aéreo es uno de los medios más utilizados, tanto para el transporte de pasajeros como para el transporte de mercancías. Al desarrollarse en el medio aéreo, goza de la ventaja de no necesitar una pista en la superficie durante todo su trayecto (solo en el origen y en el destino). Se trata del medio de transporte más rápido para largas distancias, siendo imprescindible para envíos urgentes, de alto valor monetario o de mercancías perecederas. Además, según los datos recogidos por IATA[1] (International Air Transport Association) en su página web [10], es el medio de transporte de larga distancia más seguro. La tasa de siniestros total entre los años 2013 y 2017 alcanzó un 1,79 (medido en accidentes por millón de vuelos) y ha descendido a 1,35 en 2018, lo que se traduce en que se produce un accidente, por cada 740.000 vuelos. Gracias precisamente a estas ventajas, el uso de aviones como medio de transporte se ha disparado, y se prevé que siga

[1] https://www.iata.org [Junio 2019]

creciendo en los próximos años. De hecho, EUROCONTROL[2] (Organización Europea para la Seguridad de la Navegación Aérea), en su última previsión semestral [5], prevé una tasa media de crecimiento en el periodo 2018 a 2024 anual del 3,1 % (±1,5 puntos porcentuales), y un crecimiento total del 24 % del total de unidades de servicio en ruta en comparación con 2017.

A pesar de este aumento del tráfico aéreo, el sistema ATM europeo actual, es un sistema muy poco eficiente, lo cual genera un importante coste económico, debido principalmente a su organización basada en bloques nacionales, en lugar de en bloques acordes al tráfico aéreo. Para tratar de afrontar esta situación, la Comisión Europea lanzó en el año 1999, la iniciativa de Cielo Único Europeo[3] (Single European Sky, SES), proponiendo un gran número de acciones de mejora para optimizar la gestión del tráfico aéreo. Uno de los pilares de SES es el programa de investigación SESAR[4] (Single European Sky ATM Research), en el que se afronta la integración y modernización del sistema ATM en la Unión Europea. Dentro del programa SESAR se engloba el proyecto AIRPORTS [24], proyecto que trata de explorar áreas de mejora para el transporte aéreo, a través de la reconstrucción de trayectorias de vuelo, a partir de los datos obtenidos por los sistemas de control. Una vez reconstruidas, se calculan diferentes métricas de las trayectorias para determinar la eficiencia de los vuelos.

Debido a esta necesidad de mejora de los sistemas de control de tráfico aéreo (ATC), ha surgido recientemente una nueva técnica de vigilancia ADS (Automatic Dependent Surveillance). Esta tecnología permite a la aeronave transmitir información de posicionamiento de forma automática, a diferencia de las tecnologías tradicionales de vigilancia por radar utilizadas hasta el momento, que miden el alcance y la dirección de una aeronave utilizando una antena terrestre. La capacidad de transmitir continuamente el estado de las aeronaves, disminuye la necesidad de tecnologías más costosas y precisas, reduciendo notablemente los costes derivados del control del tráfico aéreo. Es precisamente en esta tecnología, en la que se apoyan iniciativas como SESAR en Europa y NextGen[5] (Next Generation Air Transportation System) en los EEUU, para abordar nuevos retos en el ambiente de la gestión de tráfico aéreo, como por ejemplo la gestión de trayectorias 4D.

En el entorno aéreo se generan gran cantidad de datos a diario, debido al elevado número de sistemas que lo constituyen, como sensores de radar, datos

[2]https://www.eurocontrol.int/ [Junio 2019]
[3]https://ec.europa.eu/transport/modes/air/ses en/ [Junio 2019]
[4]https://www.sesarju.eu/ [Junio 2019]
[5]https://www.faa.gov/nextgen/ [Junio 2019]

meteorológicos, sistemas de control, etc. Es precisamente a consecuencia del gran volumen de datos generados por todos estos sistemas, junto con su variedad, y la velocidad de acceso, donde surge la necesidad del uso de tecnologías Big Data. Razón por la cual, el proyecto AIRPORTS, ha sido diseñado bajo una arquitectura Big Data utilizando el framework de procesamiento distribuido Hadoop.

La información procesada a través del proyecto AIRPORTS, será el punto de partida de este trabajo, a través del cual se pretende poder explotar y analizar esta gran cantidad de información [11], mediante métodos de visualización que ayuden a obtener conclusiones, que permitan detectar áreas de mejora en los sistemas ATM. Más concretamente, mediante la elaboración de un cuadro de mandos, al ser éste la herramienta idónea para llevar un control y un seguimiento de cualquier negocio, y por ende, ideal para tratar de mejorar la gestión de los sistemas de tráfico aéreo.

1.2. Motivación

Hoy en día las cancelaciones y los retrasos en los vuelos, son problemas frecuentes que afectan de forma negativa tanto a la satisfacción de los pasajeros, como al rendimiento global del sistema aéreo, generando además, pérdidas económicas debido a un aumento del consumo de combustible y provocando un aumento de la tasa de emisiones.

Durante 2018, el tráfico mundial de pasajeros ha superado los 4.300 millones de pasajeros que viajaron en vuelos regulares, un 6,1 % más que en 2017, según el informe preliminar de la Organización de Aviación Civil Internacional (OACI) [18]. Además, el número de vuelos aumentó a 38 millones en todo el mundo, y el tráfico mundial expresado como total de pasajeros-kilómetros de pago (RPK) creció un 6,7 %, alcanzando los 8.200 billones de RPK. Las previsiones dicen que estas cifras seguirán aumentando, por lo que cada vez cobrará más importancia la mejora de los sistemas de gestión actuales, insuficientes, para gestionar tales dimensiones de datos.

Este trabajo surge de la necesidad de mejorar estos sistemas de gestión de tráfico aéreo, ya no solo desde un punto de vista económico, sino además tratando de garantizar la seguridad del espacio aéreo, y de una forma comprometida con el medio ambiente. Para detectar estas áreas de mejora y debido a la gran cantidad de datos que maneja el entorno aéreo, se hace imprescindible buscar la manera de dotar de significado a esos datos y convertirlos en información. El ser humano está acostumbrado a recibir la mayor parte de la información que emplea, a través de la vista, por lo que la

forma más intuitiva de presentar la información es de forma visual. El desafío consiste en ser capaz, mediante algún tipo de visualización, de resumir y filtrar de forma satisfactoria una gran cantidad de información, destacando de forma eficiente y sagaz los aspectos a los que se desea proporcionar mayor relevancia.

Entre los métodos de visualización existentes, ha cobrado especial importancia el cuadro de mandos [23], ya que es una herramienta, que permite visualizar el problema y favorecer la toma de decisiones, consiguiendo dotar de significado a los datos. De esta forma, el usuario, será capaz de orientar la estrategia de su negocio hacia la consecución de los objetivos planteados. En este caso concreto, hacia la eficiencia y optimización de los sistemas ATM.

En un mercado cada vez más exigente, se hace fundamental la utilización de las tecnologías más punteras, así como la elección de un lenguaje de programación que se ajuste al problema a resolver. Desde hace cinco años, IEEE Spectrum[6], la organización profesional más grande del mundo dedicada a la ingeniería y las ciencias aplicadas, publica un ránking con el Top 10 de lenguajes de programación [19]. Durante los dos últimos años, Python[7] ha sido el número uno del ránking. Además Python, es uno de los lenguajes de programación más utilizados en la actualidad. Muchas de las herramientas que han aparecido en los últimos años, a raíz sobre todo de la expansión del Big Data, seguida de los avances en la Inteligencia Artificial y en el Aprendizaje Automático, así como del estallido de la Ciencia de Datos, han sido desarrolladas en Python u ofrecen Python como forma de interactuar con ellas. Todo esto ha motivado que para tratar de encontrar factores de mejora en los sistemas aéreos, se haya decidido realizar un estudio exhaustivo de las alternativas que Python ofrece, para la elaboración de cuadros de mandos.

1.3. Descripción del problema

Con el fin de satisfacer las necesidades de explotación de la información obtenida a través del proyecto AIRPORTS, llevado a cabo por la compañía aérea Boeing Research & Technology Europe (BR&T-E)[8] en colaboración con la Universidad de Valladolid, surge la necesidad de plantear un cuadro de mandos que permita de una forma óptima, mejorar la gestión de las operaciones aeroportuarias.

[6]https://spectrum.ieee.org/ [Junio 2019]
[7]https://www.python.org/ [Junio 2019]
[8]https://www.boeing.com/ [Junio 2019]

El concepto de cuadro de mandos surge de la mano de Robert Kaplan y David Norton, dos economistas norteamericanos que en 1992 realizaron una prestigiosa publicación [23] en Harvard Business Review. El artículo recoge el resultado de mejora obtenido en el rendimiento de 12 empresas colaboradoras, al hacer uso de un cuadro de mandos integral, durante un año. Los autores relacionan el cuadro de mandos integral con los existentes en las cabinas de un avión, donde se hace necesario conocer gran cantidad de parámetros, para que el piloto pueda decidir las acciones futuras. Haciendo un símil con como la complejidad de mejorar una empresa en la actualidad, requiere de un estudio exhaustivo y combinado de varios parámetros. El artículo define cuatro pilares fundamentales a través de los cuales, se pretende mejorar la gestión empresarial: visión del cliente, puntos fuertes, formación e innovación y acciones financieras.

El cuadro de mandos que se realiza en la actualidad, no sigue siempre este esquema inicial, pero sí se basa en él, englobando además, un sistema de indicadores clave de rendimiento KPIs (Key Performance Indicators), que acompañados de una representación gráfica, facilitan la toma de decisiones y la detección de puntos de cambio. Un cuadro de mandos es en definitiva, una herramienta que permite obtener de un solo vistazo, la situación global de un negocio en un momento dado. El uso de un cuadro de mandos, facilitará y agilizará la toma de decisiones para la mejora de los sistemas aéreos. Conociendo la ya mencionada importancia, de la tecnología Python, se puede acotar el problema planteado, al uso de las diferentes herramientas que Python proporciona para la realización de cuadros de mandos.

Además, aunque este trabajo parte de los datos ya procesados por el proyecto AIRPORTS [16], dado su gran volumen, no solo existe el problema de la realización de un cuadro de mandos accesible, usable, y que arroje la información necesaria para la toma de decisiones estratégicas futuras utilizando el lenguaje Python, sino también con el problema de conseguir que ese volumen de información, no ralentice el uso del dashboard. Por tanto, será necesario plantear una pequeña arquitectura que segmente los ficheros de datos, así como, agilice el filtrado de la información.

1.4. Objetivos

Para llevar a cabo este trabajo se ha definido un objetivo principal, que consiste en ***la exploración de tecnologías Python, para la creación de un cuadro de mandos rápido y funcional, que ayude a detectar áreas de mejora en la gestión del tráfico aéreo***. Para lograr este

objetivo se han propuesto cuatro objetivos parciales:

1. Realizar un estudio de las distintas alternativas que Python ofrece para la creación de cuadros de mandos.

   Para lograr este objetivo, se debe realizar un recorrido por las tecnologías Python más utilizadas en la actualidad, tratando de establecer sus ventajas e inconvenientes.

2. Seleccionar la tecnología más adecuada de entre las propuestas

   Se trata de lograr encontrar la tecnología de entre las que Python ofrece, que mejor se adapte, al escenario de mejora de gestión del tráfico aéreo, en el que nos encontramos.

3. Crear una arquitectura para la gestión de la información.

   Para alcanzar este objetivo, se debe buscar la manera óptima de gestionar los datos procedentes del proyecto AIRPORTS.

4. Construir el cuadro de mandos.

   Para lograr este objetivo, se deberá construir un cuadro de mandos rápido y funcional, que permita obtener información relevante para la mejora de la gestión de tráfico aéreo.

1.5. Estructura del documento

Este documento se encuentra estructurado en los siguientes capítulos:

- En el Capítulo 2 se describe la planificación seguida durante la realización de este trabajo, así como una estimación de los costes derivados del desarrollo del mismo.

- El Capítulo 3 presenta el marco teórico y el estado del arte. En él se recogen algunos de los conceptos más relevantes en la gestión del tráfico aéreo, imprescindibles para poder comprender la información que se va a manejar durante el trabajo, así como la importancia de llevar a cabo la gestión de la información en un entorno Big Data, y su exploración mediante un cuadro de mandos. Además, en el estado del arte, se realizará un recorrido por las distintas opciones de visualización de datos más utilizadas en la actualidad y más concretamente de las enfocadas a la visualización de información aérea.

- El Capítulo 4 describe qué es Python, y muestra un estudio de las distintas alternativas que ofrece para la realización de cuadros de mandos. Además, se elegirá una de estas herramienta para realizar la implementación de un dashboard, como caso de estudio.
- El Capítulo 5 detalla una propuesta de arquitectura para la gestión de los datos, que permitirá agilizar el funcionamiento del cuadro de mandos, explicando como se han segmentado los ficheros de origen, y como se han almacenado ciertos datos en la base de datos para facilitar los filtrados.
- El Capítulo 6 describe todo el proceso de construcción del dashboard desde la instalación y configuración del entorno, hasta los distintos elementos que componen el cuadro de mandos.
- En el Capítulo 7 se explica de forma íntegra, utilizando ejemplos reales, el funcionamiento del cuadro de mandos.
- En el Capítulo 8 se presentan las conclusiones y se propone el trabajo futuro.
- El documento incluye, por último, unos apéndices que contienen el código de algunos de los procesos utilizados para la segmentación y almacenamiento de los datos.

Planificación y estimación de costes del trabajo

En este capítulo se llevará a cabo una descripción de la planificación seguida durante la realización de este trabajo, además se realizará una estimación de los costes derivados del desarrollo del mismo.

2.1. Plan del trabajo

Debido a que las asignaturas del segundo semestre terminaron mas tarde de lo esperado, y que la elaboración de este trabajo fue llevada a cabo de forma simultánea con una jornada laboral a tiempo completo, el desarrollo de este trabajo no pudo comenzar de una forma estable hasta el mes de junio. Inicialmente se trató de alcanzar la primera convocatoria, no siendo posible la realización de este trabajo, en tiempo y forma, por lo que se decidió su entrega en segunda convocatoria.

Su desarrollo ha ido evolucionando en torno a cuatro etapas muy diferenciadas, que se detallan a continuación mediante una tabla, que describe las tareas que se han ido realizando, y el tiempo estimado para cada una de ellas. Cabe destacar, que durante este periodo se mantuvieron reuniones de seguimiento semanales con los tutores, en las que se iban presentando los avances y las dudas, para finalmente proponer las tareas a realizar durante la siguiente etapa.

2.1.1. Etapa de Investigación y Adquisición de conocimientos

En la primera etapa cuyas tareas se pueden ver en la Tabla 2.1, se llevó a cabo un proceso de investigación y análisis sobre los sistemas de gestión del tráfico aéreo actuales, consultando diversas páginas web y artículos de investigación relacionados. Tras afianzar los conceptos del sector aéreo, se procedió a comprender el modelo conceptual y el Data Lake utilizados por el proyecto AIRPORTS, poniendo el punto de interés en los datos que se deseaban analizar en el cuadro de mandos. Además una vez conocido el alcance del trabajo, se estableció la planificación a seguir durante la realización del mismo. Finalmente, se realizó un análisis exhaustivo de las diferentes herramientas de visualización, existentes en la actualidad y más concretamente de las orientadas al sector aéreo.

Id	Descripción	Tiempo Estimado
T-1	Aprendizaje y documentación sobre la gestión del tráfico aéreo	8 horas
T-2	Lectura y comprensión de artículos relacionados	10 horas
T-3	Comprensión del modelo conceptual y del Data Lake utilizado por AIRPORTS	5 horas
T-4	Realización de un estudio sobre herramientas de visualización	10 horas
T-5	Realización de un estudio sobre herramientas de visualización utilizadas en el sector aéreo	10 horas
T-6	Redacción de los capítulos 1, 2 y 3	10 horas
		53 horas

Tabla 2.1: Tareas Primera Etapa

2.1.2. Etapa de investigación de librerías Python

En esta segunda etapa, cuyas tareas se detallan en la Tabla 2.2, se ha llevado a cabo una investigación sobre las principales características del lenguaje Python, y las ventajas que proporciona para la realización de

cuadros de mando, realizando un recorrido por las distintas librerías de visualización que Python ofrece en la actualidad.

Id	Descripción	Tiempo Estimado
T-7	Investigación sobre las características más relevantes de Python	5 horas
T-8	Realización de un estudio sobre las distintas herramientas que ofrece Python para la realización de cuadros de mando	8 horas
T-9	Selección de una de las librerías de visualización, analizando las opciones más relevantes	4 horas
T-10	Redacción del capítulo 4	10 horas
		27 horas

Tabla 2.2: Tareas Segunda Etapa

2.1.3. Etapa de elaboración del cuadro de mandos

Esta etapa ha sido la mas larga, ya que en ella se recogen todas las tareas relacionadas con la construcción y el diseño del dashboard, así como las tareas relacionadas con la segmentación y almacenamiento de datos en la arquitectura establecida. En la Tabla 2.3 se pueden ver las tareas y el tiempo estimado de cada una de ellas.

2.1.4. Etapa de pruebas y conclusiones

En esta etapa se realizaron diversas pruebas del dashboard, para arreglar los posibles errores que se fueran detectando. Se añadieron inicialmente los datos generados en un día y se hicieron pruebas de todos los menús y submenús, completando posteriormente las pruebas añadiendo datos de otra fecha. Además se obtuvieron las conclusiones y se establecieron líneas de trabajo futuras. En la Tabla 2.3 se recogen las tareas y los tiempos estimados de cada una de ellas.

Id	Descripción	Tiempo Estimado
T-11	Instalación del entorno Anaconda	0.25 horas
T-12	Instalación de los paquetes y librerías utilizados para la construcción del dashboard	0.5 horas
T-13	Instalación de Postgres	0.5 horas
T-14	Generación del mapa con Mapbox Studio	0.75 horas
T-15	Generación de scripts para la segmentación de mensajes y el almacenamiento de tramos	10 horas
T-16	Aprendizaje y documentación sobre el uso de la librería Dash para la creación de dashboards	8 horas
T-17	Creación de la pestaña de trayectorias del dashboard	30 horas
T-18	Generación de scripts para la segmentación de las métricas	5 hora
T-19	Creación de la pestaña de métricas del dashboard	15 horas
T-20	Generación de scripts para el almacenamiento y la segmentación de las trayectorias sintéticas	4 horas
T-21	Creación de la pestaña de trayectorias reconstruidas del dashboard	10 horas
		84 horas

Tabla 2.3: Tareas Tercera Etapa

2.1.5. Diagrama de Gantt

En la Figura 2.1 se puede ver un diagrama de Gantt que recoge la organización temporal del trabajo. El diagrama muestra las 25 tareas clasificadas en sus 4 etapas mediante colores, realizadas del 03 de junio de 2019 al 4 de septiembre. Se ha eliminado del cronograma el periodo del 8 de julio al 12 de agosto de 2019, por destinarlo a vacaciones.

Id	Descripción	Tiempo Estimado
T-22	Realización de pruebas del dashboard	12 horas
T-23	Redacción de los capítulos 5, 6 y 7	15 horas
T-24	Análisis del trabajo realizado, obtención de conclusiones y propuesta de trabajos futuros	8 horas
T-25	Redacción del capítulo 8	6 horas
		41 horas

Tabla 2.4: Tareas Cuarta Etapa

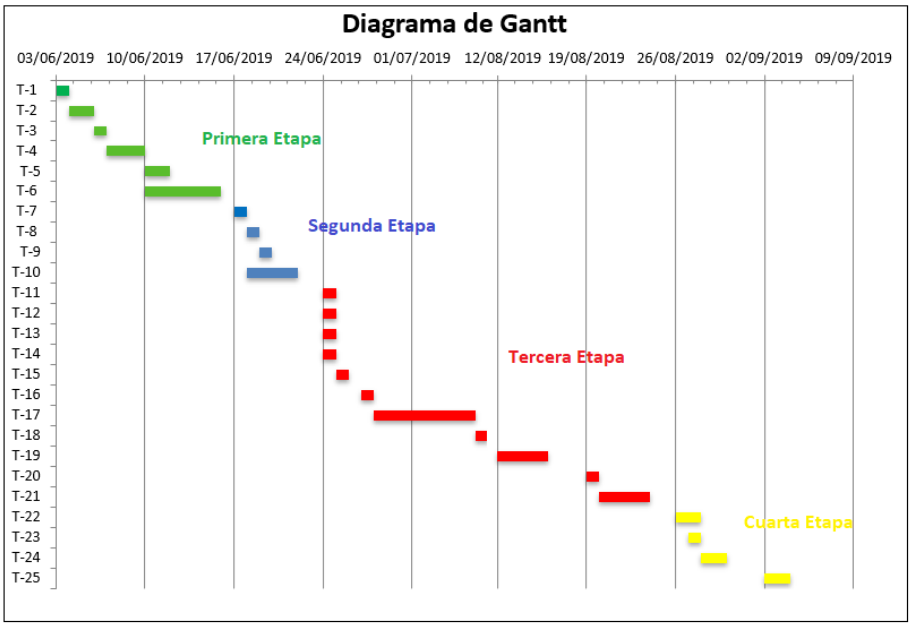

Figura 2.1: Diagrama de Gantt

2.2. Estimación de costes del trabajo

A la hora de realizar una estimación de costes es importante distinguir entre el cálculo del coste de la realización del producto, es decir cuanto costará realizarlo y el precio final que se establecerá. Este precio final es más bien una decisión ejecutiva, desde el punto de vista de que no solo influye el coste de la realización del producto, sino también otros muchos factores

como pueden ser la demanda, innovación, o el precio de mercado.

Para estimar el coste de este trabajo no vamos a establecer un precio final ajustado al mercado, sino que vamos a evaluar el coste de elaboración del producto, haciendo un recorrido por el gasto de hardware, software y de desarrollo del cuadro de mandos.

2.2.1. Coste del Hardware

Durante el desarrollo de este trabajo ha sido imprescindible la utilización de los siguientes elementos hardware:

- Ordenador portatil Intel Core i7-6700HQ CPU @ 2.60GHz 2.59GHZ con 16 GB de RAM y disco duro de SandDisk SDSSDA-480G-G26 Plus SATA de 480GB. El tiempo de vida estimado de un ordenador hoy en día, suele ser alrededor de 5 años (60 meses). Dado que el ordenador tiene un coste de unos 1.100 euros y que hemos hecho uso de él durante 213 horas, lo que se traduce en unos 27 días de 8 horas, el desgaste del ordenador se puede valorar aproximadamente en unos 18,30 euros (resultado de dividir 1.100 euros entre 60 meses para obtener el coste de un mes).
- Internet. Se ha hecho un uso aproximado de una cuota mensual, lo que viene a ser unos 35 euros.
- Luz. Se ha estimado un uso aproximado de una cuarta parte del importe de una factura mensual, es decir 11,25 euros aproximadamente.

Por tanto se puede concluir que el gasto en hardware ha sido de unos 64,55 euros.

