Sección de Matemáticas
Universidad de La Laguna

Tania Melián Expósito

Inteligencia de negocio aplicada a una empresa de transporte

Business Intelligence applied to a transport company

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Junio de 2023

Dirigido por
Gara Miranda Valladares
Pedro Antonio Toledo Delgado

Gara Miranda Valladares
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna
38200 La Laguna, Tenerife

Pedro Antonio Toledo Delgado
Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

A mis tutores, Gara y Pedro, por la paciencia, dedicación y las ganas de enseñarme.

A mis padres y mi hermana, a quienes debo todo lo que soy y lo que he conseguido. Por ser mi mayor apoyo y sacar lo mejor de mí en todo momento.

A mis amigos, a los de siempre y a los que hice en este camino. No habría sido lo mismo sin ustedes a mi lado.

A Paula, por estar siempre y por celebrar mis logros como suyos. Sin duda elegí muy bien a los tres años.

Y a ti, Sergio, por creer en mí y por ser calma cuando más lo necesito.

Tania Melián Expósito
La Laguna, 22 de mayo de 2023

Resumen · Abstract

Resumen

La importancia de los datos en la toma de decisiones empresariales se ha vuelto cada vez más evidente en los últimos años. Las empresas de transporte no son la excepción, ya que la capacidad de analizar grandes volúmenes de datos generados a diario por sus operaciones y de poder extraer de ellos información valiosa es esencial para mejorar la eficiencia y la rentabilidad del negocio.

Con el fin de profundizar en la importancia de este tipo de análisis, en este proyecto se aborda el proceso completo de análisis de datos. Primero, de una forma teórica; luego, aplicando los conocimientos adquiridos en un conjunto de datos reales proporcionados por una empresa de transporte de Tenerife. El enfoque se centra en la gestión y optimización de recursos, ayudando a entender a la empresa cómo está funcionando la flota, cuáles son los costes de mantenimiento y operación, y cómo pueden hacer mejoras de tal forma que se maximice la eficiencia y se minimicen los costos. El proyecto abarca desde la identificación de los datos necesarios, su recopilación y carga a bases de datos hasta su procesamiento y validación haciendo uso de diferentes herramientas informáticas. Finalmente, se han creado cuadros de mando que permitirán a la empresa acceder a una visión global del estado de su flota y el rendimiento de la misma.

Cabe destacar que en este proyecto aparecieron desafíos que debieron superarse. Uno de ellos fue la existencia de datos erróneos, incoherentes e incompletos, por lo que se realizó un proceso de depuración y validación. Además, se encontró un déficit de datos disponibles para realizar un análisis más profundo, lo cual limitó en cierta medida los resultados obtenidos. Por eso, se proponen líneas de trabajos futuros como realizar análisis predictivos para anticipar posibles problemas o necesidades de la flota. Por otro lado, se sugiere la exploración de nuevos campos con el fin de enriquecer el análisis para seguir impulsando el crecimiento y la optimización del negocio.

Palabras clave: *Ciencia de datos – Análisis de negocio – Cuadro de mandos – Python – SQL*

Abstract

The importance of data in business decision making has become increasingly apparent in recent years. Transport companies are no exception since the ability to analyze large volumes of data generated daily by their operations to extract valuable information is essential to improve business efficiency and profitability.

In order to deepen the importance of this type of analysis, this project addresses the complete process of data analysis. First, in a theoretical way and then, apply the knowledge acquired in a set of real data provided by a transport company in Tenerife. The approach is focused on the management and optimization of resources, helping the company to understand how the fleet is performing, what are the maintenance and operation costs, and how they can make improvements in such a way that efficiency is maximized and costs are minimized. The project ranges from the identification of the necessary data, its collection and upload to databases to its processing and validation using different computer tools. Finally, dashboards have been created that will allow the company to access a global vision of the state of its fleet and its performance.

It should be noted that in this project there were challenges that had to be overcome. One of them was the existence of erroneous, inconsistent and incomplete data, for which a purification and validation process was carried out. In addition, there was a scarcity of data available to carry out a deeper analysis, which somewhat limited the results obtained. For this reason, future lines of work are proposed, such as carrying out predictive analysis to anticipate possible problems or needs of the fleet. On the other hand, the exploration of new fields is suggested in order to enrich the analysis to continue promoting the growth and optimization of the business.

Keywords: *Data Science – Business Analysis – Dashboard – Python – SQL*

Contenido

1

Introducción

La revolución digital ha tenido un impacto significativo en nuestras vidas, sobre todo en el ámbito empresarial. Antes de la aparición de las herramientas tecnológicas actuales, dada la complejidad de recopilación de los datos, las empresas únicamente registraban la información que consideraban más importante, como los datos financieros. De esta forma, perdían muchísima información que hoy en día se puede obtener y almacenar con facilidad. En la era digital, la cantidad de datos que se genera a diario es abrumadora y esto expone a las empresas a un desafío cada vez mayor: cómo convertir estos datos en conocimiento valioso que les permita tomar decisiones acertadas y mejorar así su rendimiento.

Es por esto que, actualmente, la mayoría de organizaciones cuenta con plataformas electrónicas donde almacenan esa gran cantidad de datos generada por sus operaciones de tal forma que se puedan recopilar, manipular y analizar con mayor precisión y facilidad utilizando herramientas estadísticas y computacionales, lo que representa una gran ventaja en comparación con los datos no digitalizados. Esta gran oleada de información ha dado lugar a una nueva disciplina, el análisis de datos, que recibe el nombre de inteligencia de negocio o *Business Intelligence* (BI). Estos análisis se enfocan en examinar y extraer información valiosa a partir de grandes conjuntos de datos con el fin de respaldar e impulsar a las empresas en la toma de decisiones, basándose en datos concretos y análisis rigurosos en lugar de intuiciones o suposiciones. A través del BI, las empresas pueden responder a los problemas de negocio como la entrada a nuevos mercados, la mejora de la eficiencia operativa, la reducción de costes, analizar los perfiles de clientes, la rentabilidad de sus productos, etc. Por estas razones, el BI se ha ido convirtiendo con el paso de los años en una herramienta esencial en los procesos empresariales en todos los sectores.

En este proyecto, se profundizará en este tipo de análisis enfocado en un caso práctico en el sector del transporte. Se llevará a cabo el proceso completo de BI, no solo de manera teórica, sino también llevando a cabo su aplicación en datos reales proporcionados por una empresa dedicada al transporte discrecional de viajeros por carreteras en la isla de Tenerife. Ésta cuenta con más de 120

vehículos de diferentes capacidades y clases con los que se realizan diferentes tipos de transportes: transportes turísticos, traslados VIP, transportes de grupos, excursiones, etc. En concreto, el BI es de mucha utilidad para este tipo de empresas. Por ejemplo, puede ayudarlas a determinar las rutas de transporte más efectivas, identificar los horarios más concurridos y evaluar la demanda de los clientes. También pueden utilizar el BI para mejorar la seguridad y la satisfacción del cliente, analizando los datos de los viajes anteriores para determinar los problemas más comunes y tomar medidas para abordarlos. Además, el BI también puede ser utilizado para optimizar los recursos, como la gestión de flotas de vehículos y la planificación de mantenimiento. Al analizar los datos sobre la eficiencia del combustible, el rendimiento del vehículo y la frecuencia de mantenimiento, las empresas de transporte pueden tomar decisiones informadas sobre cuándo retirar los vehículos de la flota y cuándo es necesario realizar mantenimiento preventivo.

Por lo tanto, este Trabajo de Fin de Grado se centrará en la gestión y optimización de recursos, ayudando a entender a la empresa cómo está funcionando la flota, cuáles son los costos de mantenimiento y operación para que les sirva de respaldo a la hora de tomar decisiones estratégicas.

1.1. Objetivos

El objetivo general de este trabajo es profundizar en el proceso completo de análisis de datos en una empresa del sector transporte. Se atravesarán todas las fases del análisis, tanto de forma teórica como práctica. Se verán reflejadas todas las tareas que lo componen, desde la extracción, el preprocesamiento y la validación de los datos con diferentes herramientas informáticas hasta la aplicación de diferentes técnicas de visualización.

Más concretamente, los objetivos serán contextualizar y conocer los activos que maneja la empresa para entender los datos con los que se va a trabajar así como explorar las herramientas informáticas en las que se alojan y utilizar mecanismos adecuados para acceder a ellos y prepararlos. En esta etapa se organizarán y seleccionarán todos los datos con los que se trabajará durante el análisis, detectando posibles errores y descartando aquella información que no se vaya a utilizar. Además, se plantearán mecanismos que permitan automatizar estos procesos de preparación de datos. Una vez los datos estén preparados se pasará a la etapa de procesamiento y análisis. Para ello se utilizarán herramientas en el ámbito del análisis de datos que serán descritas en capítulos posteriores. Tras el análisis, vendrá una etapa de interpretación en la que se crearán diferentes representaciones y cuadros de mando en las que se vean claramente reflejados los resultados obtenidos durante el proceso de análisis de los datos. Así, proporcionamos a la empresa una solución informática que se adapte automáticamente

a la actualización de los datos y de tal forma que pueda interactuar con ellos y filtrar únicamente los que sean necesarios en cada momento.

Por último, comunicaremos los resultados y conclusiones obtenidos a la empresa. A través de informes detallados y presentaciones claras, comentaremos los hallazgos relevantes, las tendencias identificadas y las recomendaciones para la realización de futuros análisis que garanticen la mejora continua de la gestión de la flota.

2

Tecnologías y herramientas utilizadas

En este capítulo se hablará de las herramientas informáticas elegidas entre las múltiples alternativas que existen para llevar a cabo el proceso de análisis. Se explicarán de forma detallada los pasos a seguir para configurar el entorno de trabajo y el porqué de las elecciones realizadas en cada caso. Además, se hablará del lenguaje de programación utilizado y de algunas librerías necesarias.

2.1. Entorno de trabajo

Como hemos mencionado, trabajaremos con datos reales que una empresa de transporte proporciona a la Universidad de La Laguna. Dada la confidencialidad de los mismos, para garantizar su privacidad durante el desarrollo del proyecto, utilizaremos una máquina virtual del *IaaS* de la Universidad de La Laguna [2]. Este acrónimo viene de *Infrastructure as a Service*, que es una de las principales categorías de cómputo en la nube, es decir, una tecnología que permite acceder de manera remota a software, almacenamiento de archivos y procesamiento de datos a través de Internet, sin la necesidad de conectarse a un servidor local. Nuestros datos se almacenarán en el servidor `sygda.iaas.ull.es`.

Realizaremos la conexión a la máquina a través de un túnel *Secure Shell* (SSH), una técnica de administración que nos permite crear una conexión a servidores remotos. Gracias a él, podremos acceder a cualquier cuenta en el equipo a través de un mecanismo de autenticación y, además, nos garantiza confidencialidad y la integridad de los datos enviados y recibidos, pues el tráfico entre las máquinas está cifrado.

Para crear la conexión vamos a usar PuTTy [3], que es un emulador de terminal gratuito y de código abierto que nos permitirá hacer uso del SSH en Windows. En el enlace [1], se muestran los pasos a seguir para descargarlo. Luego, debemos crear la conexión al equipo. Para ello, debemos rellenar *Host Name* y *Port* con los datos que se muestran en la Figura 2.1. Por otro lado, para crear