2.2.2. Coste del Software

Este trabajo ha requerido de las herramientas software que se muestran a continuación:

- Sistema Operativo Windows 10 de 64 bits
- Anaconda
- Postgres
- Librerias y Paquetes Python

- Mapbox Studio
- Paletton
- Firefox Developer
- Balsamiq

Todas estas herramientas son gratuitas salvo Balsamiq, de la cual se ha utilizado una versión de prueba, por lo tanto sin coste alguno, y el sistema operativo Windows 10 que si tiene un coste que ronda los 100 euros. El presupuesto software estimado será por tanto de 100 euros.

2.2.3. Coste del desarrollo

Este coste lo calcularemos utilizando las horas dedicadas a su realización, es decir realizando la suma de las horas utilizadas durante el desarrollo de las 4 etapas en las que se ha divido el desarrollo de este trabajo y añadiendo unas 8 horas, utilizadas para reuniones de seguimiento con los profesores, obteniendo un total de 213 horas. El salario medio de un graduado en ingeniería informática es de unos 30.000 euros brutos anuales [6], lo que se traduce en unos 1.700 euros netos al mes, es decir en unos 10,65 euros la hora. Por tanto el coste de este desarrollo lo podemos estimar en unos 2.268,45 euros.

2.2.4. Coste total

A continuación se muestra la Tabla 2.5 en la que se recoge el coste total de la realización del cuadro de mandos y que asciende a 2.433 euros.

Componente	Coste Estimado
Hardware	64,55 euros
Software	100 euros
Desarrollo	2.268,45
TOTAL	2.433 euros

Tabla 2.5: Coste total estimado

Marco Teórico y Estado del Arte

Este capítulo pretende ofrecer una descripción general de los conceptos básicos de la gestión del tráfico aéreo, imprescindibles para interpretar y entender la información con la que se va a trabajar a lo largo de este proyecto. También se mostrará la relación de este sector con el Big Data a través del proyecto AIRPORTS y se llevará a cabo un análisis del estado del arte, realizando un recorrido por los sistemas de visualización de datos existentes en la actualidad, y de forma más concreta de los enfocados a visualización de datos aéreos.

3.1. Conceptos Básicos

Este trabajo nace de la necesidad de implementar procesos de mejora en los sistemas de gestión del tráfico aéreo (Air Traffic Management, ATM), que engloban no solo toda la información del vuelo en sí, sino también información relacionada, como es la información meteorológica, estructuras alrededor del aeropuerto, aerolíneas, etc. Actualmente existen diversas iniciativas de investigación que tratan de conseguir mejorar estos sistemas ATM, como es el caso de SESAR (Single European Sky ATM Research) y NextGen (Next Generation Air Transportation System). La forma de obtener los datos para este caso de estudio, se basa precisamente en una de las principales líneas de trabajo de SESAR, la gestión de trayectorias 4D, la cual consiste en añadir el tiempo en la información de la trayectoria 3D (latitud, longitud y altitud) del avión, buscando establecer las trayectorias de la forma más óptima posible.

Tanto SESAR como NextGen se apoyan en ADS (Automatic Dependent Surveillance), tecnología de vigilancia que permite a las aeronaves propor-

cionar de forma automática y precisa, los datos que extraen a partir de sus sistemas de navegación y de posición. Una de las tecnologías utilizada internamente por ADS es ADS-B (ADS-Broadcasting), técnica que permite a cada aeronave emitir de forma automática y periódica información de posicionamiento a través de unos sensores. Esta información de posicionamiento será transmitida a todos los receptores que se encuentren en su rango de frecuencia. Como su propio nombre indica, se trata de una tecnología que goza de las siguientes características:

- Automática - No requiere de ningún tipo de intervención, ni siquiera por parte del piloto.
- Dependiente - Depende de la precisión de los datos enviados por el sistema de posicionamiento de la aeronave.
- Vigilancia - Proporciona datos de vigilancia y control (altitud, velocidad, posición, etc).
- Difusión - La información se difunde continuamente para su monitorización, tanto a estaciones terrestres, como a otras aeronaves.

La figura Figura 3.2 muestra un esquema de funcionamiento del sistema ADS-B. Las aeronaves obtienen su posición a través de los sensores que llevan incorporados, junto con los datos de posicionamiento de los satélites, y transmiten esta información a cualquier receptor que se encuentre dentro de su área de cobertura. Esta información es recibida tanto por las estaciones en tierra como por otras aeronaves cercanas.

ADS-B soporta 6 tipos distintos de mensajes (MSG, SEL, ID, AIR, STA, CLK), para nuestro caso de estudio nos centraremos en los mensajes ADS-B de tipo MSG. Estos mensajes tienen 8 tipos diferentes de transmisión. La Tabla 3.6 enumera el conjunto de campos utilizados por estos mensajes MSG: los primeros 10 campos son obligatorios para todos los tipos de mensajes, pero el uso de los campos del 11 al 22 depende de la semántica particular de cada tipo de mensaje. Cabe señalar, que ningún tipo de mensaje, proporciona todos los campos necesarios para obtener la información del vector de estado (identificación del vuelo, geolocalización, altitud y velocidad). Por lo tanto, los mensajes en bruto deben fusionarse para obtener, al menos, la descripción del vector de estado.

Cuando no existen receptores en la zona, los mensajes se pierden, razón por la cual, tratando de conseguir una mayor cobertura, los datos utilizados

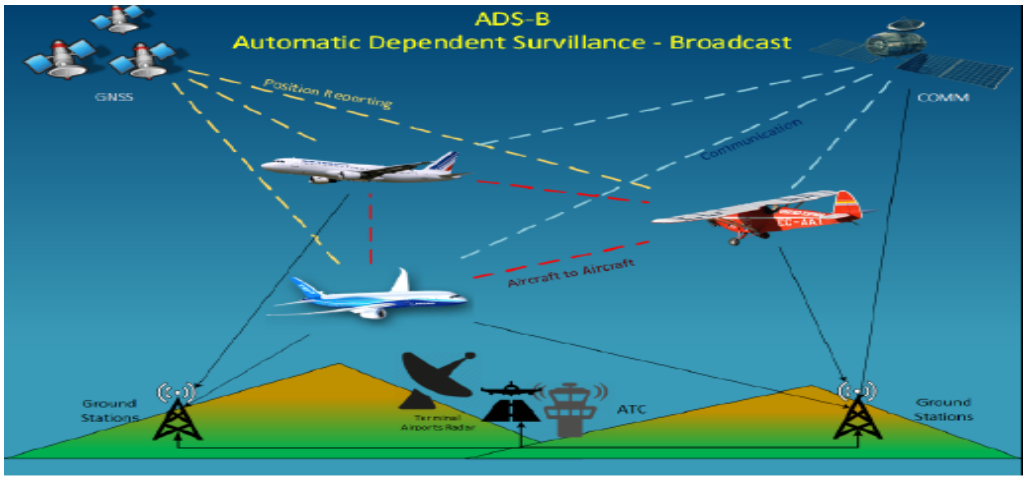

Figura 3.2: Esquema de funcionamiento de ADS-B. [9]

#	Name	Description
1	`Message`	`MSG,STA,ID,AIR,SEL,CLK`.
2	`Type`	Type of `MSG` message: 1-8.
3	`SessionID`	Database session identifier.
4	`AircraftID`	Database aircraft record number.
5	`HexIdent`	ICAO 24-bit address (*Mode-S hexadecimal code*).
6	`FlightID`	Database flight record number.
7	`Gen. Date`	Generated date (from the aircraft).
8	`Gen. Time`	Generated time (from the aircraft).
9	`Logged Date`	Logged date (from the base station).
10	`Logged Time`	Logged time (from the base station).
11	`Callsign`	Flight identifier (ICAO airline code + flight number).
12	`Altitude`	Barometric altitude (25ft or 100 ft resolution).
13	`Speed`	Ground speed (knots).
14	`Track`	Derived from the velocity E/W and N/S. (degrees)
15	`Latitude`	North/East positive, South/West negative (degrees).
16	`Longitude`	North/East positive, South/West negative.
17	`Vertical`	Aircraft vertical rate (ft/sec).
18	`Squawk`	Assigned squawk code.
19	`Alert`	Flag to indicate the squawk value has changed.
20	`Emergency`	Flag to indicate emergency code has been set.
21	`SPI (Ident)`	Flag to indicate transponder Ident has been activated.
22	`IsOnGround`	Flag to indicate ground squat switch is active.

Tabla 3.6: Campos utilizados en los mensajes ADS-B [16]

en este trabajo se han obtenido de diferentes proveedores ADS-B como son Frambuesa, OpenSky, etc.

- Frambuesa, es un proveedor ADS-B, desplegado sobre una simple Raspberry-pi, propiedad de Boeing Research & Technology Europe (BR&T-E). Frambuesa proporciona información muy detallada sobre

los vuelos que operan en Madrid, ya que se encuentra ubicado muy cerca del aeropuerto de Barajas. Para poder conocer el estado en detalle de una aeronave en cada instante, será indispensable combinar mensajes de diferentes tipos de transmisión.

- Opensky [13], es un proveedor que posee una red de receptores ADS-B distribuidos por Europa, Estados Unidos, y ciertas áreas de Latino América, Asia y Australia. OpenSky recibe los mensajes ADS-B, los preprocesa y los convierte en vectores de estado, generando de esta forma información de surveillance (vigilancia). Cada vector de estado incorpora los atributos más significativos del vuelo y se construye combinando mensajes ADS-B de diferentes tipos de transmisión. La información proporcionada por estos vectores de estado es idónea para el modelo de dominio del proyecto AIRPORTS, aunque es importante recalcar que OpenSky convierte todas las mediciones al sistema decimal (a diferencia de la información enviada originalmente por los aviones). Un artículo bastante actual [14], ofrece información sobre la arquitectura de datos de OpenSky, explicando como tuvo que ser sustituida por una arquitectura Big Data debido a la falta de escalabilidad del sistema original implementado bajo MySQL.

Todos los datos obtenidos desde estos proveedores se reciben en tiempo real. Además cada avión envía un nuevo mensaje ADS-B aproximadamente dos veces por segundo, como se describe en el artículo [14] y el número de vehículos que vuelan en un día en el mundo, es superior a 100.000, muchos de los cuales realizan más de un vuelo al día. Actualmente, el proyecto AIRPORTS [16] recibe a diario más de 50 millones de mensajes (necesitando alrededor de 10GB para su almacenamiento) que se almacenan clasificados por días, en directorios diferentes para cada origen de datos.

Conviene indicar que además de la información de vigilancia, se ha utilizado información perteneciente a los planes de vuelo que proporcionan las compañías aéreas, los cuales proporcionan datos diversos sobre el vuelo, previos a su despegue. Para ello, se ha utilizado el servicio DDR[9] (Demand Data Repository), gestionado por EUROCONTROL, obteniendo los planes de vuelo correspondientes a todos aquellos vuelos realizados en el espacio aéreo europeo. Finalmente para completar toda esta información, se ha añadido de forma manual, información relativa a descripciones de aeropuertos, aerolíneas y aeronaves.

[9]https://www.eurocontrol.int/ddr [Junio 2019]

Debido a la exigencia de almacenar y gestionar esta gran cantidad de información, ha sido necesario buscar una forma de almacenamiento capaz de operar con datos tan grandes y tan desestructurados. Las soluciones tradicionales no están diseñadas ni están optimizadas para integrar distintas fuentes de datos, además preestablecen el modelo de datos y requieren mucho tiempo y coste de procesamiento para optimizar el cálculo de analíticas. Es por ello, que Boeing Research & Technology Europe (BR&T-E) empresa colaboradora con la Universidad de Valladolid a través del proyecto AIRPORTS utiliza una arquitectura Big Data y más concretamente un Data Lake [9] para la gestión de toda esta información.

Un Data Lake es un repositorio de almacenamiento que preserva una gran cantidad de datos en su formato original (rawdata) y ofrece mecanismos para investigar, explorar, experimentar y refinar esos datos para su puesta en explotación. La Figura 3.3 muestra un esquema del funcionamiento de un Data Lake. Los artículos [21] y [22] explican en detalle el concepto de Data Lake, comparándolo con otras arquitecturas de Big Data.

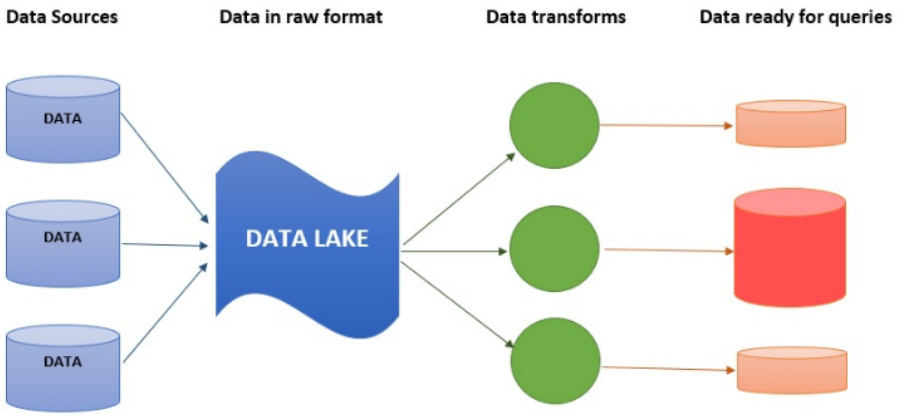

Figura 3.3: Funcionamiento de un Data Lake [1]

El proyecto AIRPORTS implementa su Data Lake utilizando el ecosistema tecnológico Apache Hadoop[10], que permite transformar los grandes volúmenes de datos recibidos a través de la tecnología ADS-B en información útil para la gestión de tráfico aéreo. Para ello utiliza entre otras, las tecnologías Big Data más utilizadas del momento, entre las que cabe

[10]https://hadoop.apache.org/ [Junio 2019]

destacar Flume[11], HDFS[12], MapReduce, Spark[13] y Hive[14]. La elección de estas tecnologías fue motivada por la necesidad de capturar, preservar y procesar grandes colecciones de datos.

A continuación, se especificará el modelo conceptual y los datos de destino generados por el proyecto AIRPORTS [17]. Dichos datos de destino serán el punto de partida de este trabajo.

3.1.1. Modelo conceptual

El objetivo principal de un modelo conceptual es organizar y representar, de manera semiformal, el conocimiento de un área o campo específico asociado a un sistema de gestión e información. Un modelo conceptual es una descripción de alto nivel de las necesidades de información de una empresa. Generalmente comprende sólo los conceptos fundamentales y las relaciones entre ellos.

En la Figura 3.4 se muestra el modelo conceptual UML que ha sido utilizado en el proyecto AIRPORTS. En el diagrama, podemos observar cómo una aerolínea opera con diferentes vuelos, mientras que un vuelo únicamente puede ser operado por una aerolínea. La entidad mensaje facilita una descripción del vector de estado. Los mensajes se utilizan para reconstruir un tramo (leg) de un vuelo (entendiendo por tramo a una parte del vuelo entre dos aeropuertos que se realiza con el mismo identificador y la misma aeronave). Es por ello que, un vuelo puede estar compuesto de uno o más tramos ordenados en el tiempo, de los que se almacena el aeropuerto de origen y de destino.

Cada tramo está regulado por su plan de vuelo, que proporciona gran cantidad de información relacionada con el vuelo, como son la pista y el modelo de la aeronave, el consumo de combustible, la hora estimada de despegue, etc. Para describir todos los conceptos del modelo, es necesario utilizar muchos atributos. La marca de tiempo de cada mensaje almacenado, su latitud y longitud, el hexident, son atributos muy valiosos que se extraen directamente de los mensajes ADS-B.

[11] https://flume.apache.org/ [Junio 2019]
[12] https://hadoop.apache.org [Junio 2019]
[13] https://spark.apache.org [Junio 2019]
[14] https://hive.apache.org/ [Junio 2019]

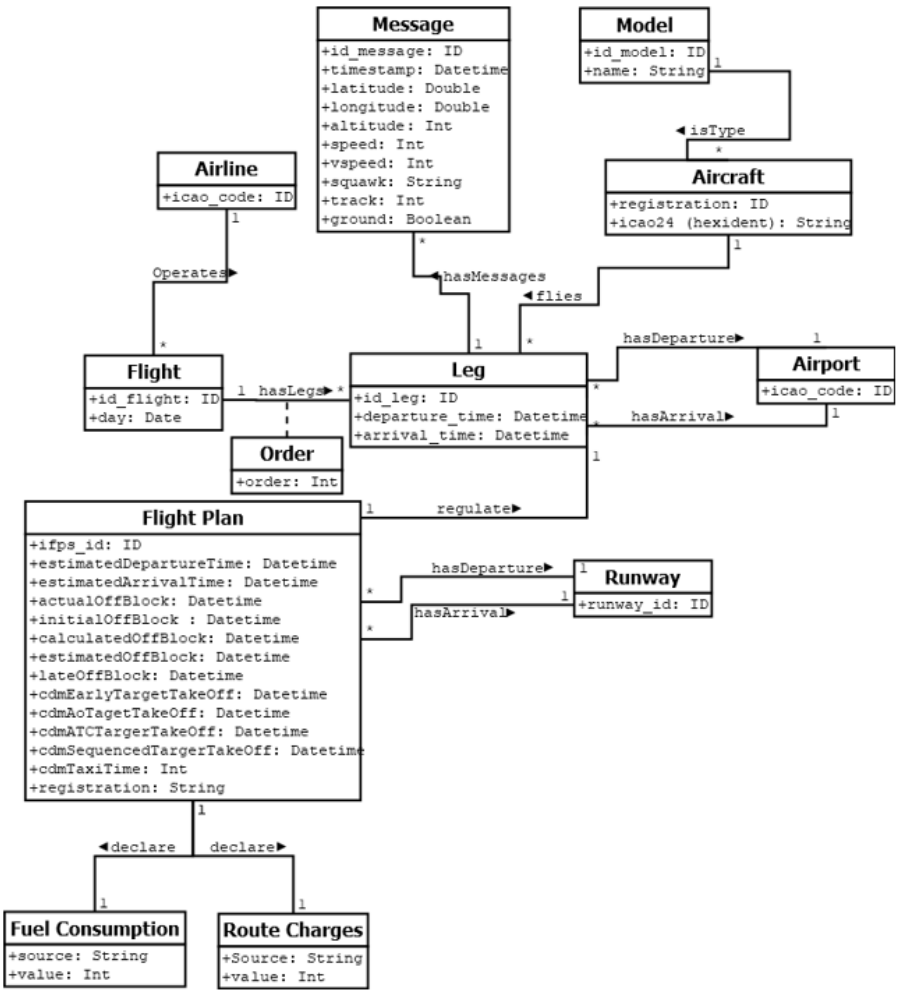

Figura 3.4: Modelo de datos conceptuales [17]

3.1.2. Datos de destino

Los datos obtenidos después de la transformación llevada a cabo por AIRPORTS resultado de la limpieza, procesamiento e integración de las diferentes fuentes de datos, serán el destino del proyecto AIRPORTS y por tanto el origen de este trabajo.

Estos datos objetivo son una implementación del modelo conceptual propuesto en la sección anterior. Debido a que se han utilizado sistemas capaces de trabajar con datos no estructurados, se ha utilizado un modelo de datos desnormalizado. En sistemas con un volumen de datos tan elevado es importante controlar los tiempos, por lo que se hace fundamental reducir al

máximo el número de entidades y relaciones, facilitando así que las consultas sean ágiles, al no requerir de un control de integridad. Para conseguir esto, ha sido necesario aumentar los atributos, en torno únicamente a dos objetos, que permiten almacenar la información imprescindible para completar el modelo conceptual: tramo y mensaje.

Como por medio de ADS-B no se puede conocer el tramo al que pertenece cada mensaje, ha sido necesario además, combinar los mensajes que pertenecen a cada uno de los tramos. Los planes de vuelo a su vez, no tienen relación directa con el tramo, por lo que ha sido necesario establecer, cuando un plan de vuelo hace referencia a un tramo concreto.

La Tabla 3.7 muestra los atributos que forman parte de cada uno de los tramos. En ella se recoge toda la información sobre un tramo, la información del plan de vuelo que regula ese tramo y demás información relacionada. Para cada atributo se muestra el nombre, el tipo de dato y una descripción.

Como ya se ha mencionado, es necesario además almacenar información sobre cada uno de los mensajes ADS-B que forman parte del tramo. La Tabla 3.8 muestra la información que se almacena para cada uno de los mensajes. Principalmente se trata de información de identificación, posicionamiento (latitud, longitud y altitud), velocidad y dirección.

3.2. Métricas

Una métrica es una medida o conjunto de medidas, que permite establecer tendencias de los datos en el tiempo. Se puede decir que una métrica representa una extrapolación de las medidas obtenidas para un valor establecido. En el proyecto AIRPORTS, las métricas han sido calculadas mediante fórmulas, que agregan trayectorias en un volumen y un tiempo definidos generando una medida representativa, según ciertos factores dependientes del tipo de métrica. De entre todas las métricas recogidas en el proyecto, en este trabajo se ha utilizado una métrica [11] que permite medir la eficiencia de un vuelo, utilizando un conjunto de índices de rendimiento clave KPIs (Key Performance Indicator). Estos KPIs están basados en el artículo [4] de Calvo et al, y permiten comparar el tiempo, la distancia y el combustible consumido entre la trayectoria real reconstruida, y otras trayectorias imaginarias o sintéticas.

Para obtener los KPI´s lo que se hace es comparar los tiempos transcurridos (t), las distancias horizontales recorridas (d) y el consumo de combustible (f) de un vuelo (calculado como la diferencia entre el peso de la aeronave al

Nombre	Tipo	Descripción
Leg_id	String()	Unique identifier of each leg. Composed of the callsign of flight, aircraft hexident, the start time and the final time of the leg.
Callsign	String()	Value of the callsign belongs a leg.
Hexident	String()	Hexident of the aircraft that carried out a leg.
Time_start_leg	Long()	Timestamp of the start of the leg.
Time_finish_leg	Long()	Timestamp of the finish of the leg.
Registration	String()	Registration of the aircraft which has flown a leg.
Ifps_id	String()	Identifier of the flight plan that regulates a leg.
Operator	String()	Airline to which it belongs.
Airport_origin	String()	Airport of origin.
Airport_destination	String()	Airport of destination.
Estimated_departure_time	Long()	Estimated take-off time.
Estimated_arrival_time	Long()	Estimated time of landing.
Departure_runway	String()	Runway in the origin.
Arrival_runway	String()	Runway in the destination.
Aircraft_model	String()	The aircraft model.
Aobt	Long()	Actual Off-block time.
Iobt	Long()	Initial Off-block time.
Cobt	Long()	Calculated Off-block time.
Eobt	Long()	Estimated Off-block time.
Lobt	Long()	Late Off-block time.
Cdm_early_ttot	Long()	Time of take-off previously calculated by CDM.
Cdm_ao_ttot	Long()	Time of take-off calculated by the airline operating the flight.
Cdm_atc_ttot	Long()	Time of take-off calculated by the control of air traffic.
Cdm_sequenced_ttot	Long()	Time when calculated take-off the aircraft already in the take-off sequence.
Taxi_time	Int()	Time elapsed since the aircraft came out of the gate to the runway.
Ftfm_consumed_fuel	Int()	Consumption of fuel calculated by Filed traffic flight model.
Ftfm_route_charges	Int()	The route cost calculated in €, by Filed traffic flight model.
Rtfm_consumed_fuel	Int()	Consumption of fuel calculated by Regulated traffic flight model.
Rtfm_route_charges	Int()	The route cost calculated in €, by Regulated traffic flight model.
Ctfm_consumed_fuel	Int()	Consumption of fuel calculated by Computed traffic flight model.
Ctfm_route_charges	Int()	The route cost calculated in €, by Computed traffic flight model.
Scr_consumed_fuel	Int()	Consumption of fuel calculated by Shortest Constrained Route.
Scr_route_charges	Int()	The route cost calculated in €, by Shortest Constrained Route.
Srr_consumed_fuel	Int()	Consumption of fuel calculated by Shortest RAD restrictions Route.
Srr_route_charges	Int()	The route cost calculated in €, by Shortest RAD restrictions Route.
Sur_consumed_fuel	Int()	Consumption of fuel calculated by Shortest Unconstrained Route.
Sur_route_charges	Int()	The route cost calculated in €, by Shortest Unconstrained Route.
Dct_consumed_fuel	Int()	Consumption of fuel calculated by Direct Constraint Route.
Dct_route_charges	Int()	The route cost calculated in €, by Direct Constraint Route.
Cpf_consumed_fuel	Int()	Consumption of fuel calculated by Correlated Positions reports for a Flight.
Cpf_route_charges	Int()	The route cost calculated in €, by Correlated Positions reports for a Flight.