[1] https://www.linuxpro.org/como-instalar-putty-en-windows

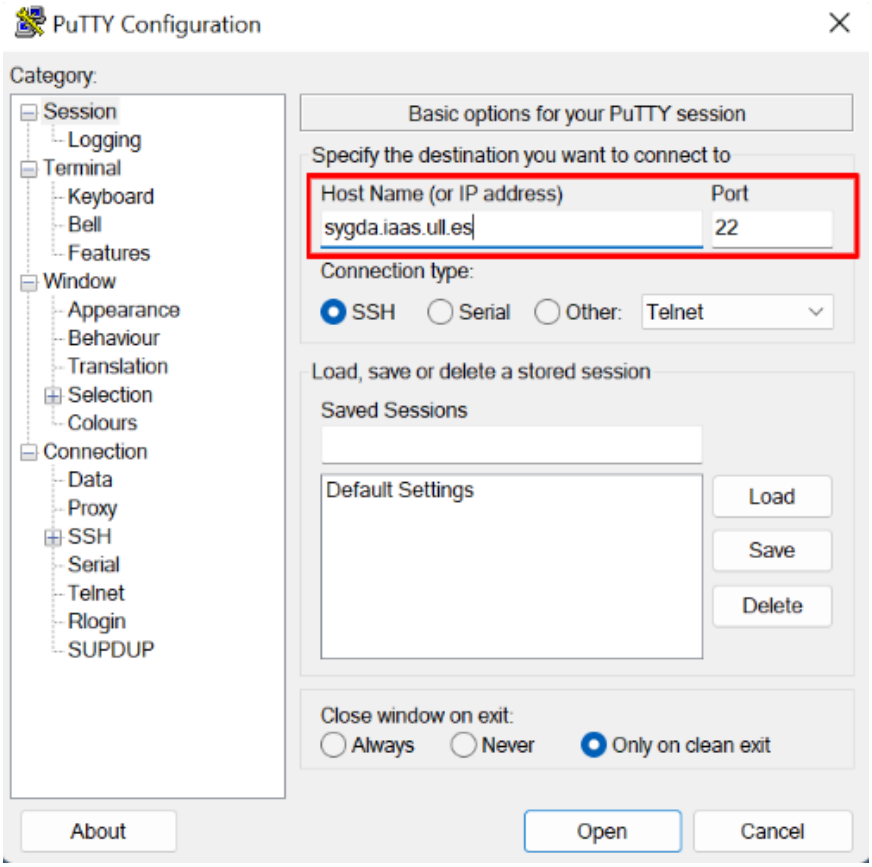


Figura 2.1. Inicio de sesión de PuTTY

el túnel, en la parte izquierda debemos hacer clic sobre la categoría *Connection*, luego sobre SSH y finalmente sobre *Tunnels*. Rellenamos *Source Port* y *Destination* como se ve en la Figura 2.2. Si hacemos clic sobre *Add* y luego sobre *Open*, se abrirá el terminal pidiendo nuestro usuario y contraseña. Una vez lo hayamos introducido todo correctamente, estaremos conectados al terminal.

El procesamiento y análisis de los datos se realizará en Jupyter Lab [4], un entorno interactivo al que podremos acceder a través de una página web que nos permitirá trabajar con distintos tipos de documentos como editores de textos, terminales y Jupyter Notebooks. Este último será el que usaremos en este proyecto.

Jupyter Notebooks una herramienta de código abierto, es decir, que su código fuente está a disposición de todos de manera gratuita, en la que se puede trabajar con múltiples lenguajes de programación. Para descargarlo, debemos seguir los pasos que se indican en el enlace [2]. Para ponerlo en marcha, a través de Putty, debemos situarnos en el directorio donde se encuentren nuestros datos y

[2] https://jarroba.com/instalar-jupyter-notebook-y-jupyterlab-por-consola-desde-cero-y-aprender-a-usarlos/

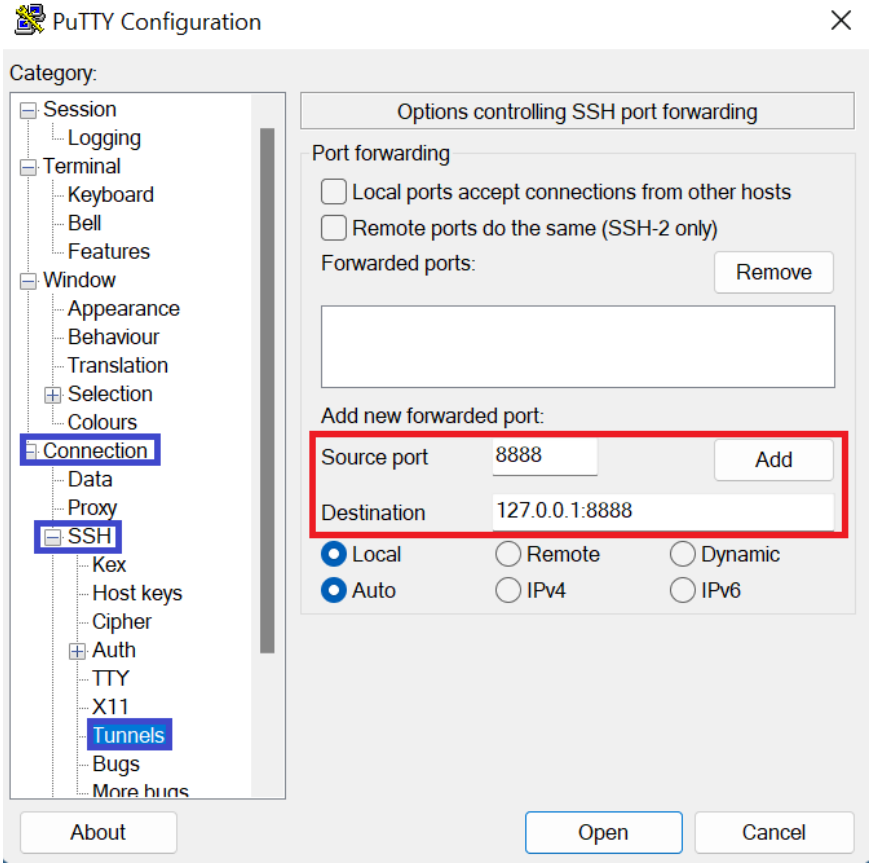


Figura 2.2. Putty Port

una vez allí, ejecutar el comando `jupyter lab`. Esto, nos mostrará por pantalla un enlace que deberemos copiar y pegar en el navegador. Una vez lo hayamos hecho, nos aparecerá en la pestaña la Figura 2.3. Al hacer clic en el icono *Notebook*, se nos abrirá una pestaña en la que podremos comenzar a escribir nuestros códigos en Python.

Por lo tanto, cuando hayamos completado todos los pasos mencionados anteriormente, ya tendremos configurado el entorno de desarrollo y podremos empezar a trabajar.

2.2. Librerías para el desarrollo

El lenguaje de programación que se va a utilizar mayoritariamente es Python [1]. La elección de este lenguaje frente a otros se debe a las múltiples ventajas que posee: se trata de un lenguaje multiplataforma, es decir, que puede ser ejecutado en casi cualquier sistema operativo siempre que se cuente con un intérprete adecuado para ello; es un lenguaje de código abierto, por lo que no se

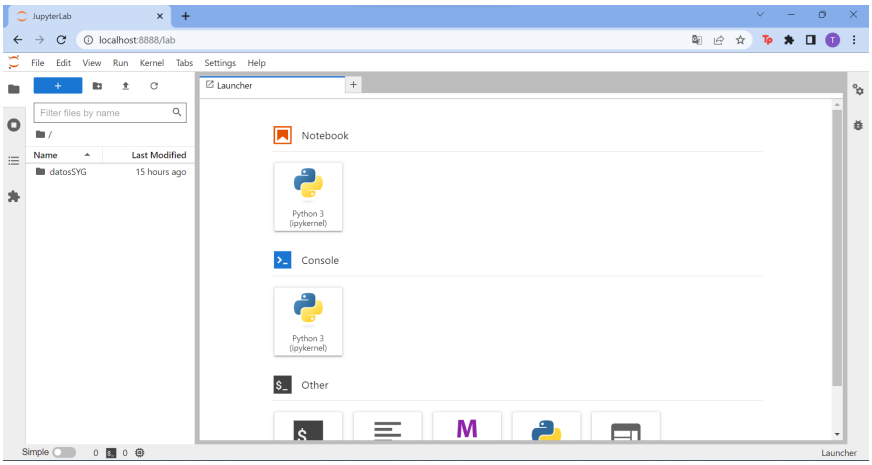


Figura 2.3. Jupyter Lab

requieren licencias de pago para comenzar a trabajar con él; es un lenguaje interpretado, por tanto, ejecuta el código línea a línea traduciéndolo directamente a "lenguaje máquina"; y es un lenguaje de tipado dinámico, así que no debemos especificar el tipo de las variables que se usen. Por otro lado, Python cuenta con múltiples librerías científicas que nos ofrecen una gran variedad de módulos que facilitarán nuestro trabajo en el contexto del análisis de datos. Una librería es una colección de códigos usados con frecuencia que los desarrolladores pueden incluir en sus programas de Python para evitar tener que escribir el código desde cero. Existen más de 137.000 disponibles. A continuación, se describen brevemente las que usaremos en este trabajo.

2.2.1. Pandas

Pandas [5] es una librería de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para que trabajar con datos relacionales, es decir, datos con relaciones predefinidas entre ellos, sea fácil e intuitivo.

El tipo de datos fundamental en pandas y sobre el que giran la mayoría de operaciones que podemos realizar con esta librería recibe el nombre de *dataframe*. Es una estructura de datos bidimensional que permite almacenar información en forma de filas y columnas y es muy útil para realizar análisis de datos.

Pandas cuenta con herramientas para leer y escribir datos en diferentes formatos, para transformar y rotar conjuntos de datos flexibles, para la segmentación inteligente basada en etiquetas, para la fusión y unión de conjuntos de datos, etc. Actualmente, es una de las mejores herramientas de manipulación y análisis de datos de código abierto disponible en cualquier idioma.

Esta librería tiene módulos que nos serán de mucha utilidad para ir haciendo modificaciones en nuestros conjunto de datos en este trabajo.

2.2.2. Datetime

Datetime [6] proporciona clases para manipular fechas y horas. Esta librería incorpora los tipos de datos siguientes:

- `Date`: cuyos componentes son el año, el mes y el día.
- `Time`: cuyos componentes son la hora, los minutos, los segundos y microsegundos.
- `Datetime`: cuyos componentes son la fecha, con año, mes y día, seguido de la hora, minutos, segundos y microsegundos.

Nos será de gran utilidad a la hora de acceder a los distintos componentes de una fecha como el año, mes o día, para extraer nuevos campos a partir de ellas, para convertir cadenas al formato de fecha y viceversa o para hacer aritmética de fechas, como sumarlas, restarlas o compararlas.

2.2.3. Pandera

Pandera [7] se encarga de realizar un proceso de validación de datos almacenados en tablas. Con ella, definimos un esquema con condiciones que deben cumplir nuestros datos y lo utilizamos para validarlos. En este esquema, pueden estar incluidas condiciones como el tipo de datos que debe almacenar una columna, el rango en el que se deben encontrar ciertos valores, si esos valores pueden ser nulos o no, etc.

En capítulos posteriores profundizaremos un poco más en su uso e importancia.

2.2.4. Plotly

Plotly [8] es una librería con la que podremos crear visualizaciones web que se pueden mostrar en cuadernos de Jupyter, guardar en archivos HTML independientes o servir como parte de aplicaciones web. Admite más de 40 tipos de gráficos que cubren una amplia gama de casos de uso.

Una de las características más destacadas de Plotly y la que me ha hecho decidirme por ella en este proyecto, es su capacidad para crear gráficos interactivos. El usuario puede habilitar funciones de zoom, panorámica, selección de puntos y herramientas de exportación en los gráficos generados.

Además, haremos uso de algunas de sus sublibrerías como Plotly Express, que nos permitirá visualizar nuestros gráficos de forma inmediata.

2.2.5. Dash

Dash [10] es una de las mejores herramientas para crear aplicaciones web interactivas. Permite a los usuarios crear paneles de control completos e interactivos utilizando Python y haciendo uso de su gran variedad de diseños.