Tabla 3.7: Atributos de un leg. [17]

Nombre	Tipo	Descripción
Leg_id	String()	Identifier of the leg to which belongs the message.
Message_id	String()	Unique identifier generated for each message.
Timestamp	Long()	Time of reception of the ADS-B message by a receiver. Use the format Epoch Unix.
Latitude	Decimal(8,5)	It determines the latitude of the aircraft in the reported position.
Longitude	Decimal(8,5)	It determines the longitude of the aircraft in the reported position.
Altitude	Int()	It determines the altitude of the aircraft in the reported position.
Speed	Int()	It determines the speed the aircraft in the reported position.
VSpeed	Int()	It determines the vertical speed the aircraft in the reported position.
Squawk	String()	It determines the (hexadecimal) squawk code.
Track	Int()	It determines the track angle of the aircraft in the reported position.
Ground	Boolean()	It determines the flight status.

Tabla 3.8: Atributos de un mensaje. [17]

principio de la trayectoria y al final), del vuelo reconstruido, con respecto a las trayectorias sintéticas. Las trayectorias sintéticas utilizadas son: la trayectoria geodésica basada en la trayectoria real volada, la trayectoria basada en el plan de vuelo y la trayectoria geodésica basada en el plan de vuelo. La trayectoria geodésica tal y como se aprecia en la Figura 3.5 es la línea de distancia mínima (más corta) que une dos puntos.

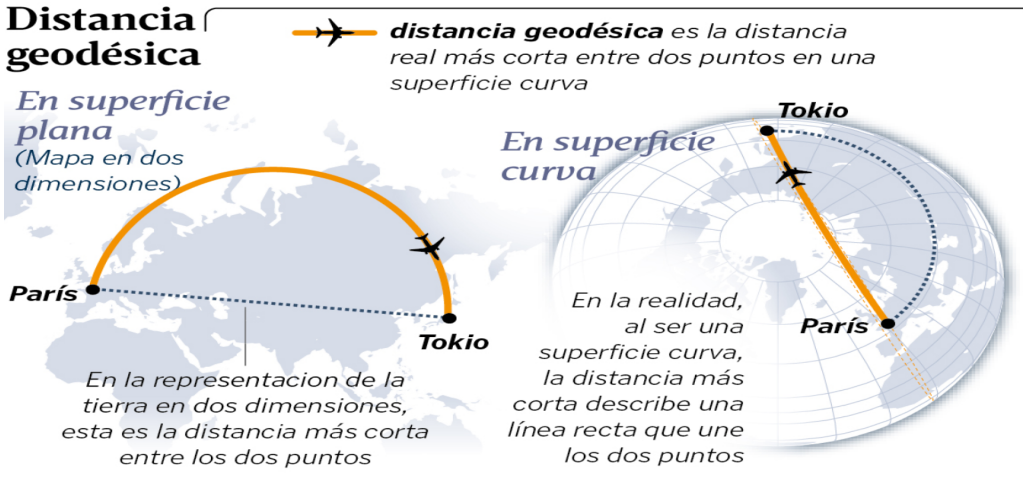

Figura 3.5: Trayectoria Geodésica [3]

3.3. Estado del arte

3.3.1. Herramientas de visualización

La visualización de datos es una de las herramientas más eficaces para dotar de significado a los datos, permitiendo su análisis a cualquier usuario sea cual sea su nivel técnico. La visualización de la información se apoya en la representación gráfica, agilizando y favoreciendo el descubrimiento de tendencias o patrones que ayuden al entendimiento y a la obtención de conclusiones.

Como punto de inicio en el estudio de las herramientas de visualización de datos existentes hasta el momento, se encuentran las hojas de cálculo (MS Excel, LibreOffice Calc, Google Sheets). Se trata de la herramienta de visualización más famosa a nivel mundial, utilizada a diario por millones de usuarios. Su interfaz es muy intuitiva, permitiendo realizar todo tipo de tareas de forma rápida y sencilla. Cuando se desea representar un gran volumen de datos o disponer de opciones de diseño más sofisticadas, surge la necesidad de herramientas y plataformas más avanzadas.

En segundo lugar se encuentran las herramientas asociadas al Business Intelligence (BI). Estas herramientas facilitan la toma de decisiones empresariales estratégicas, mediante mecanismos de gestión, análisis y visualización que ayudan a obtener información y a detectar tendencias a partir de los datos. En un mercado cada vez más exigente, herramientas como Microsoft Dynamics, IBM Cognos Analytics, SAP, Oracle, Sisense entre otras, forman parte de miles de organizaciones.

Otra opción, que requiere conocimientos técnicos, por lo que su uso es más complejo, es la utilización de lenguajes de programación para la visualización de datos. Matlab, Mathematica o R son lenguajes de programación que permiten realizar visualizaciones, sin las limitaciones de las herramientas vistas anteriormente, ya que el usuario es el propio desarrollador de la visualización.

Las opciones para crear visualizaciones son actualmente ilimitadas. En un mundo en el que la web ofrece todo tipo de servicios, que la hacen imprescindible en el día a día de las personas, no es de extrañar que también las visualizaciones tiendan a su integración con la web, mediante las llamadas aplicaciones web. A continuación, se mostrará una breve descripción de algunos ejemplos de estas herramientas:

- Tableau[15], es una plataforma de análisis y visualización de datos intuitiva y muy flexible, capaz de extraer información de utilidad a partir de los datos almacenados. De entre los productos que ofrece Tableau cabe resaltar Tableau Desktop, Tableau Server y Tableau Online. Estas herramientas permiten conseguir información útil de forma ágil y eficaz, ofreciendo cuadros de mandos interactivos, conexión a diversas fuentes de datos, análisis de tendencias y regresiones, mapas, historias, etc.

  Al tratarse de una herramienta muy potente es posible crear visualizaciones impresionantes utilizando grandes cantidades de datos. Debido a sus inmensas cualidades tanto técnicas como visuales, el uso de Tableau está muy extendido formando parte de sectores como educación, comunicación y el farmacéutico.

  En la Figura 3.6 se muestra un ejemplo de cuadro de mandos creado con Tableau Public.

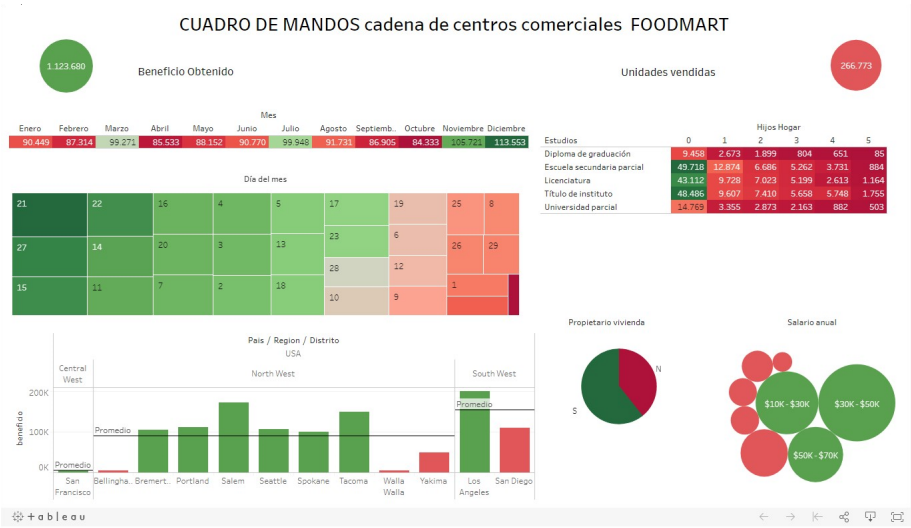

Figura 3.6: Cuadro de Mandos Tableau Public

- Tabulae[16], es una plataforma web de análisis y visualización de datos. Se trata de una herramienta multidispositivo que no requiere de clientes de escritorio, ni de conocimientos de programación y que permite integrar rápidamente cuadros de mando e informes dinámicos

[15]https://www.tableau.com/es-es [Junio 2019]
[16]https://tabulaeapp.com/ [Junio 2019]

en cualquier producto software. Tabulae dispone de un sistema de permisos para controlar los accesos, garantizando un uso seguro y permite incluir funcionalidades de BI utilizando procesadores de texto WYSIWYG (What You See Is What You Get).

En la Figura 3.7 se muestra un cuadro de mandos creado en Tabulae.

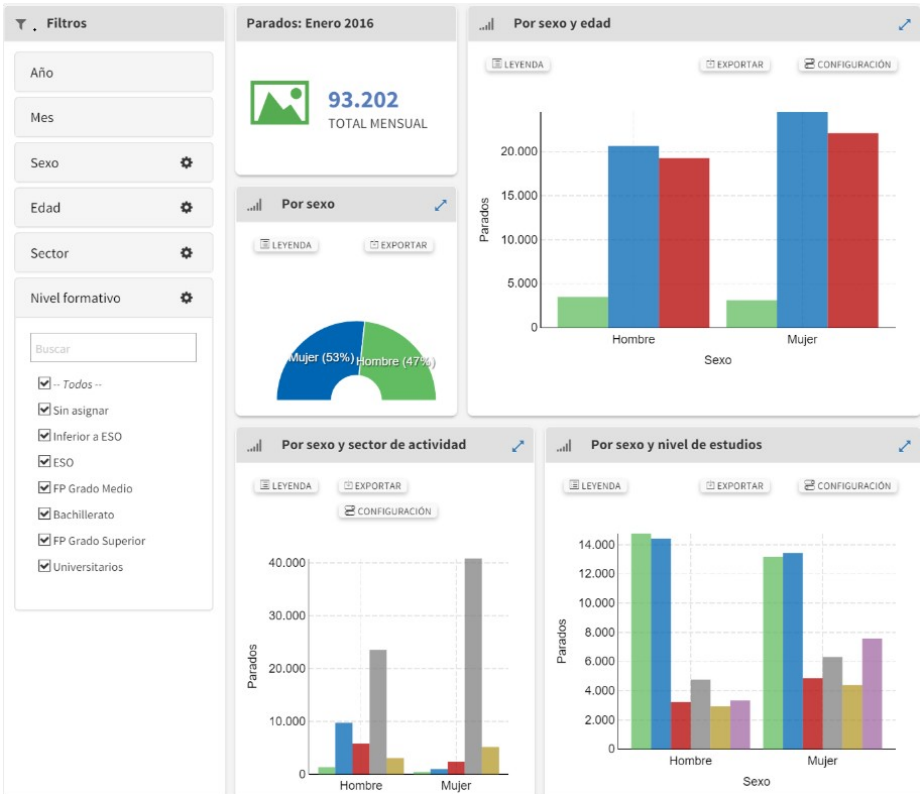

Figura 3.7: Cuadro de Mandos en Tabulae

- QlikTech[17], es una de las compañía pioneras en dotar de significado a la información a través de la transformación de datos. A través de sus productos QlikView y Qlik Sense, QlikTech brinda mediante visualizaciones innovadoras la posibilidad de analizar y explotar grandes cantidades de datos para poder tomar decisiones estrategias. El motor asociativo de Qlik permite unificar distintos orígenes de datos, ofreciendo una visión unificada de la información.

  QlikView es una plataforma de Business Intelligence rápida, sencilla y con una gran potencia visual, permitiendo a las organizaciones detectar

[17]https://www.qlik.com/us/ [Junio 2019]

grandes áreas de mejora e innovación. Se trata de una herramienta que ofrece al usuario amplias opciones de personalización ya sea visuales como de seguridad. Gracias a su tecnología propia, QlikView facilita no solo la creación de informes y cuadros de mando sino también su uso y mantenimiento.

Qlik Sense es una herramienta de visualización, mucho más sencilla e intuitiva. Es una herramienta muy potente, multidispositivo, ampliamente accesible y claramente enfocada a ofrecer visualizaciones innovadoras, configurables y adaptables a las necesidades del usuario final. En su página web se pueden consultar múltiples demostraciones, que permiten al usuario hacerse una idea de su gran potencial tanto visual como funcional.

En la Figura 3.8 se muestra un Cuadro de mandos realizado con Qlik Sense

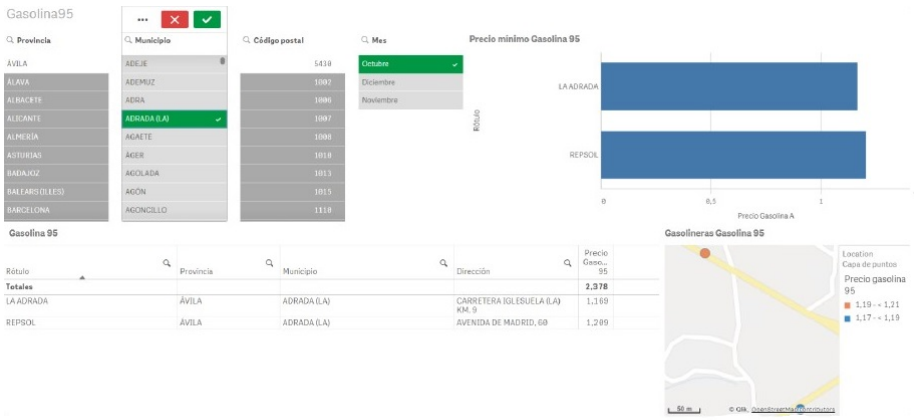

Figura 3.8: Cuadro de mandos realizado con Qlik Sense

- CARTO[18], es un servicio de computación en la nube que proporciona funcionalidades de SIG (Sistemas de Información Geográfica) además de instrumentos para la implementación de mapas. Es una herramienta sencilla pero potente, muy utilizada para la llamada Inteligencia de Localización, que permite mediante la visualización aportar valor a la información obtenida a través de datos de posicionamiento. Además, ofrece sus propios flujos de datos que permiten completar la información del usuario, y ayudar a la toma de decisiones. Su uso se ha extendido rápidamente, facilitando la creación de mapas en publicaciones web.

[18]https://carto.com/ [Junio 2019]

En la Figura 3.9 se muestra una visualización de un mapa en CARTO con el tiempo de carga.

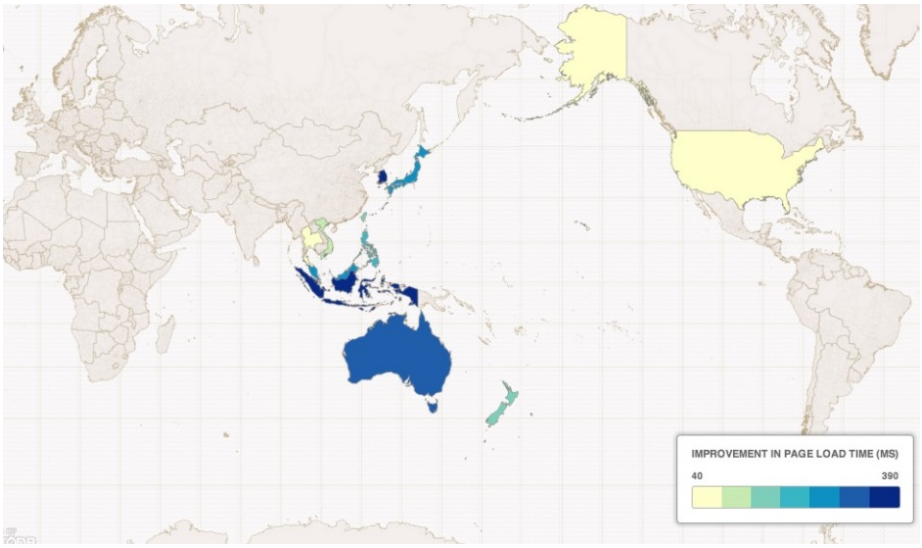

Figura 3.9: Visualización de mapa en CARTO

Finalmente, y aunque se ha llevado a cabo un recorrido muy completo, por las herramientas de visualización actuales más relevantes, es importante remarcar, que a la hora de decidir qué herramienta de visualización se debe utilizar, será necesario realizar un estudio adaptado a las necesidades del cliente, y de las herramientas que ofrezca el mercado en ese momento dado. Para facilitar esta elección, el grupo Gartner[19] elabora cada año el llamado Cuadrante Mágico de Gartner, en el que recoge las organizaciones que han obtenido la excelencia a través de los servicios tecnológicos que ofrecen. El cuadrante está compuesto por dos ejes. El eje horizontal que refleja la capacidad de visión del mercado, evaluando elementos claves en una organización como son el conocimiento de las necesidades del mercado, innovación y todo tipo de estrategias (geográficas, de ventas, de marketing, etc). El eje vertical proporciona un análisis de la capacidad de ejecución, entendiendo esta capacidad como la destreza de los proveedores de ser capaces de cubrir las necesidades adaptadas al mercado, evaluando habilidades como la capacidad de ejecutar ventas, de adaptación al mercado, de cumplir objetivos y compromisos etc.

[19]https://www.gartner.com/en [Junio 2019]

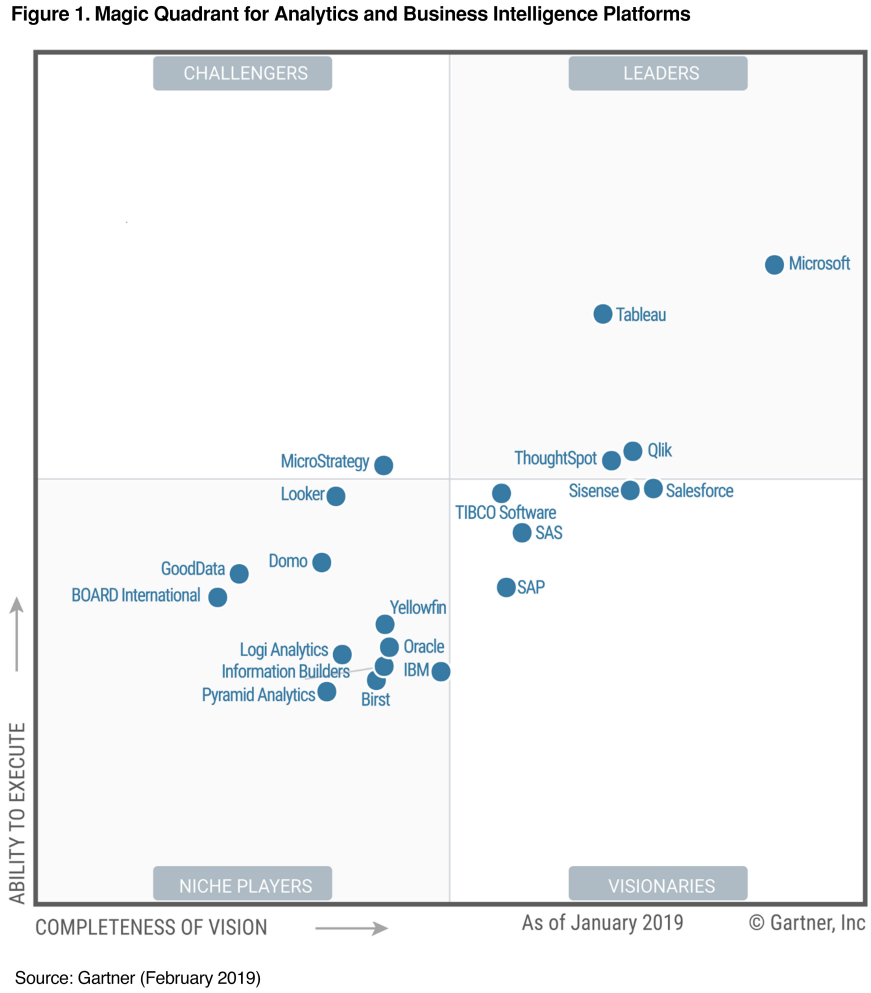

Figura 3.10: Cuadrante Mágico de Gartner [8]

El cuadrante se divide en cuatro zonas que atienden tanto a la tipología de la compañía, como a la de sus productos. Estas zonas son:

- Líderes (leaders), en este sector se encuentran los proveedores que mejores resultados han obtenido, del análisis de la combinación entre la capacidad de visión de mercado y la capacidad de ejecución.
- Aspirantes (challengers), se centran únicamente en una aspecto del mercado, lo que limita su oferta de productos.
- Visionarios (visionaries), este tipo de proveedor engloba a los provee-

dores que ofrecen servicios adaptados al mercado pero no son capaces de ejecutarlos por falta de medios.

- Jugadores de nicho (niche players), en la última posición nos encontramos con los proveedores que no destacan en ninguna de las dos capacidades.

En la Figura 3.10 se puede apreciar como tanto Tableau como Qlik ocupan puestos líderes, en el Cuadrante Mágico de Gartner.

3.3.2. Herramientas de visualización utilizadas en el sector aéreo

Enfocando el estudio de herramientas de visualización hacia el sector aéreo, nos encontramos como norma, que la forma de rastrear la información de vuelos se realiza a través de la propia página web del aeropuerto. El principal problema de estás páginas web es el desfase existente en las actualizaciones del estado del avión, así como también la falta de transparencia cuando se trata de lo que está haciendo en un momento concreto. Por ejemplo, el estado del vuelo en la web, puede aparecer como 'Despegado', pero en realidad la aeronave aún podría estar en tierra, haciendo cola en la pista de despegue.

Sistemas como Flightradar24[20], FlightwAware[21] y OpenSky[22] utilizan una estrategia diferente: reciben los datos del vuelo directamente de los aviones. Esto es posible gracias a la tecnología ADS-B, que como ya se ha visto, hoy en día es utilizada por la mayoría de aviones modernos.

- Flightradar24, es un servicio global de seguimiento de vuelos en tiempo real que opera sobre miles de aviones. Es capaz de rastrear más de 175.000 vuelos, de más de mil aerolíneas de todo el mundo. Flightradar24 está disponible actualmente a través de la web y para dispositivos móviles.

  Su web ofrece una herramienta de visualización sencilla y flexible que facilita opciones de filtrado utilizando múltiples parámetros como son entre otros, el número de vuelo (IATA o ICAO), el número de registro del avión, el código 'squawk', el nombre de la aerolínea y el modelo del avión. A través de este tipo de búsquedas, se puede obtener información

[20] https://www.flightradar24.com/ [Junio 2019]
[21] https://es.flightaware.com/ [Junio 2019]
[22] https://opensky-network.org/ [Junio 2019]

detallada de un vuelo concreto, como puede ser la localización del avión en el momento actual, o también información de las características de la propia aeronave. La versión de pago proporciona opciones de búsqueda mucho más sofisticadas.

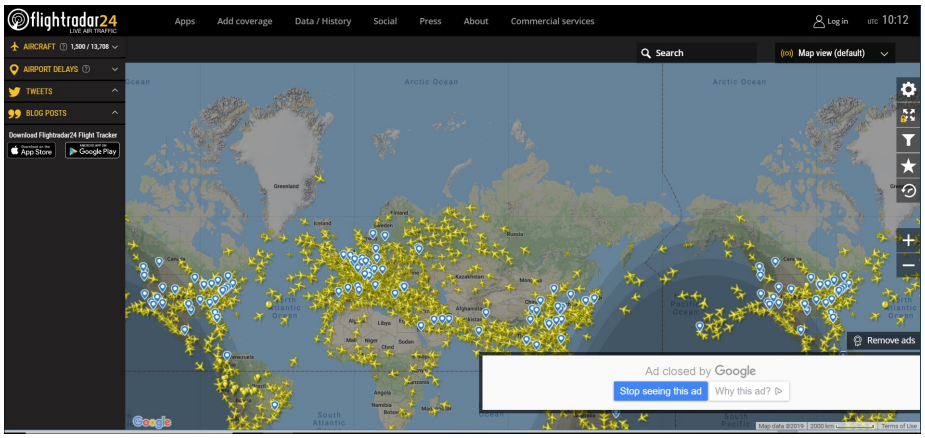

Figura 3.11: Cuadro de mandos Flightradar24

- Flightwaware, es una compañía que ofrece numerosos servicios relacionados con ATM, entre ellos información de surveillance obtenida gracias a su red ADS-B (que consta de más de 20.000 receptores, distribuidos por 195 países).

  Proporciona una herramienta de seguimiento de vuelos que ofrece mapas en tiempo real, el estado de los vuelos y los retrasos que se producen en los aeropuertos para todo tipo de vuelos. Es compatible con dispositivos móviles desde los cuales se puede consultar un mapa con las rutas de todos las aeronaves que se encuentren volando en un determinado momento.

  Es una página muy bien organizada, que ofrece la opción de conocer todos los detalles de un vuelo, proporcionando un mapa actualizado con la ruta del avión. Además una vez el vuelo ha finalizado se puede consultar multitud de información que incluye gráficos detallados de los parámetros recogidos durante el vuelo y fotografías de la propia aeronave.

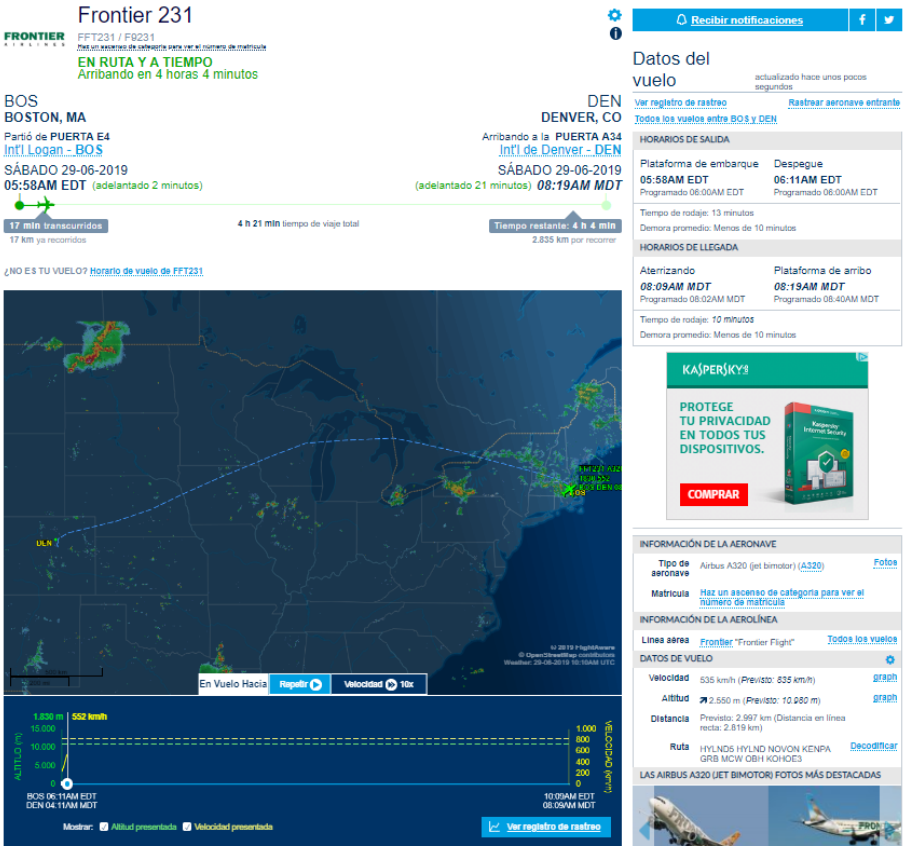

Figura 3.12: Cuadro de mandos FlighAware

- OpenSky, es una red de receptores que obtiene datos de vigilancia del tráfico aéreo de forma continua. OpenSky mantiene los datos en bruto recopilados para siempre y los hace accesibles a los investigadores. Además permite solicitar gratuitamente un receptor ADS-B, a cambio de colaborar enviando la información recibida a sus servidores. La red OpenSky Network dispone de más de diez billones de mensajes ADS-B y Mode S recogidos de más de 1.000 sensores en todo el mundo.

  Su web ofrece OpenSky Explorer, una herramienta ágil y sencilla a través de la cual, es posible acceder a un mapa en el que aparecen representados los aviones con su posición en tiempo real. Acotando las búsquedas por zonas reducidas, se muestran etiquetas con el nombre del vuelo, que permiten seleccionarlo para consultar, el trazado en el mapa del itinerario que recorrerá el avión, así como toda la información relativa al vuelo. Esta herramienta proporciona mecanismos de filtra-

do por diferentes parámetros como callsign, icao, altitud, velocidad, ciudad, etc. enriqueciendo la experiencia de usuario.

Ofrece además, un panel en el que consultar todas las alertas permitiendo realizar un seguimiento detallado del estado de los vuelos y la opción de embeber el mapa de OpenSky en cualquier sitio web a través del código iframe. Para usuarios registrados proporciona también la opción de consultar datos históricos, mediante un práctico calendario.

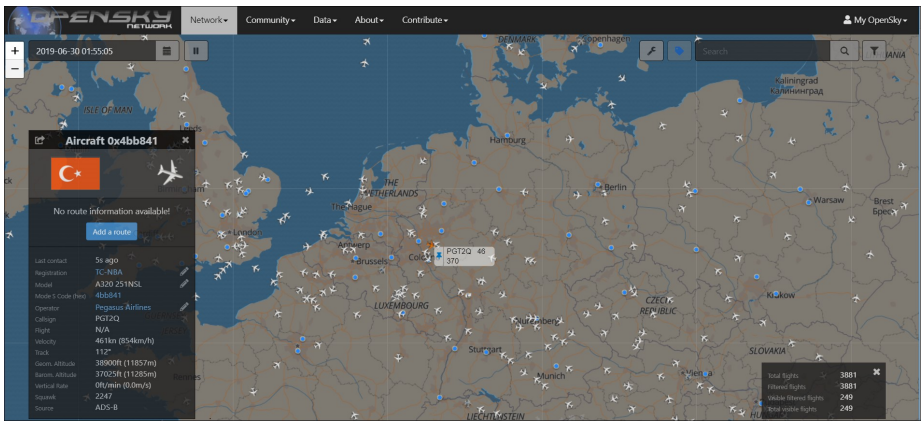

Figura 3.13: Cuadro de mandos OpenSky Explorer

El estudio de estas herramientas de visualización enfocadas al sector aéreo, resulta de gran utilidad, ya que permite generar una visión global de como se encuentra el panorama actual, ofreciendo un punto de partida para este trabajo. Todas estas herramientas son muy potentes, y permiten la consulta de información aérea en tiempo real. Cabe remarcar que el cuadro de mandos que se desea crear, no se basa únicamente en la consulta de información aérea, si no que trata de ir un paso mas allá, permitiendo extraer conclusiones de alto valor, enfocadas a la mejora de los sistemas de gestión aérea, mediante el uso de métricas y trayectorias sintéticas.

Estudio de alternativas Python

En este capítulo se explicarán las características principales del lenguaje de programación Python y se llevará a cabo un estudio de las herramientas más utilizadas para construir cuadros de mandos utilizando Python, seleccionando finalmente la más adecuada para realizar el dashboard de información aérea propuesto.

4.1. Python

El lenguaje de programación Python fue desarrollado a finales de los 80 por el ingeniero holandés Guido Van Rossum. Su nombre tiene su origen en los humoristas Monty Python de los que Guido era un gran seguidor. Python nació como un hobby para Guido, nunca se dedicó en exclusiva a él, compaginándolo con empleos en grandes empresas como Google o Dropbox. Desde 2001 es la Python Software Foundation[23] una asociación sin fines de lucro, la que se encarga de promover el lenguaje y favorecer su progreso así como ayudar al crecimiento de su comunidad de programadores. Aunque Guido ha seguido implicado en su desarrollo y en las decisiones de diseño, en 2018 ha decidido abandonar su papel de supervisor del lenguaje.

Python en un lenguaje de software libre, cuya licencia Python Software Foundation License (PSFL) es permisiva, del tipo de la licencia BSD (Distribución de software Berkeley) y compatible con la licencia GPL (Licencia Pública General de GNU). Se trata de una licencia que no es copyleft, es decir consiente la modificación del código fuente o la realización de obras derivadas, sin la obligación de que tanto las modificaciones como las nuevas obras sean a su vez de código abierto.

[23]https://www.python.org/psf/ [Junio 2019]

El lenguaje Python destaca por ofrecer bibliotecas estándar muy completas, donde la mayor parte de las tareas de programación más utilizadas ya están implementadas, lo que incrementa notablemente la productividad de los programadores. Python se caracteriza por ser un lenguaje interpretado, es decir no se compila, se interpreta en tiempo de ejecución, a diferencia de otros lenguajes como C o Java, lo que provoca que tarde más tiempo en la ejecución. Esta lentitud en la ejecución es despreciable en la web debido a las capacidades de los procesadores actuales, siendo más significativa cuando la ejecución se realiza en dispositivos móviles.

Se trata de un lenguaje multiparadigma, es decir combina características de distintos modelos de programación, siendo un lenguaje orientado a objetos que además incluye propiedades de otros tipos de programación como son la programación funcional, procedural, imperativa, reflexiva, etc. Es compatible a su vez, con distintos Sistemas operativos y proporciona opciones de tipado dinámico, así como en sus últimas versiones, de tipado estático (a partir de la 3.5). Un factor destacable de Python es que cuenta con una gran comunidad de desarrolladores muy activa y colaborativa, lo cual proporciona un valor añadido a la hora de realizar cualquier aplicación en Python.

Tim Peters, colaborador durante muchos años de Python, redactó en 1999 una colección de 19 principios básicos conocidos como el "Zen de Python" [2], que sintetizan la filosofía a utilizar para desarrollar con Python. El "Zen de Python" se enfoca fundamentalmente en la importancia de un código simple, reutilizable y legible, lo cual hace de Python un lenguaje muy fácil de aprender y de mantener. Es precisamente por todas estas cualidades, que es un lenguaje ampliamente presente en distintas disciplinas y en grandes empresas como Google, Netflix e Instagram. A continuación, se muestran algunos ejemplos de herramientas que o bien utilizan Python o bien se basan en él:

- Desarrollo web, a través de frameworks como Django o Pyramid
- Protocolos de internet como FTP, IMAP, HTML, XML
- Herramientas de computación científica y matemática como SciPy
- Herramientas de inteligencia de negocio como Odoo.
- Herramienta de scripting facilitando la creación de scripts
- Infraestructura y despliegues

- Herramientas de scraping y crawling como Scrapy[24]

Una mención aparte merece la relación de Python con el Big Data. Python es uno de los lenguajes más usados en soluciones Big Data, ya que es un lenguaje fácil de aprender y rápido de codificar, no tiene limitación a la hora de procesar datos, y gracias a la plataforma Anaconda ha aumentado sus tiempos de desarrollo y de ejecución. En Big Data la plataforma de código abierto más famosa que existe es Hadoop, la cual es perfectamente compatible con Python. Paquetes como Pandas, Numpy, Scikit-learn, Tensorflow o iPython hacen de Python un poderoso lenguaje enfocado al análisis y a la ciencia de datos.

Como hemos visto, Python sirve para casi todo dentro del universo de los datos, uno de sus puntos fuertes son las librerías dedicadas a la visualización de datos, ya que no dejan de crecer y evolucionar. Es precisamente debido a este crecimiento de herramientas de visualización en Python, por lo que surge la necesidad de realizar un estudio para encontrar la alternativa que más se ajuste a nuestros requerimientos.

4.2. Matplotlib

No podemos comenzar a hablar de sistemas de visualización de datos en Python sin mencionar la librería Matplotlib[25], ya que a pesar de tener más de una década, sigue siendo la biblioteca de Python más utilizada en ciencia de datos.

Con Matplotlib se pueden realizar, mediante la utilización de muy pocas instrucciones, todo tipo de gráficos como por ejemplo, gráficos de líneas, de barras, de áreas, o diagramas de dispersión. Matplotlib está formado por dos módulos esenciales, pyplot que es una colección de funciones que hacen que se comporte de forma similar al programa de cálculo Matlab[26] y pylab que combina pyplot con Numpy[27] (librería que proporciona funciones para operar con estructuras de datos potentes, como matrices y matrices multidimensionales).

Una de sus mayores ventajas, es que permite un muy buen rendimiento en la representación de grandes cantidades de datos tan necesario actualmente, debido al exceso de información a tratar en casi cualquier ámbito. Además

[24]https://scrapy.org/ [Junio 2019]
[25]https://matplotlib.org/ [Junio 2019]
[26]https://www.mathworks.com/ [Junio 2019]
[27]https://www.numpy.org/ [Junio 2019]

ofrece la posibilidad de añadir mapas geográficos, incluyendo Cartopy[28] y gráficos en 3D, incluyendo Mplot3d[29]. También permite exportar las imágenes con los gráficos resultantes, lo cual la hace muy atractiva para su uso en artículos o revistas digitales.

Si bien matplotlib de entre todas las librerías disponibles, es la más completa y permite además crear todo tipo de gráficos, tiene como principal inconveniente, que los gráficos creados por Matplotlib son muy básicos (apariencia de los 90), lo que conlleva la necesidad de realizar grandes cambios estéticos para obtener un resultado más actual y sofisticado.

En la Figura 4.14 se muestra ejemplos de gráficas disponibles en Matplotlib

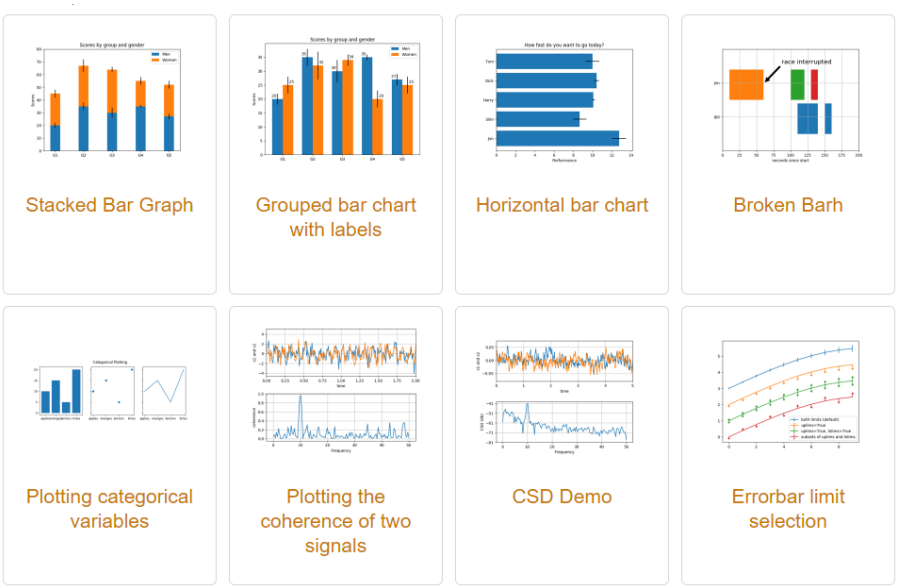

Figura 4.14: Gráficas disponibles con Matplotlib

4.3. Seaborn

La librería Seaborn[30] es una librería de visualización de datos Python basada en Matplotlib y estrechamente integrada con las estructuras de datos de Pandas[31].

[28] https://scitools.org.uk/cartopy/docs/latest [Junio 2019]
[29] https://matplotlib.org [Junio 2019]
[30] https://seaborn.pydata.org/ [Junio 2019]
[31] https://www.learnpython.org/es/Pandas%20Basics [Junio 2019]

Seaborn ofrece una interfaz que permite elaborar gráficos estadísticos mucho más atractivos. El principal propósito de Seaborn es conseguir, a través de la visualización, dotar de significado a los datos, para poder deducir comportamientos o tendencias. Incluye varios temas y distintas paletas de colores, que proporcionan la modernidad y flexibilidad que le falta a Matplotlib. Además incorpora múltiples funciones para trabajar con conjuntos de datos, visualizar matrices y posibilita la gestión de series estadísticas.

Como inconveniente principal, se puede destacar que aunque genera gráficas más atractivas que Matplotlib, los tipos de gráfica que ofrece son más limitados.

La Figura 4.15 muestra ejemplos de gráficos creados con Seaborn.

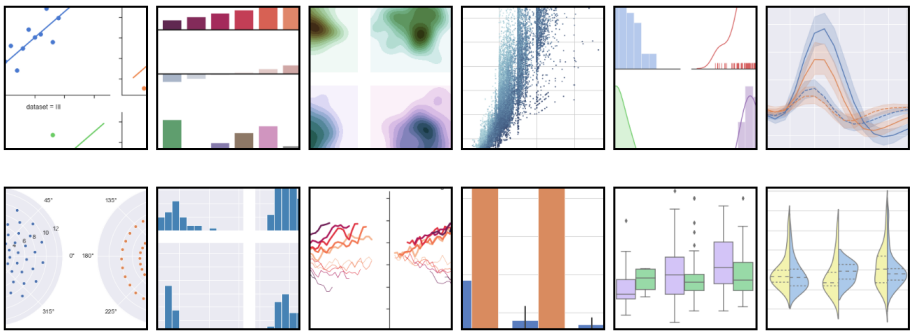

Figura 4.15: Gráficos disponibles en Seaborn

4.4. Bokeh

Bokeh[32] se distingue de Matplotlib o Seaborn en el hecho de que es una librería de visualización interactiva, ideal para crear rápida y fácilmente gráficos dinámicos e imágenes estáticas, cuadros de mandos y aplicaciones web (ya que genera código HTML/JS listo para ser añadido en cualquier sitio web). Bokeh esta patrocinado de forma fiscal por NumFOCUS[33], una organización sin ánimo de lucro, dedicada a apoyar a la comunidad de investigación, datos y computación científica en proyectos de código abierto.

La librería Bokeh soporta el uso de cuadernos de Jupyter y Zeppelin, por lo que es muy recomendada si se desea combinar texto, con cálculos

[32]https://bokeh.pydata.org/en/latest/ [Junio 2019]
[33]https://numfocus.org/ [Junio 2019]

y gráficos en un mismo documento. Se recomienda su uso sobre todo en sitios web donde lo importante sea mostrar datos, de una forma atractiva y funcional. Su objetivo principal es ofrecer una herramienta capaz de proporcionar un buen rendimiento sobre volúmenes de datos grandes o streaming, en la construcción de gráficos precisos, atractivos e interactivos. Como inconveniente, se puede mencionar que para interacciones demasiado sofisticadas serán necesarios conocimientos técnicos de Javascript.

Bokeh ofrece tres niveles operacionales distintos:

- El nivel superior, que facilita métodos para crear gráficos muy fácilmente.
- El nivel intermedio, que funciona al mismo nivel que Matplotlib, controlando la construcción de cada gráfico.
- El nivel inferior, en el que es necesario definir cada componente del gráfico, y que está enfocado a desarrolladores.

Figura 4.16: Distintos gráficos disponibles en Bokeh

4.5. Dash (by Ploty)

Dash[34] es un framework libre bajo licencia MIT de Python, para la creación de aplicaciones web analíticas. Está construido sobre Ploty[35], React[36] y

[34]https://plot.ly/products/dash/ [Junio 2019]
[35]https://plot.ly/ [Junio 2019]
[36]https://es.reactjs.org/ [Junio 2019]

Flask[37]. Ploty es una librería libre destinada a la construcción de gráficas interactivas funcionales, y responsivas que ofrece más de 30 tipos diferentes de gráficas, entre los que se incluyen mapas. La librería de javascript React, permite interactuar con la visualización en tiempo real, sin requerir que la página sea recargada.

Dash es una herramienta íntegramente personalizable, que proporciona una interfaz muy fácil de utilizar permitiendo añadir todo tipo de controles que permitan un alto grado de interactivididad, facilitando al usuario el filtrado de la información. Las aplicaciones construidas en Dash son muy rápidas y ligeras, al publicarse en la Web, se aprovecha toda la potencia de HTML, CSS e incluso Javascript. Se compone de 3 paquetes principales:

- Dash Core Components, contiene un conjunto de componentes de alto nivel como son selectores, gráficos, desplegables, etc.
- Dash HTML Components, proporciona todas las etiquetas HTML como clases Python fáciles de usar.
- Dash DataTable, es el componente más moderno. Permite generar tablas interactivas que soportan formato condicional, clasificación, filtrado, etc.

La Figura 4.17 y la Figura 4.18 representan ejemplos de gráficos disponibles en Plotly y ejemplos de cuadros de mandos realizados con Dash respectivamente.