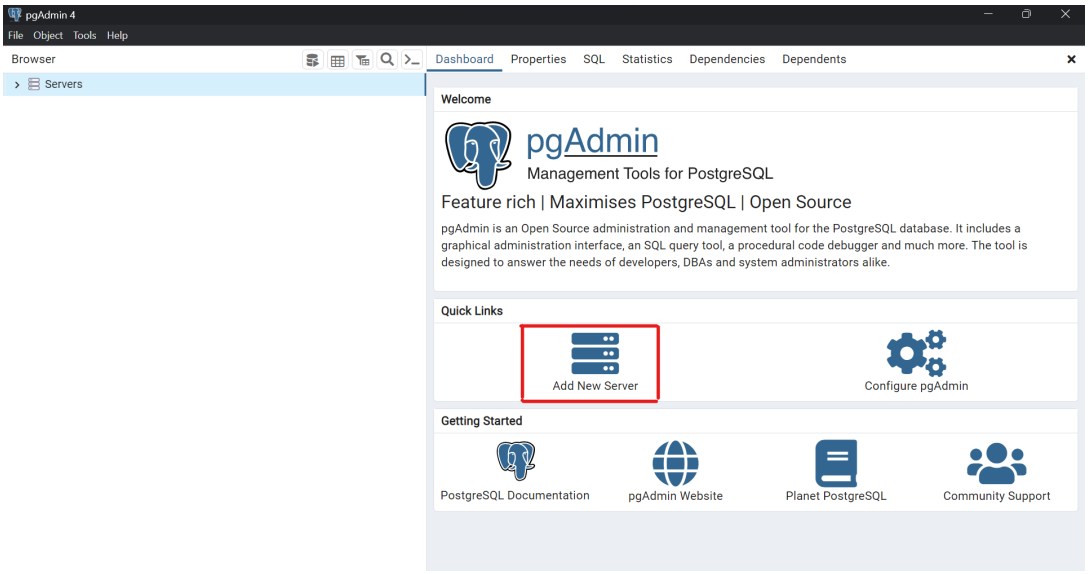


Figura 2.4. Inicio PgAdmin

La elección de esta librería para la realización del proyecto se debe a que Dash permite definir funciones de tipo callback, que son funciones que se activan en respuesta a eventos, como hacer clic en un seleccionador o cambiar un valor en un control deslizante. Esto permite la actualización en tiempo real de los componentes y la interacción fluida con la aplicación sin la necesidad de ejecutar un script varias veces tras realizar cambios.

2.2.6. Psycopg2

Psycopg2 [9] es el adaptador de base de datos de PostgreSQL [11] más popular para Python. Con él, creamos una conexión a la base de datos con la que vamos a trabajar utilizando los parámetros de conexión, como el nombre del host, el puerto, el nombre de usuario y la contraseña.

La he elegido ya que nos permite seguir trabajando desde Python creando un cursor con el que se ejecutan todos los comandos SQL que nos sean necesarios para el manejo de los datos en las bases creadas.

2.3. Gestión de bases de datos

El uso de bases de datos para el desarrollo del proyecto se debe a que, en el ámbito empresarial, es muy común que el volumen de los datos vaya creciendo y puede que esto dificulte nuestro trabajo, tanto por las limitaciones de almacenamiento como de procesamiento y organización.

Las bases de datos están diseñadas para manejar grandes cantidades de datos. Cuentan con herramientas que facilitan la optimización del rendimiento,

obteniendo resultados de una forma más rápida y eficiente. También, permiten definir un esquema de datos, establecer relaciones y restricciones entre ellos, ofreciendo así un modelo de datos consistente, lo que facilita la integración y gestión de los mismos desde diferentes fuentes en proyectos de análisis de datos más complejos. Por otro lado, la seguridad y el control de acceso a los datos son preocupaciones importantes para una empresa. Las bases de datos ofrecen mecanismos de seguridad avanzados, como autenticación y autorización, que permiten proteger los datos sensibles y garantizar el cumplimiento de regulaciones de privacidad y seguridad. Además, muchas herramientas de análisis de datos están diseñadas para trabajar únicamente con bases de datos, lo que nos puede facilitar el trabajo futuro.

Aunque el lenguaje que usaremos mayoritariamente será Python, también haremos uso de *Structured Query Language* (SQL) [12]. Es un lenguaje de programación que se utiliza para almacenar y procesar información en una base de datos relacional, que es aquella que almacena información en forma de tabla, con filas y columnas que representan diferentes atributos de datos y las diversas relaciones entre los valores. Las instrucciones SQL se utilizan para almacenar, actualizar, eliminar, buscar y recuperar información de la base de datos.

Este lenguaje será utilizado en PostgreSQL, un sistema gratuito y de código abierto que sirve para la gestión de bases de datos relacionales realizando consultas en lenguaje SQL. Lo he elegido frente a otras opciones gratuitas de este tipo de sistemas como MySQL, MariaDB o SQLite porque es la mejor elección en el caso de proyectos grandes y complejos con operaciones con grandes volúmenes de datos.

Para poder administrar las bases de datos de PostgreSQL donde se almacenarán los datos he elegido usar PgAdmin [13], que es una *Graphical User Interface* (GUI). Este tipo de herramientas actúan de interfaz de usuario utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles, haciendo así que las interacciones sean más visuales y fáciles de usar. PgAdmin se puede ejecutar como una aplicación web o de escritorio y se utiliza para realizar cualquier tarea de administración de bases de datos de PostgreSQL en servidores locales y remotos.

Para comenzar a configurar la base de datos, lo primero que hay que hacer es descargar PgAdmin en su versión más reciente, la cual se encuentra disponible en el siguiente enlace [3]. Al iniciar, PgAdmin solicitará una contraseña. Al rellenarla, veremos lo que se muestra en la Figura 2.4. Haremos clic en *Add New Server*, lo que abrirá la ventana que se muestra en la Figura 2.5. En primer lugar, accederemos a la pestaña *General*, en la que pondremos el nombre que recibirá el nuevo servidor que vamos a registrar. En segundo lugar, iremos a la pestaña *Connection*, donde rellenaremos los datos del servidor de PostgreSQL, incluida