[37]http://flask.pocoo.org/ [Junio 2019]

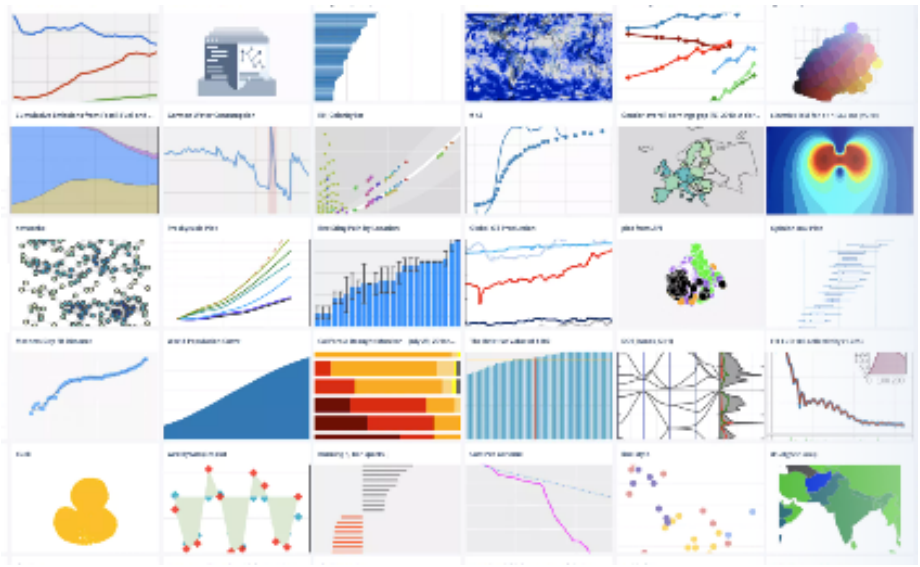

Figura 4.17: Gráficos disponibles en Plotly

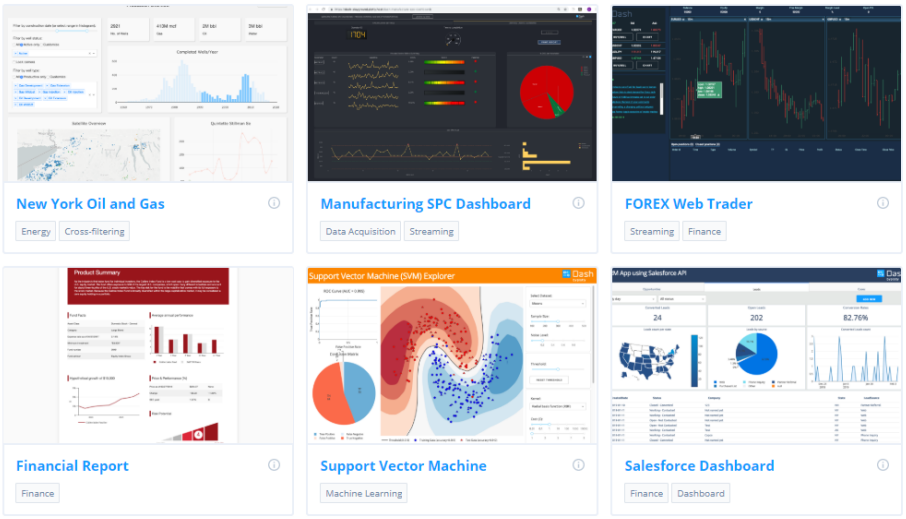

Figura 4.18: Cuadros de mandos creados con Dash

4.6. Conclusión

Tras valorar las distintas alternativas se ha llegado a las siguientes conclusiones:

Aunque Matplotlib es una de las librerías más utilizadas en la actualidad, no es adecuada para nuestro caso de estudio, ya que proporciona gráficos que no son interactivos, además estéticamente, se ha quedado un poco desfasada perdiendo atractivo. El proceso de incrustación de los gráficos generados para poder crear un cuadro de mandos puede ser un proceso largo y complejo.

Por otro lado, se ha descartado también la librería Seaborn, porque aunque mejora la estética con respecto a Matplotlib sigue sin ofrecer gráficos interactivos. Esta librería es ideal para crear gráficos estadísticos sofisticados, que incluyan el uso de colores para aportar significado.

En cuanto a la decisión entre Bokeh y Dash ha sido más complicada ya que ambas librerías ofrecen potentes gráficos interactivos y una interfaz muy atractiva.

Finalmente nos hemos decantado por Dash, ya que Dash es más rápido para grandes volúmenes de datos que Bokeh, además al ser sus componentes muy intuitivos, resulta más fácil de programar, y con un menor número de líneas. Utiliza Plotly lo que lo hace muy poderoso, al igual que el uso de React en la interfaz, facilitando la adición de componentes en caso de que sea necesario. Otro punto a tener en cuenta en esta decisión es la consistencia en las interacciones. Bokeh se comporta de forma inconsistente cuando se filtra a la vez con menús desplegables y seleccionando datos en los propios gráficos. No existiendo este problema en los cuadros de mando creados en Dash.

A pesar de todas las ventajas mencionadas de Dash, uno de los factores determinantes además de la necesidad obligatoria de interacción es la integración de la librería dentro de cuadernos Jupyter. Bokeh proporciona soporte nativo de cuadernos Jupyter, y aunque Plotly también los soporta, la librería Dash no. Investigando sobre la posibilidad de incluir Dash dentro de cuadernos Jupyter, se ha descubierto que ha sido creada, recientemente, una librería que permite la integración de Dash con Jupyter, lo cual ha terminado de inclinar la balanza.

Propuesta de Arquitectura

En este apartado se realizará una descripción de la arquitectura propuesta para construir un cuadro de mandos, con los datos recibidos del proyecto AIRPORTS. Se detallarán los componentes de la arquitectura y las responsabilidades de cada uno, así como nuestra propuesta de implementación. Finalmente mostraremos un ejemplo del flujo de datos a lo largo de nuestra arquitectura.

5.1. Descripción de la Arquitectura

Aunque la idea inicial de este trabajo era utilizar de entrada al cuadro de mandos, directamente los ficheros resultantes del proyecto AIRPORTS sin ningún tipo de procesamiento, durante el desarrollo del mismo y dado el gran volumen de datos a tratar, surgió la necesidad de plantear una arquitectura, que fuera capaz de conseguir filtrar la información a mostrar en el cuadro de mandos, de una forma rápida y eficiente. Dicha arquitectura ha sido elaborada en torno a dos enfoques principales, por un lado la segmentación de la información de los mensajes, organizándolos por fechas y por su identificador de tramo (leg), y por otro el almacenamiento de los datos de los tramos en una tabla que facilitase el filtrado del cuadro de mandos. Estos mismos enfoques se extendieron posteriormente para los datos de métricas y trayectorias reconstruidas.

La Figura 5.19 muestra una visión general de la arquitectura propuesta junto con las tecnologías utilizadas para la implementación de cada una de ellas.

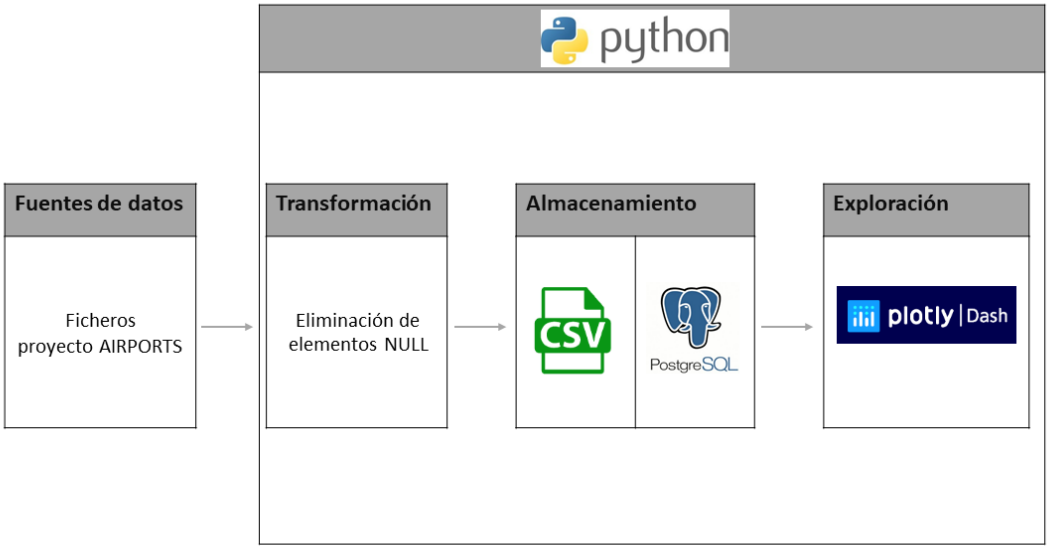

Figura 5.19: Arquitectura Propuesta

5.2. Componentes

Los componentes principales que forman nuestra propuesta de arquitectura son:

5.2.1. Fuentes de datos

El primer componente representa las fuentes de datos desde las que nuestra arquitectura recibe la información a procesar. Estas fuentes de datos son los ficheros csv generados por el proyecto AIRPORTS. Se utilizará un fichero config.txt que guardará información de los nombres de los ficheros a tratar, sus rutas (relativas) y las rutas donde se crearán las carpetas de segmentación.

Los ficheros de entrada (por defecto), serán los siguientes:

- legs.csv. Fichero que contiene la información de cada uno de los tramos para un día concreto.

- messages.csv. Fichero que contiene la información relativa a los mensajes enviados por cada identificador de tramo en un día concreto.
- metrics.csv. Fichero que almacena la información de KPIs y Trayectorias Sintéticas.
- reconstructed_trajectories.csv, syn_trj_up.csv, syn_trj_geo_fr.csv, syn_trj_geo_fp.csv. Ficheros que almacenan respectivamente, la trayectoria reconstruida, la sintética para el plan de vuelo, la sintética sobre la reconstruida y la sintética respecto al plan de vuelo.

5.2.2. Transformación

El segundo componente representa la transformación de datos a bajo nivel. Este bloque es responsable de la transformación de los datos de cada uno de los ficheros recibidos. Aunque los datos ya llegan procesados del proyecto AIRPORTS antes de su almacenamiento, ha sido necesario realizar unas pequeñas transformaciones. La primera y más importante fue la sustitución de las cadenas "NULL" y "\n" por la cadena vacía. También ha sido necesario eliminar las comillas dobles que acompañaban a las cadenas de caracteres. Al eliminar estos caracteres del fichero csv, se ha conseguido aprovechar todo el potencial que Postgres ofrece a la hora de realizar inserciones, consiguiendo importar los datos directamente a la base de datos desde los ficheros csv, en lugar de realizando inserciones individuales.

Para la correcta lectura de los datos de los ficheros a través de la librería Pandas de Python, ha sido necesario eliminar de las cabeceras los sufijos que acompañaban a los nombres de los campos, como ha sido el caso del fichero de legs teniendo que eliminar el sufijo "legs.", ya que al llevar el carácter '.' se producían comportamientos incorrectos. También ha sido necesario añadir la cabecera al fichero de mensajes, ya que no había sido incluida en la exportación.

5.2.3. Almacenamiento

Como tercer componente, nos encontramos el bloque de almacenamiento de los datos, que es el punto de entrada de la información, este componente es el más importante ya que toda la arquitectura se desarrolla entorno a él.

Este componente de almacenamiento garantiza la persistencia, no permitiendo modificaciones sobre los datos originales. Además, se comportará de forma distinta en función del tipo de fuente de datos que se reciba. En el caso del fichero de messages.csv se llevará a cabo una segmentación del

fichero original, extrayendo los datos de cada identificador de tramo a un fichero distinto y organizando todos ellos, dentro de una carpeta etiquetada con la fecha de captura del mensaje. El mismo proceso se llevará a cabo con los 4 ficheros de trayectorias reconstruidas salvo por la peculiaridad, de que los ficheros resultantes se agruparan en un directorio extra anterior al de la fecha, que determine el tipo de trayectoria entre uno de los cuatro existentes.

A la hora de almacenar la información del fichero de tramos y de métricas, se creará una tabla en una base de datos PostgreSQL[38], que almacenará la información de los ficheros. De igual modo se creará una tabla nueva que registre la existencia de cada identificador de tramo, en cada uno de los cuatro ficheros de trayectorias sintéticas, mediante el uso de un booleano.

5.2.4. Exploración

El cuarto y último componente de nuestra arquitectura será el que llamaremos de exploración y se centra en la extracción del valor de los datos almacenados. Implementa el cuadro de mandos en sí, objetivo de este trabajo y especialmente útil para los científicos de datos. Para ello, este componente accederá a la información almacenada, ya sea directamente de los ficheros segmentados o mediante consultas a la base de datos y mediante la herramienta de visualización seleccionada, en este caso Dash, mostrará los resultados al usuario final.

5.3. Ejemplo de flujo de datos

Veamos a continuación, un ejemplo del flujo de datos a través de los distintos componentes de la arquitectura propuesta. Se ha decidido explicar el procedimiento de almacenamiento de las trayectorias reconstruidas, por ser el más completo, al requerir tanto el almacenamiento mediante segmentación de ficheros, como por medio de la base de datos.

El la primera fase de Fuentes de datos disponemos de 4 ficheros:

- constructed_trajectories.csv
- syn_trj_up.csv
- syn_trj_geo_fr.csv

[38]https://www.postgresql.org/ [Junio 2019]

- syn_trj_geo_fp.csv

Los datos de estos ficheros serán cargados mediante un cuaderno de Python y se realizará la fase de transformación de los valores NULL por la cadena vacía. Una vez los datos han sido transformados, se pasará a la etapa de almacenamiento, en este caso concreto el almacenamiento se llevará a cabo de dos maneras, por un lado se realizará la segmentación de los ficheros en función del identificador de tramo, organizándolos en carpetas según el tipo de trayectoria y la fecha, tal y como se muestra en la Figura 5.20

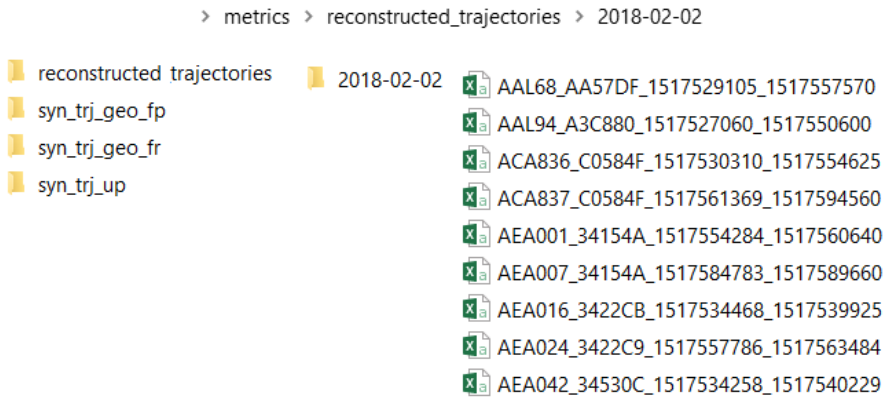

Figura 5.20: Segmentación de Ficheros de Trayectorias Reconstruidas

Por otro lado y para acelerar el filtrado del cuadro de mandos, se ha creado una tabla en la base de datos, en la cual se almacena para cada identificador de tramo en una fecha, si tiene o no registros en cada uno de los cuatro ficheros correspondientes, mediante un booleano, en la Figura 5.21 se muestra la estructura de la tabla en PostgreSQL

SYNTHETIC_TRAJECTORY

General | Columns | Constraints | Advanced | Parameters | Security | SQL

Inherited from table(s) — Select to inherit from...

Columns

	Name	Data type	Length	Precision	Not NULL?	Primary key?
	leg_id	character varying			No	No
	date	date			No	No
	reconstructed_trajectories	boolean			No	No
	syn_trj_geo_fp	boolean			No	No
	syn_trj_geo_fr	boolean			No	No
	syn_trj_up	boolean			No	No

Figura 5.21: Tabla de Trayectorias Sintéticas

Finalmente, el siguiente paso del proceso, sería la exploración, que se llevaría a cabo mediante el uso de Python, la librería Dash y estableciendo una conexión a la base de datos y a los ficheros segmentados, mostrando por medio de menús y representaciones gráficas la información en el cuadro de mandos. En la figura Figura 5.22 se muestra el resultado final de este ejemplo concreto.

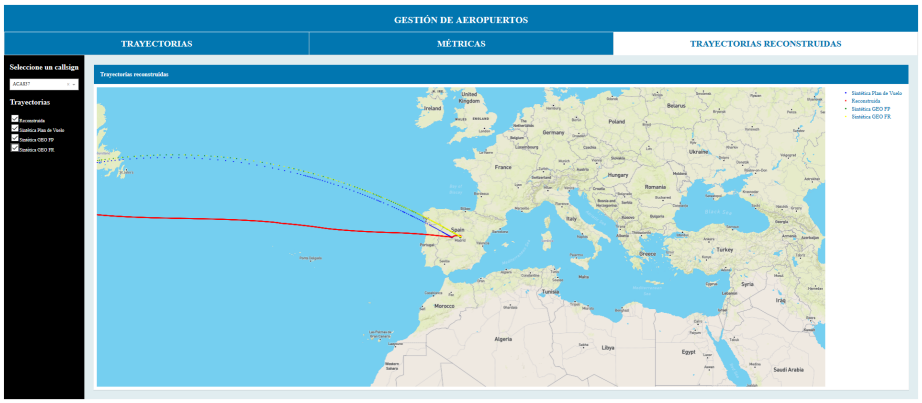

Figura 5.22: Exploración de datos de Trayectorias Sintéticas

Implementación del dashboard

A lo largo de este capítulo se hará un recorrido por los pasos realizados para la elaboración del cuadro de mandos. Se describirá la instalación y configuración de las herramientas necesarias y se explicará como se han realizado los distintos componentes que formarán el cuadro de mandos.

6.1. Instalación y configuración del entorno

6.1.1. Descripción del sistema informático utilizado

Para la realización de este trabajo se ha utilizado un ordenador portátil Intel Core i7-6700HQ CPU @ 2.60GHz 2.59GHZ con 16 GB de RAM y Sistema Operativo Windows 10 de 64 bits. También ha sido necesaria conexión a internet.

6.1.2. Instalación del entorno de Desarrollo Anaconda

Como entorno de desarrollo Python se ha utilizado Anaconda, distribución de Python que incluye los paquetes actualmente más utilizados en el ámbito de la ingeniería, la ciencia y las matemáticas, como pueden ser Matplotlib, NumPy o SciPy. Anaconda es considerada una herramienta de software libre, ya que se distribuye bajo la licencia BSD de tres cláusulas. Se trata de una herramienta fácil de utilizar y de instalar, ya que la Suite de Anaconda facilita de forma considerable la instalación del entorno e incluye Jupyter Notebooks.

Jupyter Notebooks, es un entorno interactivo que permite trabajar con código Python e incorporar en un mismo documento texto, gráficas, imágenes o bloques de código. La mayor ventaja de utilizar Jupyter para la creación del cuadro de mandos es que posibilita la integración de la programación con la visualización, agilizando así el desarrollo, al no tener que estar utilizando un terminal, un editor de texto y una ventana de visualización.

Para realizar su instalación lo único que se debe hacer es acceder a su página web[39], y descargar la versión compatible con el sistema operativo del que disponga el equipo, en este caso Anaconda 2019.03 para Windows que incluye Python 3.7. La instalación es muy sencilla y solo debemos seguir las indicaciones del asistente. Una vez finalizada la instalación, podremos abrir un cuaderno de Jupyter Notebooks a través del enlace a Jupyter Notebook que aparece en la ventana de los programas de Windows dentro de Anaconda.

6.1.3. Instalación de Pandas

Pandas es una librería destinada a facilitar el análisis de datos, que proporciona potentes estructuras de datos, capaces de convertir los datos en bruto en datos útiles para el análisis (por ejemplo, tablas). Pandas, aunque fue creada inicialmente como una alternativa a la utilización de hojas de cálculo, actualmente se ha convertido en una muy poderosa librería, para tareas muy avanzadas sobre los datos, en el lenguaje de programación Python. Durante el desarrollo de este trabajo, el uso de esta librería ha resultado de gran utilidad permitiendo, entre otras, operaciones de ordenación, filtrado y concatenado sobre los datos.

Ofrece tres estructuras de datos:

- DataFrame, es una estructura de datos con forma de tabla ordenada, formada por filas y columnas. Es la estructura de datos que más se ha utilizado a lo largo de este trabajo, ya que permite tanto recoger datos en forma de tabla de consultas a la base de datos, como directamente de los ficheros.

- Series, es la segunda estructura de datos que ofrece Pandas. Es un objeto unidimensional semejante a la columna de una tabla.

- Panel, es una estructura que funciona como un contenedor de datos de tres dimensiones.

[39]https://www.anaconda.com/distribution/ [Junio 2019]

Anaconda ya trae por defecto, incorporado el paquete de Pandas por lo tanto no debemos hacer ningún paso extra para su instalación. Aunque sí será necesario realizar la importación de la librería desde el cuaderno.

6.1.4. Instalación de Dash

Una vez tomada la decisión de utilizar Dash y más en concreto a través de Jupyter Notebooks, hubo que atajar la limitación de que Dash por defecto no permitiera su utilización dentro de los cuadernos de Jupyter. Para conseguir integrar Dash dentro de los cuadernos de Jupyter, y después de un largo periodo de investigación, se encontraron las siguientes tres alternativas:

- Uso de la extensión Jupyter Lab[15]. Se trata de una extensión que permite desarrollar aplicaciones interactivas, Dash en Jupyter, pero que tiene como inconveniente que no es soportada por el Sistema Operativo Windows, razón por la cual tuvo que ser descartada.

- Creación de la aplicación Dash sin utilizar los cuadernos, para luego añadirla como un iframe. En el siguiente enlace[12] se muestra un ejemplo de uso. Esta opción tampoco era la que se buscaba en el desarrollo de este trabajo, ya que con ella en realidad no se estaba trabajando con un cuaderno, sino se estaba creando el código de forma externa y después se estaba representando en un cuaderno.

- Uso de la librería jupyter-plotly-dash 0.2.1[7] que permite ejecutar el código de Dash dentro de un cuaderno de Jupyter, creando la aplicación utilizando la clase JupyterDash en lugar de la clase por defecto, dash.Dash. Tras varias pruebas y ver que su funcionamiento se adaptaba a las necesidades del trabajo, esta opción fue la seleccionada.

Por tanto, para poder utilizar Dash dentro de los cuadernos, se debe instalar la librería jupyter-plotly-dash. Además dado que Dash utiliza internamente los gráficos de Plotly, se debe instalar también Plotly. Ambas librerías se instalan fácilmente en cuestión de segundos, mediante la instrucción "conda install" seguida del paquete a instalar, por medio de la consola Anaconda Prompt.

6.1.5. Instalación de Postgres

Para instalar Postgres SQL se debe acceder a la página oficial de PostgreSQL[40] y descargar la versión compatible con nuestro Sistema Operativo, en este caso la 11.4 para Windows x86-64. El instalador de PostgreSQL proporciona un asistente que nos guía por la instalación de una forma rápida y sencilla. Cabe destacar la necesidad de prestar atención a la hora de introducir la contraseña para el súper usuario "postgres", ya que será necesario para poder iniciar sesión a la hora de administrar la base de datos.

Una vez terminada la instalación de PostgreSQL, será necesario realizar la conexión con Jupyter Notebook, para ello se debe instalar a través de Anaconda Prompt, la librería sqlalchemy psycopg2.

6.1.6. Otras herramientas

Mapbox

A la hora de integrar los mapas necesarios para reflejar la información de localización en el cuadro de mandos, existen nuevamente varias alternativas a valorar. Las opciones más utilizadas actualmente son: la API de Google Maps, el SDK de Mapbox y el framework MapKit de Apple. Tras valorar el uso de estas opciones, Mapbox ha sido considerada la mejor alternativa para nuestros propósitos, debido en gran medida a su simplicidad y flexibilidad.