[3] https://www.pgadmin.org/download/pgadmin-4-windows/

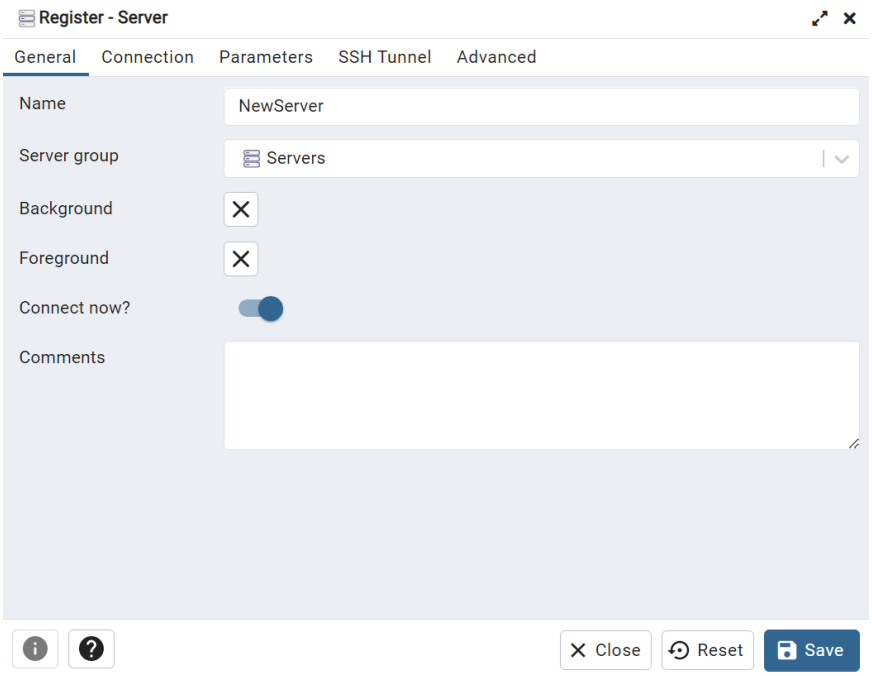


Figura 2.5. PgAdmin Add New Server

la contraseña que hemos elegido al iniciar (veáse la Figura 2.6). A continuación, nos situaremos en la pestaña *SSH Tunnel*, donde realizaremos un segundo túnel SSH a través de PgAdmin, que nos ayudará a conectar este servidor con el servidor en el que se encuentran nuestros datos, rellenando los campos tal y como se muestra en la Figura 2.7. De esta forma, ya tendremos todo listo para poder empezar a trabajar.

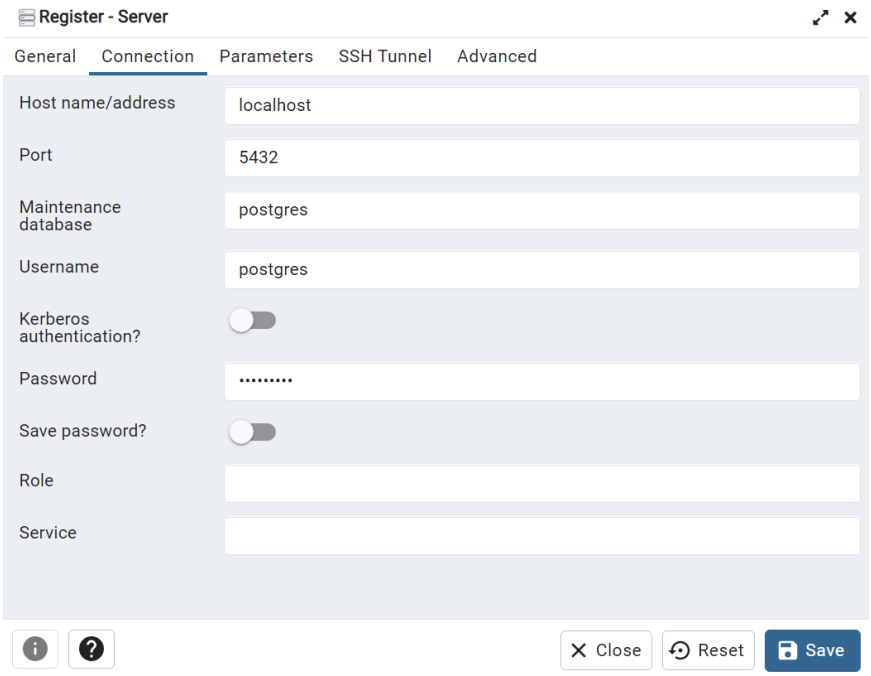


Figura 2.6. PgAdmin Connection

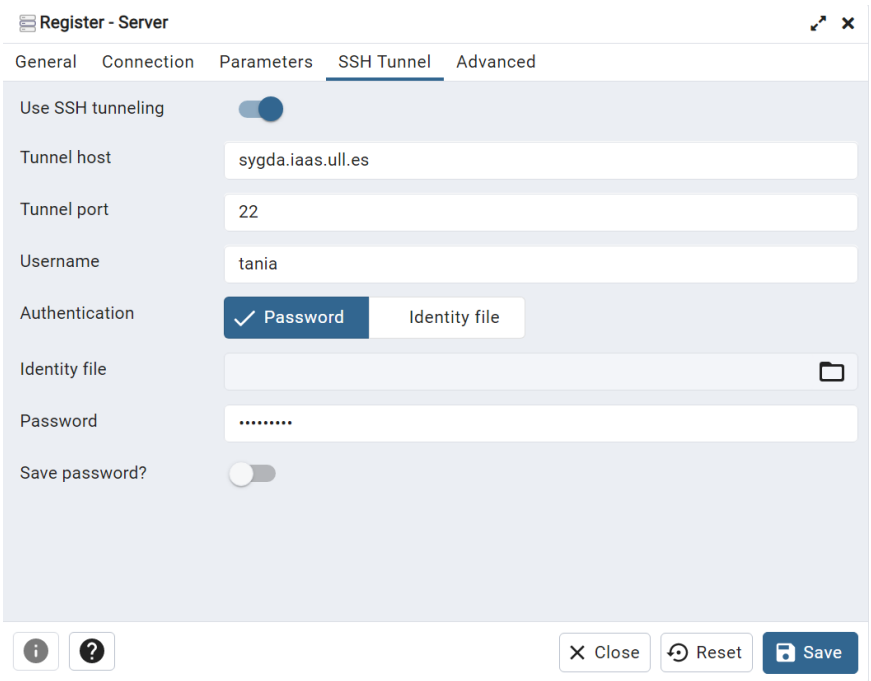


Figura 2.7. PgAdmin SSH Tunnel

3

Inteligencia de negocio

Inteligencia de negocio o *Business Intelligence* (BI) es el proceso mediante el cual las empresas utilizan métodos y tecnologías estadísticas para analizar datos con el fin de obtener nuevos conocimientos y mejorar la toma de decisiones estratégicas. El proceso BI se divide en cinco fases, las cuales se explican mejor teniendo de referencia la Figura 3.1 [14]. Estas fases no se llevan a cabo de manera lineal, sino que son cíclicas y se repiten a lo largo del tiempo para mejorar de forma continua la toma de decisiones estratégicas de la empresa.