En Mapbox[41] todo se puede personalizar, desde colores hasta las capas que se desean mostrar en el mapa. Mapbox proporciona una serie de estilos ya creados (Dark, Light, etc.), pero también ofrece la posibilidad de crear estilos personalizados por completo a través de la herramienta Mapbox Studio. Los estilos se incorporan en la aplicación incluyendo únicamente una línea de código, lo que permite generar mapas acordes a las necesidades especificas de cada escenario. En nuestro caso se ha utilizado el estilo que ofrece Mapbox //styles/mapbox/outdoors-v9 y que se muestra en la Figura 6.23.

[40] http://www.postgresql.org/download/windows [Junio 2019]
[41] https://www.mapbox.com/ [Junio 2019]

Figura 6.23: Estilo Utilizado para Mapas

Se trata además de una herramienta de código abierto, que dispone de más de 400 repositorios en Github. Su núcleo está desarrollado en C++ y basado en mapas vectoriales, lo que proporciona rapidez y poco peso a las visualizaciones. Para añadir el mapa generado con Mapbox Studio tan sólo es necesario incluir el token de acceso. Este token se obtiene de forma muy sencilla, tras registrarse en la página de Mapbox (es gratis para los primeros 50,000 usuarios activos).

Paletton

Para la elección de los colores del diseño del dashboard se ha utilizado la herramienta Paletton[42], ya que el color es una propiedad sumamente importante en la web, ya no solo porque es lo primero que se percibe al entrar, sino porque esta demostrado que influye en los sentimientos, condicionando las acciones de los usuarios. En el mundo empresarial la elección de un color debe hacerse con suma atención, más relevancia cobra si cabe, cuando el escenario es un producto web, ya que la elección de un color adecuado, puede influir directamente en un incremento de las ventas.

En nuestro caso como el uso del cuadro de mandos está relacionado con el espacio aéreo, se ha optado por el color azul que transmite libertad, eficacia, confianza, calma y seguridad. Este color esta íntimamente relacionado con el progreso y las tecnologías, y es uno de los colores más utilizados en la web, ya que no agota la vista y aporta connotaciones positivas que alientan al usuario a quedarse e interactuar con el sitio. Se ha decido combinarlo con el color blanco para crear un entorno agradable y serio.

[42]https://paletton.com/ [Junio 2019]

Firefox Developer

Firefox Developer[43] es una versión de Firefox integrada, que ofrece multitud de herramientas para desarrolladores. El resultado final de nuestro dashboard no deja de ser una página web, que contiene código HTML y CSS, por lo que se ha utilizado Firefox Developer para localizar posibles errores y para diseñar los estilos de la web. Las herramientas más utilizadas de entre las que Firefox Developer proporciona, han sido el explorador de código HTML y CSS, la consola de depuración y el editor de estilos CSS en vivo.

6.2. Construcción del dashboard

Como paso previo a la construcción del dashboard en Dash, se llevó a cabo un prototipo con la herramienta Balsamiq [44], que ofrece una versión de prueba de 30 días gratuita en la nube. Un prototipo es un boceto, que tiene como función principal ofrecer al usuario una idea del diseño de la aplicación que se desee realizar, de forma rápida y muy visual. Dado que se decidió distribuir nuestro cuadro de mandos en tres pestañas diferentes se ha realizado un boceto por cada una de ellas. En la Figura 6.24 se muestra el prototipo realizado para la pestaña de Trayectorias, en la Figura 6.25 se muestra el prototipo para la pestaña de Métricas y por último en la Figura 6.26 se muestra el prototipo de la pestaña de Trayectorias Reconstruidas.

Debido a la decisión de utilizar Dash dentro de los cuadernos de Jupyter, nos hemos tenido que enfrentar a ciertas limitaciones que se comentarán en detalle a lo largo de las siguientes secciones.

[43] https://www.mozilla.org/es-ES/firefox/developer/ [Junio 2019]
[44]https://balsamiq.cloud/ [Septiembre 2019]

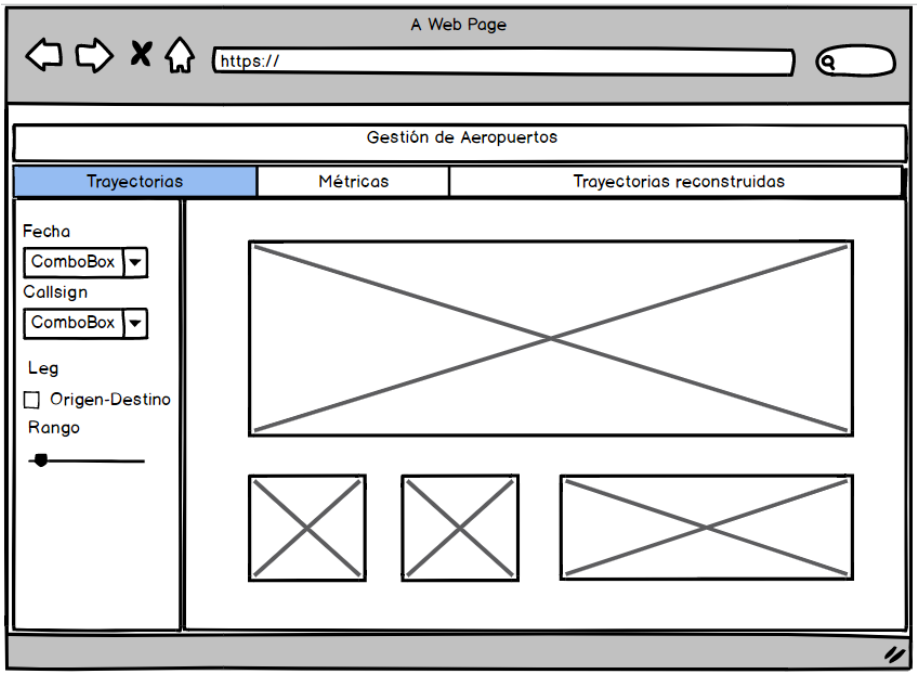

Figura 6.24: Prototipo Pestaña de Trayectorias

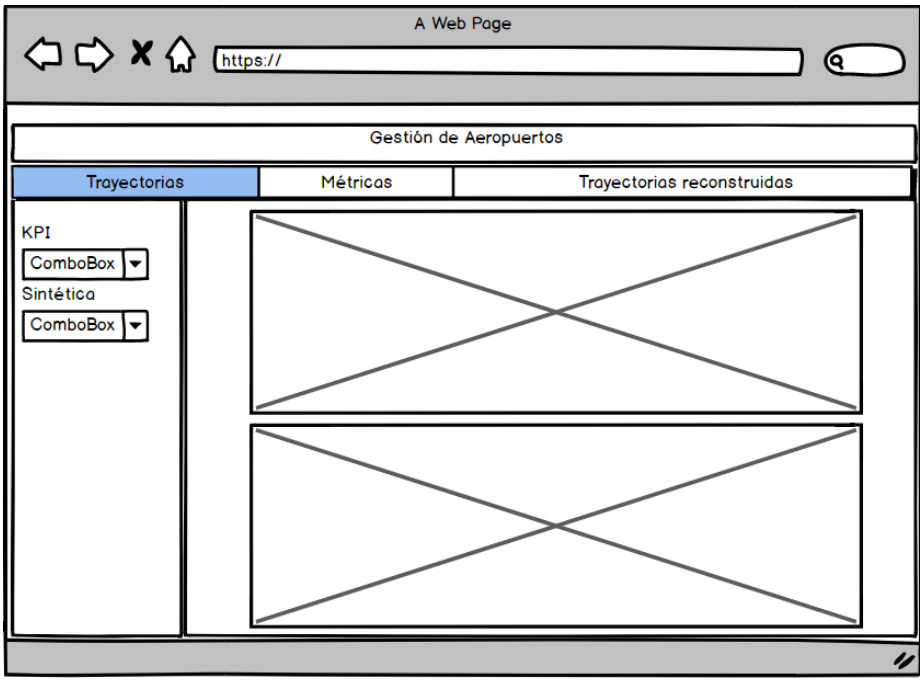

Figura 6.25: Prototipo Pestaña de Métricas

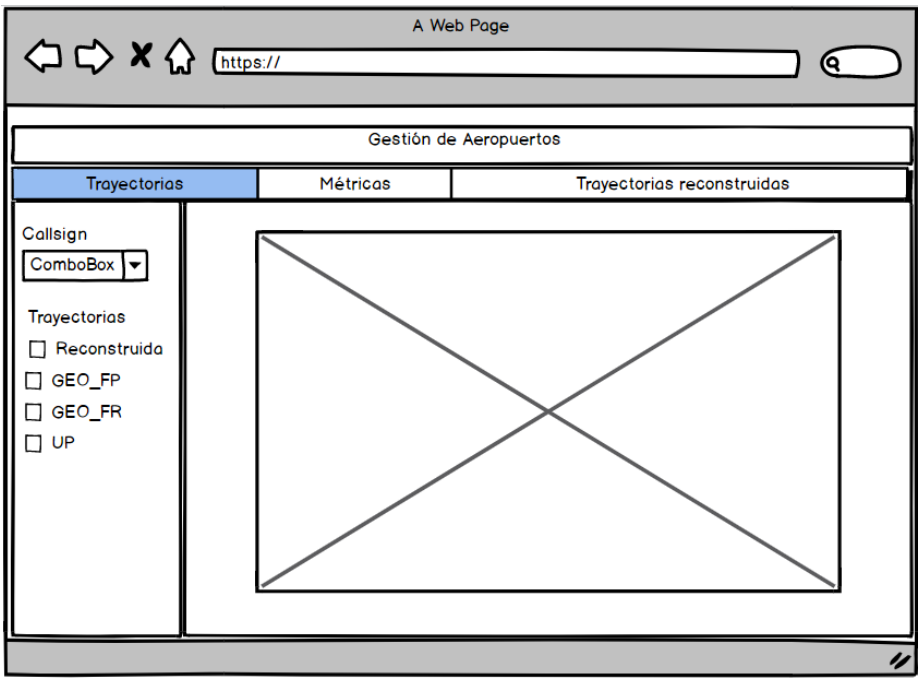

Figura 6.26: Prototipo Pestaña de Trayectorias reconstruidas

6.2.1. Pestañas

Para ofrecer al usuario un entorno accesible, intuitivo y ordenado, se ha decidido dividir el cuadro de mandos en 3 pestañas: Trayectorias, Métricas y Trayectorias reconstruidas. Para realizar estas pestañas se ha utilizado el componente tabs, que ofrece Dash.

En primer lugar se ha creado un componente tabs y dentro de este se han creado 3 componentes de tipo tab, siguiendo la estructura que se muestra a continuación.

Figura 6.27: Pestañas del Cuadro de Mandos

En la Figura 6.27 se muestra la cabecera del cuadro de mandos seguida por las 3 pestañas, mostrando en color blanco la pestaña seleccionada, por defecto la de Trayectorias. En la Figura 6.28 se muestra del código en Dash

correspondiente. En el siguiente capítulo se verá que información proporciona cada una de estas tres pestañas.

```
dcc.Tabs(id="tabs", value='tab1', children=[
    dcc.Tab(label='Trayectorias', value='tab1'),
    dcc.Tab(label='Métricas', value='tab2'),
    dcc.Tab(label='Trayectorias Reconstruidas', value='tab3')
])
```

Figura 6.28: Código creación de pestañas

6.2.2. Selectores

Para dotar de interactividad al cuadro de mandos se han utilizado tres tipos distintos de selectores, menús desplegables, checks y un control deslizante. Cabe resaltar que para poder utilizar todos estos componentes, es necesario importar la librería dash_core_components.

Menús desplegables

Se han utilizado desplegables en las tres pestañas existentes en el cuadro de mandos. En la primera pestaña de trayectorias, como se muestra en la Figura 6.29, se ofrece la opción de filtrar por Fecha mediante un desplegable con las fechas disponibles, al seleccionar una fecha inmediatamente se actualiza el segundo desplegable con los callsigns correspondientes a esa fecha.

Todos los desplegables se han construido utilizando un código similar al que se muestra en la Figura 6.30

Siendo options una lista de diccionarios en los que se definen la etiqueta (label) y el valor (value), de cada uno de los elementos del desplegable, y value, una cadena de caracteres que debe coincidir con el valor de alguna de las opciones definidas en el diccionario en caso de que se desee, que aparezca algún campo seleccionado por defecto.

En algunas ocasiones, ha sido necesario crear los desplegables con el campo options vacío y rellenarlo después al ser dependiente de selecciones previas. En estos casos el componente se declara de la misma forma indicada anteriormente, con la diferencia de que options será una lista vacía, y al llevarse a cabo el evento correspondiente, será cuando se rellene el valor de options mediante una llamada de retorno.

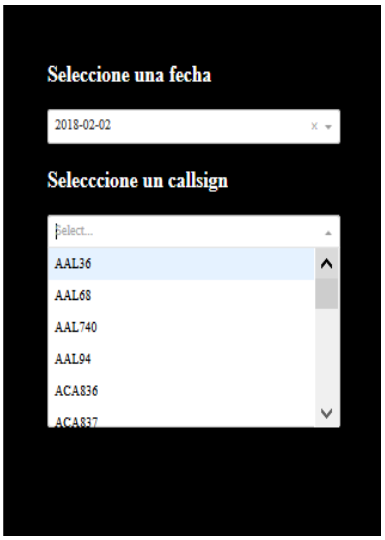

Figura 6.29: Desplegables del Cuadro de Mandos

```
dcc.Dropdown(
    id='dropdown-callsign',
    options=[
        {'label': 'AAL68_AA57DF_1517529105_1517557570','value': 'AAL68_AA57DF_1517529105_1517557570, 2018-02-02'},
        {'label': 'AEA058_3444CE_1517534322_151754082','value': 'AEA058_3444CE_1517534322_151754082, 2018-02-02'},
        {'label': 'AEA1040_3453C1_1517559194_15175684','value': 'AEA1040_3453C1_1517559194_15175684, 2018-02-02'}
    ],
    value=''
)
```

Figura 6.30: Código de desplegable

Lista de checks

Se han utilizado listas de checks (checkslist) para poder seleccionar los tramos que se desean pintar sobre el mapa en la pestaña de trayectorias, y para los tipos de trayectorias reconstruidas, que se desean mostrar sobre el mapa, en la pestaña de trayectorias reconstruidas.

Inicialmente, ambos componentes se han ocultado y se han creado con el campo options vacío, ya que dependen tal y como muestra en la Figura 6.31, de las selecciones que se hagan en el desplegable previo de callsigns. Una vez seleccionado un callsign, se rellena el campo options mediante una llamada de retorno y se muestra el componente oculto.

En el atributo value de la propiedad options, del checklist de tramos de la pestaña de trayectorias, hemos añadido el identificador de tramo y la fecha, de forma que cuando se marque un tramo, sepamos con el valor

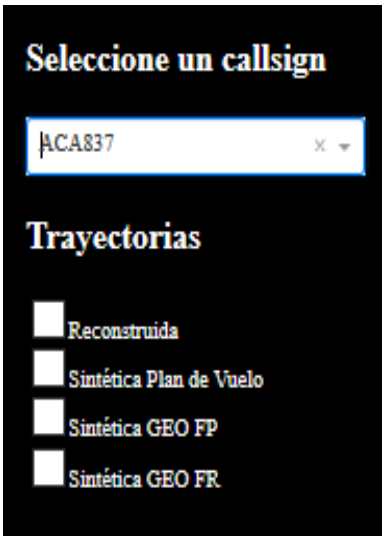

Figura 6.31: Lista de checks del Cuadro de Mandos

del campo value, en que directorio fecha y de que fichero identificador de tramo debemos obtener los datos de los mensajes a mostrar. Para la pestaña de trayectorias reconstruidas, se ha seguido el mismo procedimiento con la salvedad, de que además de la fecha y del identificador se almacena el tipo de trayectoria reconstruida seleccionada, ya que en la estructura de ficheros de las trayectorias reconstruidas, previamente al directorio con la fecha existe un directorio con el nombre de la trayectoria correspondiente.

La Figura 6.32 muestra la forma genérica de crear un checklist.

```
dcc.Checklist(
    id='checklist-legs',
    options=[
        {'label': 'KJFK-LEMD', 'value': 'AAL68_AA57DF_1517529105_1517557570, 2018-02-02'},
        {'label': 'CYYZ-LEMD', 'value': 'AEA1040_3453C1_1517559194_1517568450, 2018-02-02'},
    ],
    value=[ 'AAL68_AA57DF_1517529105_1517557570, 2018-02-02', 'AEA1040_3453C1_1517559194_1517568450, 2018-02-02']
)
```

Figura 6.32: Código Lista de Checks

Control deslizante

El control deslizante (RangeSlider) tal y como se muestra en la Figura 6.33, se ha utilizado en la pestaña de Trayectorias, para permitir al usuario filtrar un rango de horas de los mensajes recogidos, para un tramo previamente seleccionado.

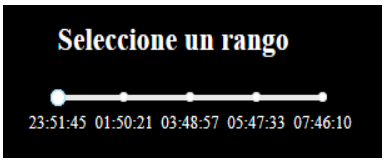

Figura 6.33: Lista de checks del Cuadro de Mandos

Inicialmente se ha creado con valores vacíos de forma que al seleccionar un tramo se han rellenado los campos min, max y marks con los valores mínimo y máximo de las horas de los mensajes en milisegundos y unas marcas con horas intermedias. En la Figura 6.34 se muestra un ejemplo de código para crear el control deslizante.

```
dcc.RangeSlider(
    id='range-slider',
    min=1517572720,
    max=1517576929,
    marks={
        1517572720: {'label': '11:58:40'},
        1517576929: {'label': '13:08:49'}
    },
    value=[]
)
```

Figura 6.34: Código Control Deslizante

6.2.3. Gráficos

Para la creación del cuadro de mandos de entre los múltiples gráficos que ofrece Dash se han utilizado mapas, gráficas de puntos, un diagrama de cajas y un violín. A continuación, veremos cómo se ha realizado cada uno de ellos.

Mapas

Tal y como ya se ha explicado, se ha utilizado un token generado a través de Mapbox Studio para incluir el mapa. Lo que se ha hecho es devolver un elemento figure con dos componentes: data, que inicialmente aparece vacío, y layout, que es en el que se indica como se debe pintar el mapa, indicando los estilos que se desean aplicar, el token, y añadiendo distintas propiedades como son la elección de mostrar la leyenda, o donde se desea situar el centro del mapa. En la Figura 6.35 se muestra el mapa utilizado tanto en la pestaña de Trayectorias como en la de Trayectorias reconstruidas.

Figura 6.35: Mapa utilizado en el Cuadro de Mandos

Cuando se selecciona algún checkbox es cuando se rellena el elemento data del figure con los puntos y la leyenda que se desea mostrar. En la Figura 6.36 se muestra un ejemplo de como se rellenan los datos del mapa de trayectorias, obteniendo los datos de un dataframe pandas messages y teniendo en cuenta que hay que indicar el tipo de gráfico que vamos a representar mediante la instrucción go.Scattermapbox.

```
data = [go.Scattermapbox( lat=messages["latitud"], lon=messages["longitud"],
                                        mode='markers', hoverinfo='text',
                                        marker={'symbol': "circle", 'size': 5, 'color':color},
                                        text='Hora: '+messages['hora']+'<br>Longitud: '+messages['longitud']
                                        +'<br>Latitud: '+messages['latitud'] +'<br>Altitud: '
                                        +messages['altitud'],
                                        name=name_final)]

```

Figura 6.36: Código para pintar una trayectoria en el mapa

Gráficas de puntos

Los gráficos de puntos se utilizan para representar datos cuantitativos de un modo ordenado. Los gráficos de puntos utilizan puntos, para proyectar datos dentro de unos ejes establecidos. Un gráfico de puntos es parecido a un gráfico de líneas, prescindiendo de las líneas que unen los puntos. En la Figura 6.37 se muestra un ejemplo de una de las gráficas del cuadro de mandos.

Este tipo de gráficos se construyen de la misma forma que un mapa, es decir, mediante un elemento Figure. En el atributo layout se definen los estilos tanto para el gráfico en sí como para el fondo, y los márgenes que

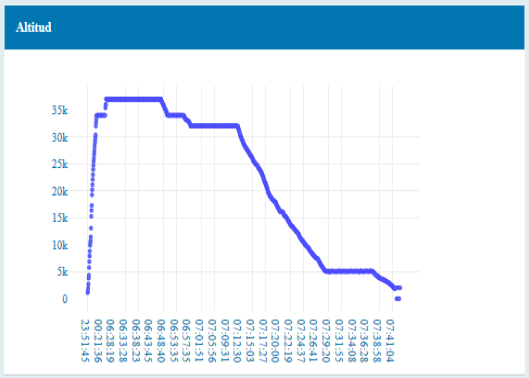

Figura 6.37: Gráfico de puntos del Cuadro de Mandos

se desean aplicar. En el data, al igual que con el mapa, se debe indicar el tipo de gráfico a representar, en este caso go.Scatter y la columna del dataframe de la que se deben coger los valores para los ejes, así como los estilos a aplicar para los puntos, y el texto de la leyenda. En la Figura 6.38 se muestra un ejemplo del código para generar los puntos.

```
1 data= [go.Scatter(x=messages["hora"] , y=messages["altitud"],text=messages["altitud"],
2                    mode='markers',opacity=0.7,
3                    marker={'size': tamanio_punto,'line': estilo_punto,name=name_final})]
4
```

Figura 6.38: Código utilizado para gráfico de puntos

Gráfica de Cajas-Bigotes y Gráfica Violín de la pestaña de métricas

Un diagrama de Caja-Bigotes (boxplot) es una representación gráfica capaz de describir a la vez, diversas características importantes, como la dispersión y la simetría. Este diagrama muestra sobre un rectángulo, los valores mínimo y máximo, los tres cuartiles y la mediana de los datos a representar. Los gráficos de cajas muestran información muy básica, por lo que se ha decidido complementarlo con un diagrama de violín. Un diagrama de Violín, es una versión más actual de los diagramas de cajas, tratando de trazar la distribución real de los datos. Se trata de auténticas gráficas de densidad, que se giran para favorecer su interpretación. En la Figura 6.39 se muestra el gráfico de cajas y violín representado en la pestaña de métricas del cuadro de mandos.

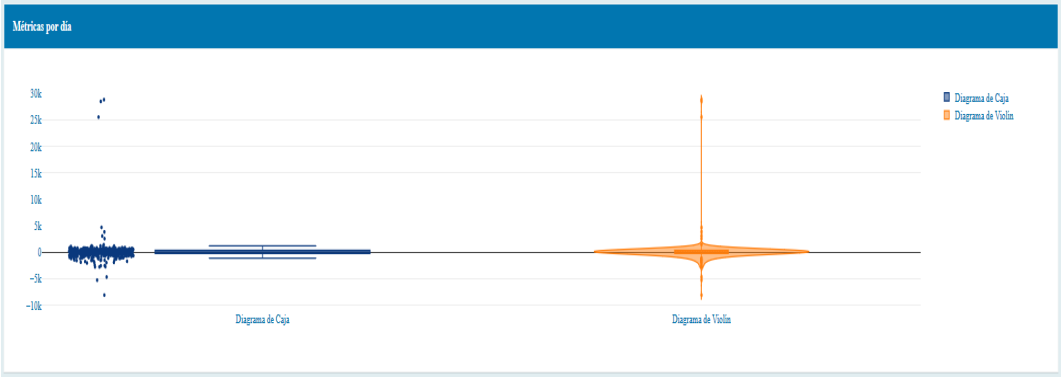

Figura 6.39: Gráfico de cajas y violín del Cuadro de Mandos

Este tipo de gráficos se construyen de la misma forma que los anteriores, es decir, mediante un elemento Figure. En el atributo layout se definen los estilos al igual que en los gráficos de puntos. En el atributo data, nuevamente, se debe indicar el tipo de gráfico a representar, en este caso al tratarse de dos gráficos en uno, lo único que se debe hacer, es añadir dos elementos a data uno con la definición de cada gráfico. Para el gráfico de caja se debe utilizar go.Box y para el gráfico de violín go.violin.