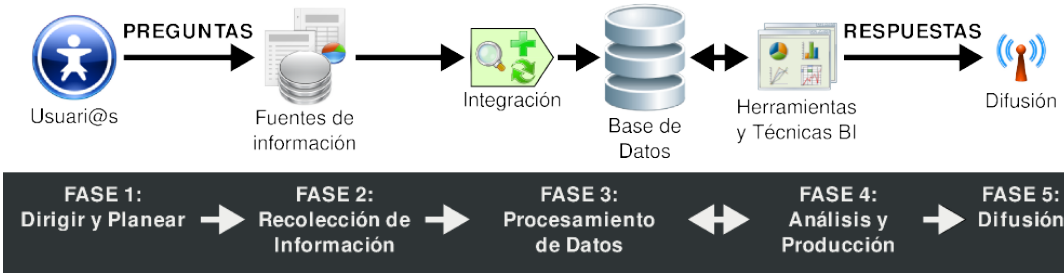


Figura 3.1. Pasos BI
Fuente: https://www.dataprix.com/es/book/export/html/441

A continuación, se explicarán cada una de ellas. Primero, de una forma teórica para poder definir algunos conceptos necesarios que facilitarán la comprensión este tipo de análisis. Luego, nos centraremos en el contexto específico de nuestro caso, aportando una visión general de los pasos seguidos en este proyecto en particular. Y, por último, profundizaremos en el modelo de datos, el cual resulta crucial para comprender su formato de almacenamiento, así como su significado e importancia en el contexto del proyecto.

3.1. Marco teórico

En primer lugar, debemos **dirigir y planear** nuestro análisis. En esta fase inicial debemos ponernos en contacto con la empresa para que nos comunique de forma detallada los objetivos de mejora que quieren alcanzar. A partir de ellos, definiremos cuáles van a ser nuestros indicadores de rendimiento clave o *Key Performance Indicators* (KPI). Estos indicadores son métricas que reflejan el rendimiento de una empresa y con los que podremos crear visualizaciones que animen o respalden a la empresa a la hora de tomar decisiones estratégicas. Además, deberá aclararse dónde se encuentran los datos que se necesitan para el análisis.

Estos primeros pasos son muy importantes ya que, dependiendo de los objetivos, encaminaremos el análisis de una forma u otra y podremos saber qué herramientas nos serán necesarias.

Una vez definidos los objetivos, comenzará el desarrollo de la solución BI, cuya arquitectura se muestra en la Figura 3.2.

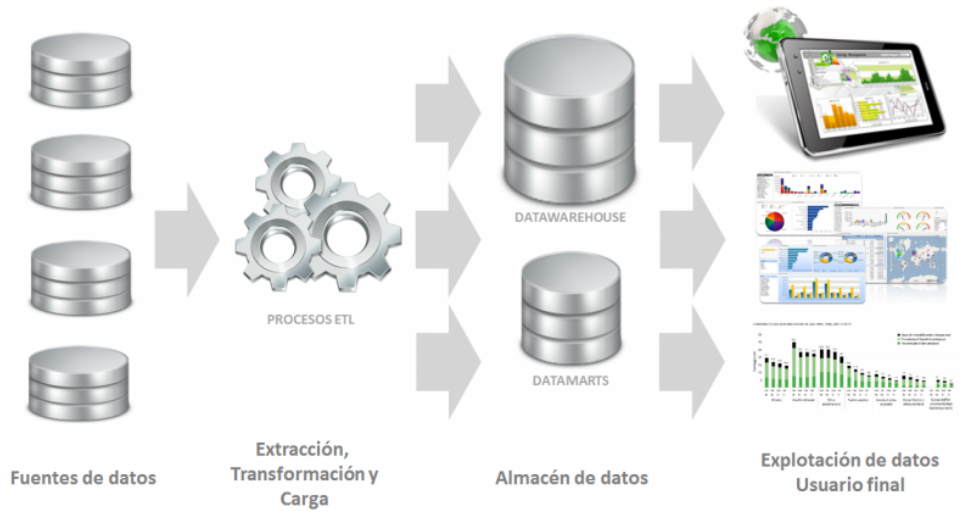


Figura 3.2. Arquitectura solución BI
Fuente: https://blog.mirai-advisory.com/que-es-business-intelligence/

Empezaremos por **la recolección de la información**. Estudiaremos las diferentes fuentes de información de la empresa para recolectar aquellos datos que darán respuestas a las necesidades planteadas en la fase anterior. Haremos uso de diferentes herramientas informáticas para la lectura de los mismos.

El siguiente paso es **el procesamiento de los datos**. Una solución BI se inicia desde los sistemas de origen de la empresa, en los que suele ser necesario modificar su estructura para facilitar el proceso de análisis.

Para ello, suele ser necesario crear un almacén de datos intermedio, que recibe el nombre de *Operational Data Store* (ODS), en el que integraremos los datos en crudo en un formato utilizable para el análisis. De esta forma, evitamos la saturación de los servidores de la empresa.

Luego, se realiza un proceso llamado *Extract, Transform and Load* (ETL), donde se depuran y homogeneizan. Además, el preprocesamiento de datos ayuda a identificar y corregir posibles errores y permite una mejor interpretación y comprensión de los mismos.

Una vez depurada, validada y consolidada la información, crearemos una segunda base de datos para almacenarla. Esta recibe el nombre de *Data Warehouse* (DW). La información que se almacena en un DW debe ser homogénea y fiable. Estas bases de datos poseen múltiples ventajas como que son orientadas a temas, integradas, no volátiles y reflejan las variaciones con respecto al tiempo. El modelo dimensional de un DW se organiza en "hechos", que son las tablas principales del modelo y contienen aquellas medidas que queremos analizar y "dimensiones", que son las tablas que almacenan aquellos atributos cualitativos que podemos unir a los hechos para añadir información adicional. Además, existen los "hechos derivados", que son medidas que podemos crear a partir de otros hechos.

Existen tres tipos de modelados distintos:

- Esquema en estrella: Consta de una tabla de hechos central que se puede relacionar con múltiples dimensiones a partir de sus claves, como se muestra en la Figura 3.3.