En la Figura 6.40 se muestra un ejemplo del código para generar ambos gráficos y añadirlo a data.

```
data1= go.Box(y=new_df['campo'],boxpoints='all',marker = dict(color = 'rgb(9,56,125)'),
                        line = dict(color = 'rgb(9,56,125)'),name='Diagrama de Caja')
data2= go.Violin(
            y= new_df['campo'],
            name='Diagrama de Violín',
            box= dict(visible=True),
            meanline=dict(visible=True)
            )
return {"data": [data1,data2], "layout": layout}

```

Figura 6.40: Código utilizado para gráfico de cajas y violín

6.2.4. Tablas

Dash ofrece un componente propio para realizar tablas, este componente se ha utilizado en la pestaña de trayectorias para realizar las tres tablas que se muestran en la figura Figura 6.41

Para utilizar este componente lo primero que se debe hacer es importar la librería dash_table. A continuación se muestra el código genérico para construir una tabla, lo primero que se debe definir es la propiedad columnas

OPERADOR DE VUELO	FUEL
LEMD-LRCV:WZZ	LEMD-LRCV:6256

MODELO DE AERONAVE	PISTA DE ATERRIZAJE DE ORIGEN/DESTINO
LEMD-LRCV:A320	LEMD-LRCV:LEMD36R-LRCV27

AIRCRAFT REGISTRATION	AIRCRAFT HEXIDENT
LEMD-LRCV:HALWB	LEMD-LRCV:471F49

Figura 6.41: Tablas utilizadas en el cuadro de mandos

(columns) con su nombre y su id y después mediante la propiedad data se deben añadir las filas, convirtiendo el dataframe con el que se está trabajando a un diccionario récords. En la Figura 6.42 se muestra un ejemplo de código de creación de una tabla

```
dash_table.DataTable(
    id='tabla1',
    columns=[{"name": i, "id": i} for i in df.columns],
    data=df.to_dict('records'),
)
```

Figura 6.42: Código creación de una tabla

En nuestro caso como las tablas dependen de la información seleccionada previamente, nuevamente se han creado las tablas con los elementos columnas y data vacíos, para después rellenarlos por medio de una llamada de retorno.

6.2.5. Llamadas de Retorno

Las llamadas de retorno que ofrece Dash son las encargadas de añadir interactividad al cuadro de mandos, se lanzan de forma automática cuando se produce algún evento que se esté controlando. En este trabajo no se ha utilizado Javascript, utilizando en todo momento las llamadas de retorno que ofrece la propia herramienta.

En la Figura 6.43 se muestra el código del evento que se produce cuando se selecciona una fecha en el desplegable, con Input se indica cuando se produce el evento, es decir, indica que el evento se lanzará, en este caso, cuando se modifique la propiedad value del elemento 'dropdown-date'. Con

Output se indica el destino del evento, en este caso la propiedad options del elemento 'dropdown-callsign'. Básicamente lo que estamos haciendo es que cuando se seleccione una fecha en el desplegable, se carguen los callsign correspondientes a esa fecha. Después se define la función que recibe el valor seleccionado, y debe calcular los valores de callsign a devolver, toda esta gestión se realiza mediante Python realizando una consulta a la base de datos de la tabla correspondiente, sobre la que queremos filtrar los resultados. Finalmente, devolvemos options tal y como se espera, es decir, una lista de diccionarios claves valor.

```
#Carga los callsign en función de la fecha seleccionada
@app.callback(
    Output('dropdown-callsign', 'options'),
    [Input('dropdown-date', 'value')])
def set_callsign_options(selected_date):
    if(selected_date is not None):
            legs = pd.read_sql_query('''SELECT DISTINCT(leg_callsign) as callsign FROM public."LEG"
            where date = %(selected_date)s;''',cnx,params={"selected_date":selected_date})
            return [{'label': c, 'value': c}
                    for c in callsigns['leg_callsign']]
    else:
        return []
```

Figura 6.43: Código utilizado para llamada de retorno

Una de las limitaciones encontradas al hacer uso de la librería de Dash para Jupyter, ha sido que no se permite la posibilidad de definir varias salidas, en una misma función de retorno, siendo necesario implementar una por cada salida. En las últimas versiones de Dash, esta opción ya está soportada y hace que el código se reduzca notablemente, sobre todo en un caso como el nuestro, en el que varios componentes dependen de la selección de un mismo elemento.

6.2.6. Estilos

Con los estilos es donde nos hemos encontrado otra de las limitaciones más importantes. Dash permite añadir hojas de estilos CSS externas de forma muy sencilla. La librería Jupyter Dash no permite incluir hojas externas por lo que para no definir los estilos dentro del propio código lo que se ha hecho es destinar una celda del cuaderno Jupyter para crear diccionarios con los estilos a utilizar, y luego asignar esas variables diccionario a la etiqueta style de los componentes.

La Figura 6.44 muestra un ejemplo de la definición de un estilo de cabecera de caja, que es el utilizado para las cajas en las que se alojan, los mapas y los gráficos.

```
#Celda que recoge todos los estilos utilizados en el cuadro de mando
box_title = {
    'background-color': '#0176B0',
    'border-bottom': '1px solid #f4f4f4',
    'padding': '20px',
    'color': fondo_grafico,
    'font-weight': 'bold',
    'font-size': '20px',
}
```

Figura 6.44: Ejemplo de Estilos Cabecera de Caja

Visualización del dashboard

En este capítulo se mostrará un ejemplo de uso del cuadro de mandos final obtenido, realizando un recorrido por los distintas opciones que ofrece. El cuadro de mandos se encuentra organizado en tres pestañas, que permitirán explotar la información aérea recibida a través del proyecto AIRPORTS, para finalmente poder analizarla en busca de aspectos de mejora.

7.1. Trayectorias

En la Figura 7.45 se muestra la pantalla de inicio del dashboard. Como se puede observar aparece seleccionada la pestaña de Trayectorias, que es en la que se podrá obtener información de los vuelos en función de los parámetros de filtrado, que se introduzcan en el panel lateral izquierdo.

Figura 7.45: Pantalla de inicio del dashboard

Lo primero que se debe hacer es seleccionar una fecha, ya que como se ha explicado anteriormente para agilizar el funcionamiento del cuadro de mandos, se han agrupado los mensajes por fechas. Una vez selecciona una fecha, en el caso de la Figura 7.46, la del 7 de febrero de 2018, se actualizará el desplegable que permite seleccionar un callsign con los identificadores de

las aeronaves para los que dispongamos de mensajes. Tras esto, se deberá seleccionar un callsign de entre los existentes, lo que producirá que automáticamente se muestren los legs (tramos) correspondientes a ese callsign con un check por cada tramo, permitiendo mostrar o no cada tramo de forma independiente y a elección del usuario. Cada tramo se identifica mediante el aeropuerto de origen, seguido del aeropuerto de destino.

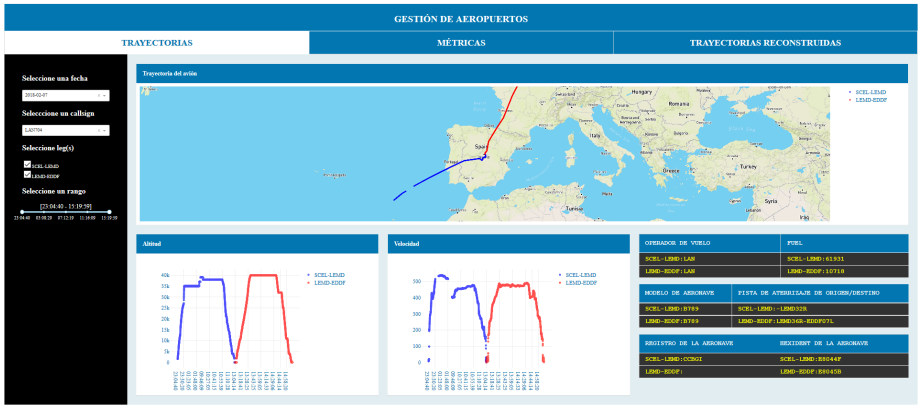

Figura 7.46: Pantalla de trayectorias

En el ejemplo de uso que se muestra en la Figura 7.46 se han seleccionado los dos tramos correspondientes al callsign LAN704. Una vez seleccionado cualquiera de los tramos o ambos como es el caso, se actualizará el dashboard mostrando sobre el mapa los mensajes, mediante puntos, identificando cada tramo por medio de un color tal y como se puede observar en la Figura 7.47.

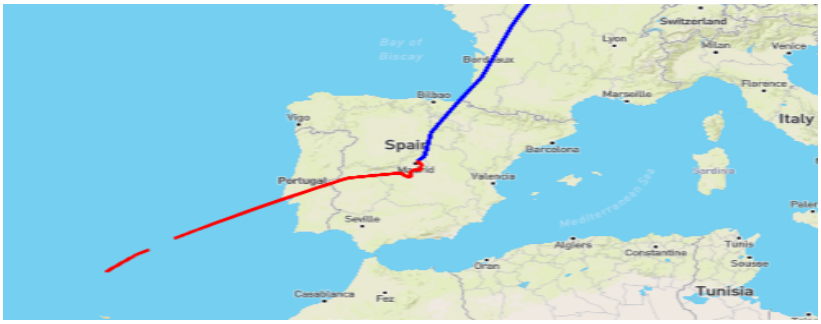

Figura 7.47: Tramos representados sobre el mapa

Al posicionar el ratón sobre cada punto se podrá consultar la hora, la longitud, la latitud y la altitud de un mensaje concreto, tal y como se

muestra en la Figura 7.48. También se podrá descargar el gráfico como imagen, mediante las herramientas que proporciona Plotly tal y como se puede ver en la Figura 7.49 o rotar la posición del mapa pulsando la tecla Control y arrastrando con el ratón sobre el mismo, tal y como se muestra en la Figura 7.50

Figura 7.48: Información de un mensaje

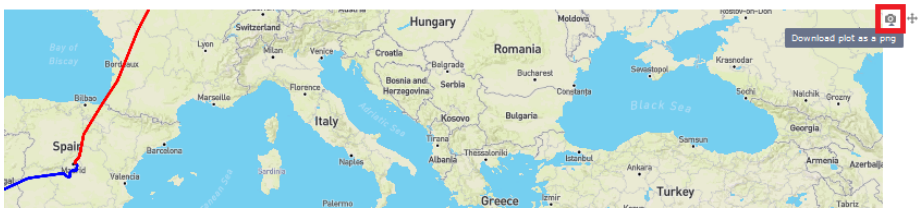

Figura 7.49: Descarga de gráficos como imágen

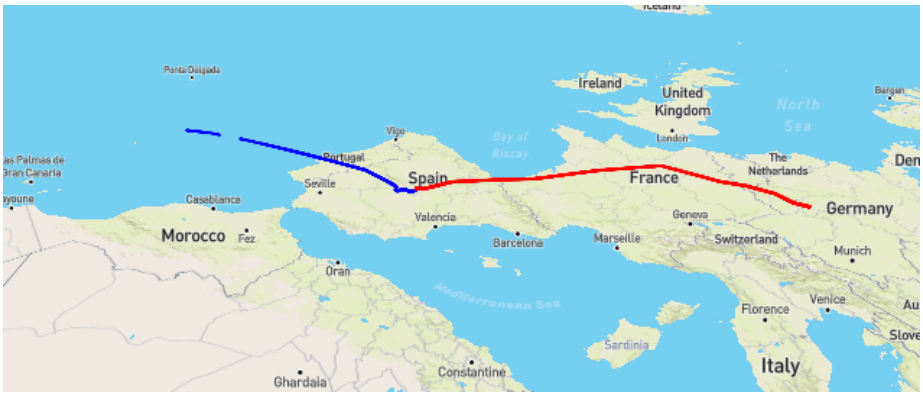

Figura 7.50: Rotación del mapa

Además el dashboard proporciona dos gráficas que representan la velocidad y la altitud en el tiempo, siendo esta información de gran utilidad a la hora de buscar puntos de mejora en los sistemas ATM. En la figura Figura 7.51 se muestran las dos gráficas para el caso de uso en el que nos encontramos, como podemos observar la altura y la velocidad están íntimamente relacionadas.

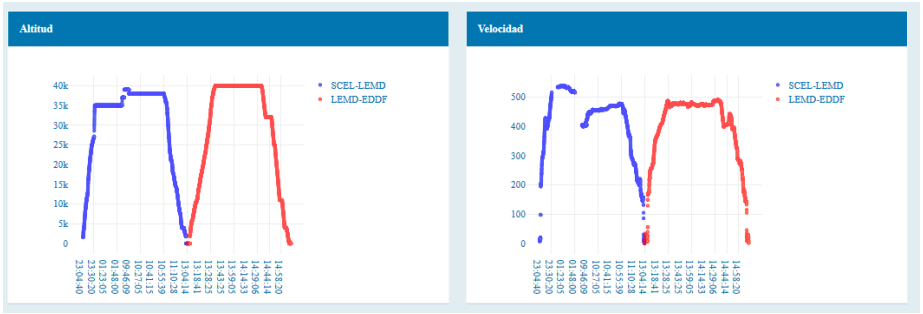

Figura 7.51: Altitud y Velocidad

A la derecha de estos gráficos se muestra simulando el panel de un aeropuerto, información de relevancia de cada tramo, como es el operador, el fuel consumido, el modelo de aeronave, la pista de aterrizaje de origen y de destino, el código de registro y el hexident de la aeronave. En la figura Figura 7.52 se muestra un ejemplo de este panel, con la información de los dos tramos que han sido seleccionados, en caso de que algún dato no haya sido registrado, simplemente no será pintado permaneciendo su hueco vacío, como es el caso de la pista de aterrizaje de origen para el segundo tramo del ejemplo.

OPERADOR DE VUELO	FUEL
SCEL-LEMD:LAN	SCEL-LEMD:61931
LEMD-EDDF:LAN	LEMD-EDDF:10718

MODELO DE AERONAVE	PISTA DE ATERRIZAJE DE ORIGEN/DESTINO
SCEL-LEMD:B789	SCEL-LEMD:-LEMD32R
LEMD-EDDF:B789	LEMD-EDDF:LEMD36R-EDDF07L

REGISTRO DE LA AERONAVE	HEXIDENT DE LA AERONAVE
SCEL-LEMD:CCBGI	SCEL-LEMD:E8044F
LEMD-EDDF:	LEMD-EDDF:E8045B

Figura 7.52: Panel con información de los tramos

Finalmente, cabe destacar que al seleccionar cualquiera de los tramos o más de uno, en el menú lateral se muestra la barra deslizante de la Figura 7.53, que permite acotar los tramos, permitiendo al usuario poner el foco en unas horas concretas del día. En la Figura 7.54 y la Figura 7.55 se puede ver como se han actualizado los datos ajustándose al rango establecido en el selector.

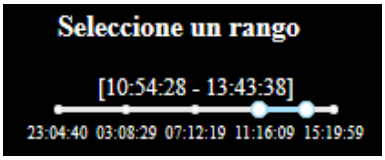

Figura 7.53: Filtro por rango horario

Figura 7.54: Trayectoria de acuerdo al rango seleccionado

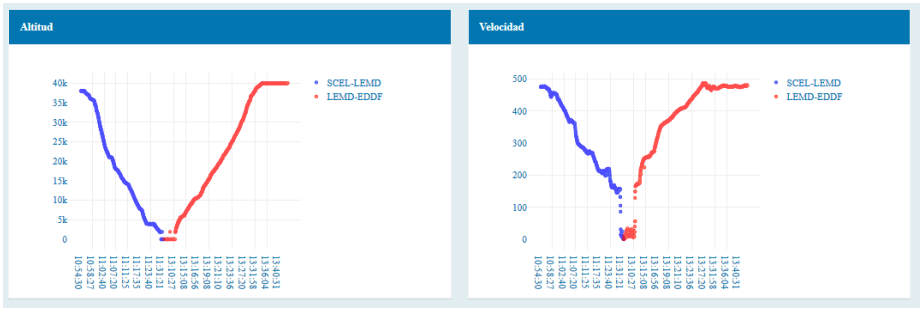

Figura 7.55: Altitud y velocidad de acuerdo al rango seleccionado

7.2. Métricas

Si pulsamos sobre la pestaña de métricas se muestra una nueva ventana como la de la Figura 7.56, en la que de nuevo aparece un menú lateral, pero que en esta ocasión permite filtrar por los tres KPI´s tiempo, distancia y consumo de combustible, para un tipo de trayectoria sintética de entre las tres existentes: geodésica respecto a la real, plan de vuelo y geodésica respecto al plan de vuelo.

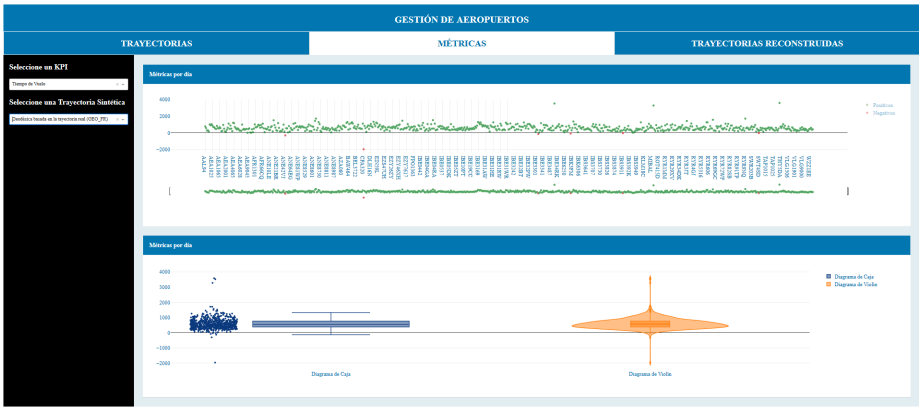

Figura 7.56: Pantalla de métricas

Una vez seleccionado un KPI y una trayectoria sintética se muestra una primera gráfica, con los resultados obtenidos para todos los callsign, apareciendo en verde los valores positivos y en rojo los negativos, tal y como se puede ver en la figura Figura 7.57 .

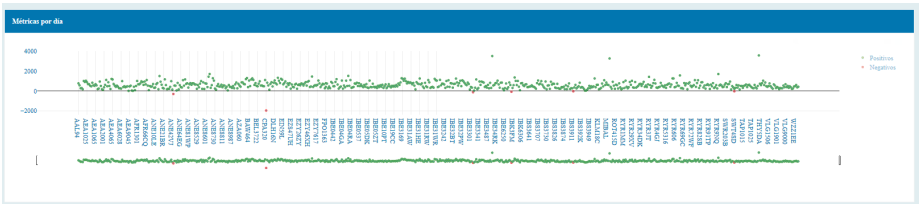

Figura 7.57: Gráfica de métricas

Esta gráfica, tal y como se muestra en la Figura 7.58 permite además acotar los callsign facilitando la detección de comportamientos localizados.

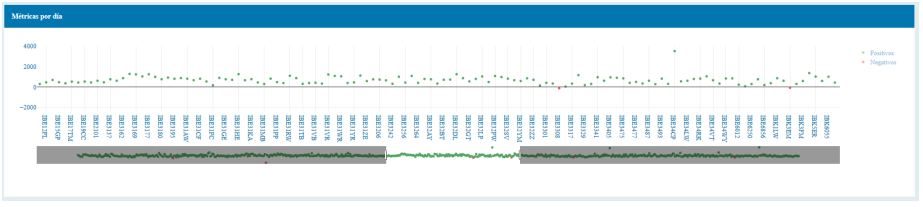

Figura 7.58: Gráfica de métricas acotada

Debajo de esta gráfica y tal y como se puede observar en la Figura 7.59, se muestra un diagrama de caja y un diagrama de violín que permiten ofrecer

al usuario más información sobre la densidad y la dispersión de los datos analizados.

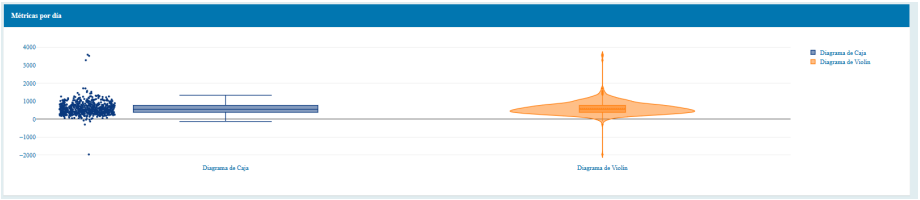

Figura 7.59: Diagrama de caja y de violín para métricas

7.3. Trayectorias reconstruidas

Por último se ha recogido en una nueva pestaña denominada Trayectorias reconstruidas, un gráfico que permite realizar comparaciones entre las trayectorias sintéticas de un vuelo. Nuevamente mediante el panel lateral, se ofrece al usuario la posibilidad de seleccionar el vuelo. Una vez seleccionado el vuelo se muestran unos checks que permiten decidir, de entre las trayectorias sintéticas disponibles para ese vuelo, cuales se desean mostrar. En la Figura 7.60 se puede observar como se ha decidido mostrar las 4 trayectorias existentes para el vuelo ACA837.

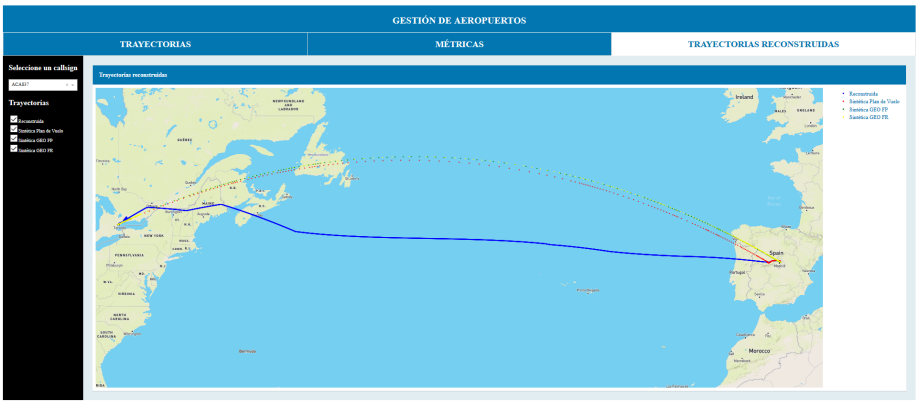

Figura 7.60: Pantalla de trayectorias reconstruidas

Esta pestaña ofrece una herramienta de comparación muy potente, ya que permite ver en detalle las diferencias de trazado entre la trayectoria reconstruida y las trayectorias sintéticas, pudiendo detectar de forma muy

rápida y sencilla deficiencias en los trayectos realizados durante los vuelos. Por ejemplo en la Figura 7.61 se aprecia como las 3 trayectorias sintéticas, trazan una curva muy parecida que representa la trayectoria óptima calculada, y que difiere completamente de la trayectoria reconstruida a partir de los mensajes recibidos de la aeronave, lo cual indica la clara necesidad de una mejora en las rutas que siguen actualmente las aeronaves.

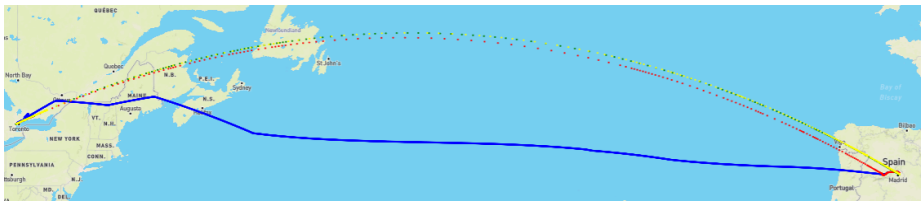

Figura 7.61: Mapa con trayectorias reconstruidas

Conclusiones y Líneas de trabajo futuras

En este último capítulo se explicarán las conclusiones observadas durante el desarrollo del trabajo, así como se propondrán líneas de trabajo futuras.

8.1. Conclusiones

Este Trabajo Fin de Máster ha planteado un desafío importante, a nivel de investigación, sobre un área emergente como es la visualización de grandes volúmenes de datos, en un entorno Big Data, enfocado al sector de Tráfico Aéreo, ámbito que genera un gran impacto social, económico y medioambiental. El Big Data es una herramienta muy potente que permite gestionar grandes volúmenes de datos, pero toda esa gestión no sirve de nada, sin una visualización adecuada que ayude a convertir esos datos en información. El ser humano por su gran capacidad ocular, analiza y comprende mejor la información que recibe de forma visual, por lo que no tiene sentido, conseguir gestionar miles de millones de datos, si después no somos capaces de extraer conclusiones "de un solo vistazo", a partir de toda esa información.

Elegir una herramienta de visualización adecuada a los datos que se necesitan explorar es fundamental. La decisión de buscar una herramienta en Python para realizar la visualización ha sido muy acertada, ya que el uso de su librería Pandas para la gestión de datos, y más concretamente de su estructura de datos estrella el Dataframe, ha facilitado considerablemente la realización de los filtrados de la información. Además con la arquitectura planteada, se ha conseguido el desarrollo de un cuadro de mandos ágil,

rápido y muy fácilmente escalable, logrando de esta forma alcanzar todos los objetivos propuestos.

Al tratarse de un entorno Big Data es imprescindible que la librería de visualización elegida, entre las múltiples ofrecidas por Python, permita interacción. Esta restricción ha condicionado que librerías potentes y muy utilizadas en la actualidad como son Matplotly o Seaborn hayan sido descartadas. La elección entre Bokeh y Dash ha sido más complicada. De hecho se propone como línea de trabajo futura, realizar el mismo cuadro de mandos utilizando Bokeh, para comparar el rendimiento entre ambas, aunque cuando se ha estado investigando, múltiples usuarios indicaban que Dash es una herramienta mucho más rápida a la hora de trabajar con grandes volúmenes de datos.

La experiencia con Dash ha sido muy satisfactoria. Los componentes son muy fáciles de incorporar y muy intuitivos, además al ofrecer la gran potencia de los gráficos de Plotly, resulta muy sencillo añadir al cuadro de mandos, todo tipo de gráficos, incluidos mapas. El único inconveniente ha sido las limitaciones derivadas de su uso dentro de los cuadernos Jupyter, ya que Dash permite el uso de frameworks de maquetación como Bootstrap o Materialize, así como incluir hojas CSS externas e incluso el uso de Javascript, pero en nuestro caso al utilizar la librería de Dash para Jupyter, no ha sido posible disfrutar de todas estas ventajas. Tampoco hemos podido utilizar la salida múltiple en las llamadas de retorno, que Dash ya ha implementado en sus últimas versiones, lo que ha provocado que en los casos en los que se necesita filtrar la misma información para varios componentes de salida, se haya tenido que ejecutar una función por cada componente, teniendo que repetir las operaciones. Viendo el gran potencial que ofrece la librería, y el ritmo al que avanza la tecnología, en un futuro muy cercano todas estas limitaciones desaparecerán, lo que se traducirá en una integración perfecta entre Jupyter y Dash.

8.2. Líneas de trabajo futuras

Dada la importancia de la visualización que ya se ha puesto de manifiesto en reiteradas ocasiones a lo largo de este trabajo, sería conveniente que en un futuro cercano, se le proporcionase al usuario final herramientas fáciles y rápidas que permitan configurar un cuadro de mandos a su medida, sin necesidad de conocimientos técnicos. De esta forma será el propio usuario el encargado de decidir que tipo de gráficos mostrar, y podrá modificarlos fácilmente a su antojo. Existen diversas herramientas de este tipo, destinadas

a que sea el propio usuario el responsable de crear las visualizaciones, pero requieren de una curva de aprendizaje demasiado alta, no accesible para usuarios con conocimientos técnicos básicos.

Otra línea de trabajo futuro que propongo es tratar de integrar la realidad aumentada al cuadro de mandos. La realidad aumentada es un conjunto de técnicas que mediante la utilización de un dispositivo, son capaces de crear un mundo en el que se combinan elementos ficticios sobre un entorno real, permitiendo al usuario interactuar con ellos. En la actualidad podemos encontrar la realidad aumentada en múltiples sectores como el de los videojuegos, la moda o el turismo. En la Figura 8.62 se muestra un cuadro de mandos de la empresa LLOG VR[20] para la gestión de un almacén con realidad aumentada. Lograr dotar a nuestro cuadro de mandos actual de realidad aumentada sería todo un reto y más al tratarse del sector aéreo, ya que es un entorno bastante complejo, en el que conviven múltiples actores como son aviones, aeropuertos, aerolíneas, etc.

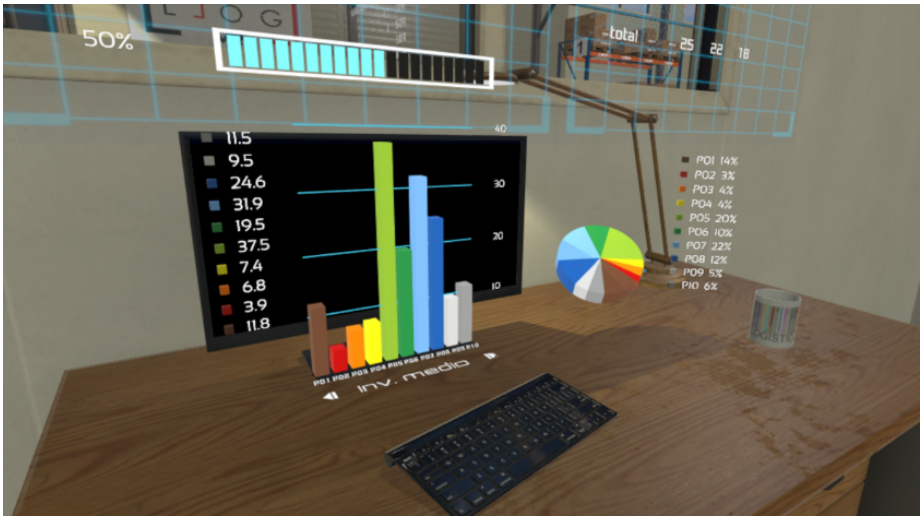

Figura 8.62: Cuadro de mandos con realidad aumentada [20]

Apéndices

Apéndice A

Código Python Almacenamiento de Tramos (Legs)

```python
#Leemos la ruta donde vamos a guardar los ficheros auxiliares del fichero de configuración
#Leemos el nombre del fichero donde están los legs
f = open("..\\config.txt","r")
j=0
it=(linea for i,linea in enumerate(f) if i==1 or i==3 or i==7)
for linea in it:
    if(j==0):
        path_messages=linea
        j+=1
    elif (j==1):
        path_legs=linea
        j+=1
    else:
        fich_legs=linea

path_messages=path_messages.replace('\n','')
path_legs=path_legs.replace('\n','')
fich_legs=fich_legs.replace('\n','')

print(path_messages)
print(path_legs)
print(fich_legs)
```

Figura A.1: Lectura fichero de configuración

```python
import pandas as pd
from sqlalchemy import create_engine

# Postgres username, password, and database name
POSTGRES_ADDRESS = 'localhost' ## INSERT YOUR DB ADDRESS IF IT'S NOT ON PANOPLY
POSTGRES_PORT = '5432'
POSTGRES_USERNAME = 'postgres' ## CHANGE THIS TO YOUR PANOPLY/POSTGRES USERNAME
POSTGRES_PASSWORD = 'admin' ## CHANGE THIS TO YOUR PANOPLY/POSTGRES PASSWORD POSTGRES_DBNAME = 'database'
## CHANGE THIS TO YOUR DATABASE NAME
POSTGRES_DBNAME = 'DATOS_AEREOS'
#A long string that contains the necessary Postgres login information
postgres_str = ('postgresql://{username}:{password}@{ipaddress}:{port}/{dbname}'
               .format(username=POSTGRES_USERNAME,
                password=POSTGRES_PASSWORD,
                ipaddress=POSTGRES_ADDRESS,
                port=POSTGRES_PORT,
                dbname=POSTGRES_DBNAME))
# Create the connection
cnx = create_engine(postgres_str)
```

Figura A.2: Conexión base de datos

```python
#Código para reemplazar NULL por nada

import os

filename = fich_legs
filename_new = path_legs+"\legs_new.csv"
filename_final=path_legs+"\legs_final.csv"
#Si no existe la carpeta indicada por el fichero path la creamos
try:
    os.stat(path_legs)
except:
    os.mkdir(path_legs)

filer = open(filename, "r")
filew = open(filename_new, "w")
buff = filer.read()
rbuff = buff.replace('NULL', '')
sbuff = rbuff.replace('legs.', '')
wbuff = sbuff.replace('"', '')
filew.write(wbuff)

filer.close()
filew.close()


```

Figura A.3: Limpieza de datos

```

#leemos fichero
legs = pd.read_csv(filename_new)

#añadimos la columna path
legs['path']=legs['leg_id']+".csv"
#lo guardamos en un fichero
filename=path_legs+"\legs_final.csv"
legs_sin_espacios = legs.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
legs_sin_espacios.to_csv(filename, index=False, header=False,sep=',',decimal='.')

#Eliminamos el fichero intermedio
os.remove(filename_new)


```

```
#Código para importar los datos del fichero resultante sin NULL y con ruta directamnte a postgres
import psycopg2;
conn=psycopg2.connect("host=localhost dbname=DATOS_AEREOS user=postgres password=admin")
cur=conn.cursor()
with open(filename,'r') as f:
    cur.copy_from(f,'''public."LEG"''',sep=',')
conn.commit()

#Eliminamos el fichero final
os.remove(filename_final)
#Eliminamos la carpeta legs
os.removedirs(path_legs)

#next(f) por si hay que saltar la cabecera en la inserción
```

Figura A.4: Inserción de los tramos en la base de datos

Apéndice B

Código Python Segmentación de Mensajes

```python
#Obtenemos los leg_id distintos y por cada uno creamos un fichero con los registros de ese leg_id
#ordenador por la hora
#los guardamos en una carpeta del fichero de configuración y dentro en una carpeta con la fecha de la recogida de
#los datos y con el nombre messages_leg_id

import os

#Obtenemos los distintos leg_id
leg_ids=messages.leg_id.unique()
messages_leg_id= pd.DataFrame()

#Si no existe la carpeta indicada por el fichero path la creamos
try:
    os.stat(path)
except:
    os.mkdir(path)

#Recorremos los leg_id y vamos particionando las filas de ese leg_id en ficheros
for leg_id in leg_ids:
    messages_leg_id=messages[messages['leg_id']==leg_id]
    messages_leg_id_ordenado=messages_leg_id.sort_values(by='timestamp', ascending=True)
    fecha=messages_leg_id_ordenado.fecha.unique()
    dir_aux=path+str(fecha)
    dir_aux=dir_aux.replace('[','')
    dir_aux=dir_aux.replace(']','')
    dir=dir_aux.replace("'",'')
    #Si el directorio no existe lo creamos con el nombre de la fecha
    try:
      os.stat(dir)
    except:
      os.mkdir(dir)
    #Construimos el nombre del fichero con el valor de su leg_id y lo guardamos
    filename=dir+"\\"+leg_id+".csv"
    messages_leg_id_ordenado.to_csv(filename, index=False, header=False,sep=',',decimal='.',encoding='utf-8')

```

Figura B.1: Segmentación y almacenamiento de Mensajes

Apéndice C

Código Python Almacenamiento de Métricas

```
#Código para reemplazar NULL por nada
import psycopg2;
import os

filename =fich_metricas
filename_new = path_metricas+"\metricas_new.csv"
filename_final=path_metricas+"\metricas_final.csv"
#Si no existe la carpeta indicada por el fichero path la creamos
try:
    os.stat(path_metricas)
except:
    os.mkdir(path_metricas)

filer = open(filename, "r")
filew = open(filename_new, "w")
buff = filer.read()
rbuff = buff.replace('NA', '')
wbuff = rbuff.replace('"', '')
filew.write(wbuff)

filer.close()
filew.close()



#Código para importar los datos del fichero resultante sin NULL y con ruta directamnte a postgres

conn=psycopg2.connect("host=localhost dbname=DATOS_AEREOS user=postgres password=admin")
cur=conn.cursor()
with open(filename_new,'r') as f:
    cur.copy_from(f,'''public."METRIC"''',sep=',')
conn.commit()


```

Figura C.1: Almacenamiento de Métricas en base de datos

Apéndice D

Código Python Segmentación y Almacenamiento de Trayectorias Sintéticas

```python
def particionar(fichero):

    datos_fichero = pd.read_csv(fichero)

    #Recorremos los leg_id y vamos particionando las filas de ese leg_id en ficheros
    legs=datos_fichero.leg.unique()

    salida= pd.DataFrame()

    for leg in legs:
        salida=datos_fichero[datos_fichero['leg']==leg]
        salida_ordenada=salida.sort_values(by='timestamp', ascending=True)
        fecha=salida_ordenada.date.unique()
        dir_aux=str(fecha)
        dir_aux=dir_aux.replace('[','')
        dir_aux=dir_aux.replace(']','')
        dir_fecha=dir_aux.replace("'",'')
        fecha_str=str(dir_fecha)
        dir=path+"//"+fichero.split('/')[1].split('.')[0]
        subdir=dir+"//"+dir_fecha
        #Si el directorio no existe lo creamos con el nombre de la fecha
        try:
          os.stat(dir)
        except:
          os.mkdir(dir)
        #Creamos también el subdirectorio correspondiente al tipo de métrica
        try:
          os.stat(subdir)
        except:
          os.mkdir(subdir)
```

Figura D.1: Limpieza de datos

```
        #Construimos el nombre del fichero con el valor de su leg_id y lo guardamos
        filename=subdir+"//"+leg+".csv"
        salida_ordenada.to_csv(filename, index=False, header=False,sep=',',decimal='.',encoding='utf-8')

        leg_id = pd.read_sql_query('''SELECT leg_id
        FROM public."SYNTHETIC_TRAJECTORY"
        where leg_id = %(leg)s and date=%(date)s;''',cnx,
        params={"leg":str(leg),"date":str(fecha)})

        if len(leg_id)<=0:
            #Insertamos
            print('Insertamos')
            print(str(leg))
            sql = """
                INSERT INTO public."SYNTHETIC_TRAJECTORY"
                VALUES('"""+ str(leg)+"""','"""+fecha_str+"""',"""

            if fichero.split('/')[1]=='reconstructed_trajectories.csv':
                sql_aux = """True,False,False,False)"""


            elif fichero.split('/')[1]=='syn_trj_geo_fp.csv':
                sql_aux = """False,True,False,False)"""
            elif fichero.split('/')[1]=='syn_trj_geo_fr.csv':
                sql_aux = """False,False,True,False)"""
            else:
                sql_aux = """False,False,False,True)"""
            with cnx.begin() as conn:     # TRANSACTION
                    conn.execute(sql+sql_aux)
```

Figura D.2: Inserción indicador pertenencia a fichero

```
        else:
            #Actualizamos
            print('Actualizamos')
            print(str(leg))
            sql_inicio="""
                UPDATE public."SYNTHETIC_TRAJECTORY" """
            sql_fin="""WHERE leg_id = '"""+ str(leg)+"""' and date='"""+fecha_str+"""'"""

            if fichero.split('/')[1]=='reconstructed_trajectories.csv':
                sql = """
                SET reconstructed_trajectories=True
                """

            elif fichero.split('/')[1]=='syn_trj_geo_fp.csv':
                sql = """
                SET syn_trj_geo_fp=True
                """

            elif fichero.split('/')[1]=='syn_trj_geo_fr.csv':
                sql = """
                SET syn_trj_geo_fr=True
                """
            else:
                sql = """
                SET syn_trj_up=True
                """

            with cnx.begin() as conn:     # TRANSACTION
                    conn.execute(sql_inicio+sql+sql_fin)
```

Figura D.3: Actualización indicadores pertenencia a fichero

```
1
2
3
4
5
6  import os
7
8  #Si no existe la carpeta indicada por el fichero path la creamos
9  try:
10     os.stat(path)
11 except:
12     os.mkdir(path)
13
14 particionar(config_path_reconstructed_trajectories)
15 particionar(config_path_syn_trj_geo_fp)
16 particionar(config_path_syn_trj_geo_fr)
17 particionar(config_path_syn_trj_up)
18
19
20
21
22
```

Figura D.4: Llamada con los cuatro ficheros de trayectorias

Bibliografía

[1] Databricks. Glosary data lake. https://databricks.com/glossary/data-lake [Agosto-2019].

[2] Javier Daza. El zen de python, 2019. https://gist.github.com/javierdaza/4258b74e2eb7cfd4f55286061b592f37 [Junio-2019].

[3] La Voz de Galicia. Trayectoria geodésica. https://www.lavozdegalicia.es/default/2018/01/12/00161515788809189357576/Foto/HE13P51G1-01.jpg [Agosto-2019].

[4] Calvo et al. "A New Method to Validate the Route Extension Metric against Fuel Efficiency", Eleventh USA/Europe Air Traffic Management Research and Development Seminar (ATM2015).

[5] EUROCONTROL. Previsión semestral, 2018. https://www.eurocontrol.int/sites/default/files/content/documents/official-documents/forecasts/seven-year-flights-service-units-forecast-2018-2024-Feb2018.pdf [Junio-2019].

[6] Universidad Europea. Cuánto gana un ingeniero informático. https://universidadeuropea.es/blog/cuanto-gana-un-ingeniero-informatico [Septiembre-2019].

[7] Python Software Foundation. Librería jupyter dash, 2019. https://pypi.org/project/jupyter-plotly-dash/ [Junio-2019].

[8] Gartner. Magic quadrant for analytics and business intelligence platforms, 2019. https://www.qlik.com/es-es/gartner-magic-quadrant-business-intelligence [Junio-2019].

[9] I. García, M. A. Martínez-Prieto, A. Bregón, P. C. Álvarez, and F. Díaz. "Towards a Scalable Architecture for Flight Data Management", in 6th International Conference on Data Science, Technology and Applications (DATA), pages 263-268, 2017.

[10] International Air Transport Association (IATA). Seguridad de aviación, 2019. `https://www.iata.org/travelers/Pages/aviation-safety.aspx` [Junio-2019].

[11] Iván García-Miranda, Laura Hernán-Muñoz, Fernando Díaz, Anibal Bregón, Miguel A. Martínez-Prieto, Pedro C. Álvarez-Esteban and Javier López-Leones. "AIRPORTS Metrics: A Big Data application for computing flights performance indexes based on flown trajectories", in Digital Avionics Systems Conference (DASC), 2018 IEEE/AIAA 37th, pp. 1–10, IEEE, 2018.

[12] Kevin M. Run dash app in jupyter, 2017. `https://community.plot.ly/t/can-i-run-dash-app-in-jupyter/5235/2` [Junio-2019].

[13] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm. Bringing Up. "OpenSky: A Large-scale ADS-B Sensor Network for Research", in 13th International Symposium on Information Processing in Sensor Networks (ISPN), pages 83-94, 2014.

[14] M. Strohmeier, I. Martinovic, M. Fuchs, M. Schäfer, y V. Lenders. "Opensky: A swiss army knife for air traffic security research", 34th Digital Avionics Systems Conference (DASC), pp. 4a1-1-4a1-14, 2015.

[15] Jon Mease. Librería jupyter lab, 2019. `https://github.com/plotly/jupyterlab-dash` [Junio-2019].

[16] Miguel A. Martínez-Prieto, Anibal Bregon, Iván García-Miranda, Pedro C. Álvarez-Esteban, Fernando Díaz and David Scarlatti. "Integrating Flight-related Information into a (Big) Data Lake", in Digital Avionics Systems Conference (DASC), 2017 IEEE/AIAA 36th,pp. 1–10, IEEE, 2017.

[17] Iván García Miranda. "Modeling and Integrating Flight-related Data for the Improvement of Airport Operations", Trabajo Fin de Máster, Máster Universitario de Investigación en TIC (MUITIC), Universidad de Valladolid, 2018.

[18] OACI. Informe preliminar, 2018. `https://www.icao.int/Newsroom/Pages/ES/Solid-passenger-traffic-growth-and-moderate-air-cargo-demand-in-2018.aspx` [Junio-2019].

[19] IEEE Spectrum. Top lenguajes de programación 2018, 2018. `https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages` [Junio-2019].

[20] Prof. Dr. Alex Rodríguez Villalobos. Llog vr – cuadro de mando logístico en realidad aumentada, 2019. `https://arodriguez.blogs.upv.es/llog-vr-cuadro-de-mando-logistico-en-realidad-aumentada/` [Agosto-2019].

[21] C. Madera y A. Laurent. "The next information architecture evolution: the data lake wave", Proceedings of the 8th International Conference on Management of Digital EcoSystems, pp. 174-180, ACM, 2016.

[22] N. Miloslavskaya y A. Tolstoy. "Application of big data, fast data, and data lake concepts to information security issues", Future Internet of Things and Cloud Workshops (FiCloudW), IEEE International Conference on, pp. 148-153, IEEE, 2016.

[23] Robert Kaplan y David Norton. Definición de cuadro de mandos, 1992. `https://hbr.org/1992/01/the-balanced-scorecard-measures-that-drive-performance-2` [Agosto-2019].

[24] Álvaro Alonso-Isla, Pedro C. Álvarez-Esteban, Aníbal Bregón, Luís D'Alto, Fernando Díaz, Iván García, Paula Gordaliza, Javier López-Leonés, Miguel A. Martínez-Prieto, David Scarlatti, Miguel Vilaplana. "Airports: Análisis de Eficiencia Operacional basado en Trayectorias de Vuelo", en XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD), 2018.