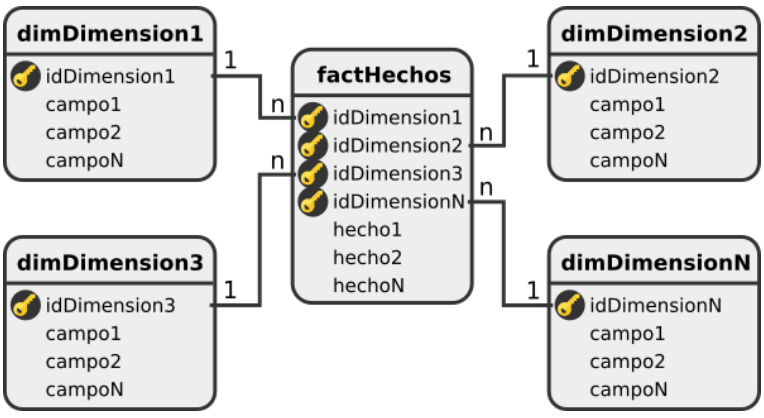


Figura 3.3. Esquema en estrella
Fuente: https://troyanx.com/Hefesto/estrella.html

- Esquema en copo de nieve: Está formado por una tabla de hechos central pero, a diferencia del esquema en estrella, esta está unida a dimensiones que

a su vez se unen con otras dimensiones, tal y como se muestra en la Figura 3.4.

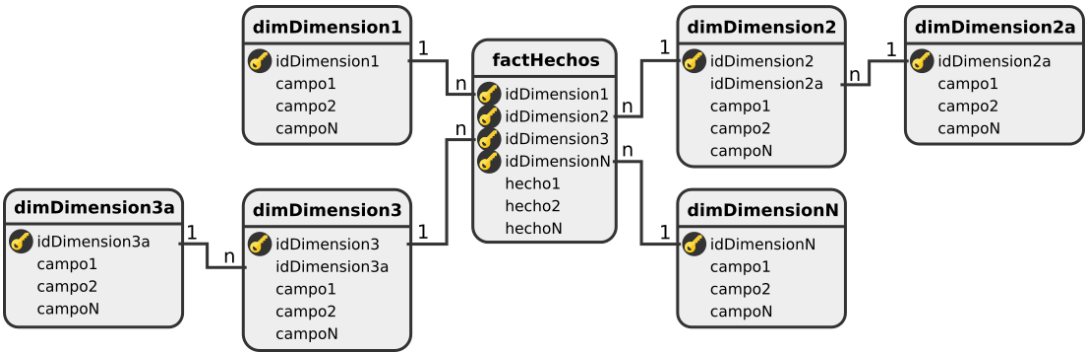


Figura 3.4. Esquema en copo de nieve
Fuente: https://troyanx.com/Hefesto/copo-de-nieve.html

- Esquema en constelación: Está compuesto por varios modelos en estrella. Pueden aparecer varias tablas de hechos conectadas por dimensiones entre ellas, como se muestra en la Figura 3.5.

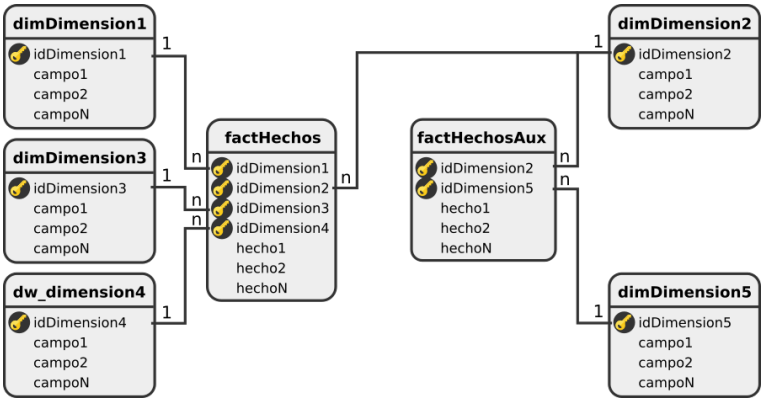


Figura 3.5. Esquema en constelación
Fuente: https://troyanx.com/Hefesto/constelacion.html

Una vez integrados los datos, procederemos a **su análisis y producción**. Aquí, se realiza la validación de los datos, la creación de nuevos campos calculados a través de otros, las visualizaciones...

La visualización de datos es una herramienta clave en el análisis de datos porque permite una comprensión más profunda y significativa de los mismos. Los datos son una colección de información, pero cuando se presentan de forma visual, se pueden identificar patrones, tendencias, relaciones y anomalías que

pueden no ser tan obvias en una tabla o en una colección de datos. También, ayuda a simplificar los datos complejos y a resumirlos de manera clara y concisa, lo que permite una comunicación más efectiva de los resultados a un público más amplio.

Como resultado final de esta fase se obtendrán las respuestas a las preguntas planteadas en la primera mediante la creación de reportes, indicadores de rendimiento, cuadros de mando, gráficos estadísticos, etc.

Finalmente, se encuentra **la difusión**. En esta fase, los usuarios haciendo uso de diferentes herramientas, podrán explorar los datos que deseen de manera sencilla e intuitiva gracias al cuadro de mando que hemos creado. De esta forma, podrán tener una idea clara de sus activos y podrán llevar un seguimiento detallado de su rendimiento y evolución cuando lo deseen.

3.2. Contexto empresarial

Se trata de una empresa de transporte discrecional de viajeros. Este tipo de transporte se ofrece de manera privada y personalizada, generalmente para grupos de personas, como turistas, empresas, organizaciones o particulares. A diferencia del transporte público regular, el transporte discrecional no sigue una ruta o horario fijo, sino que se adapta a las necesidades específicas de los clientes.

El funcionamiento de este tipo de empresas, comienza con la recepción de solicitudes por parte de los clientes interesados en el servicio. Una vez recibida la solicitud, la empresa recopila todos los detalles necesarios, como la fecha del viaje, el número de pasajeros, la duración del servicio y la ruta requerida. En base a esta información, se proporciona al cliente una cotización detallada que incluye el costo del servicio.

Cuando se realiza la reserva, comienza la preparación de los servicios a realizar. Cada uno de ellos, según lo requiera el tamaño del grupo y la duración del viaje, deberá tener asignado un conductor y un vehículo de la flota de la empresa, que se divide en vehículos para transporte colectivo (autobús o guagua) y turismos. Es importante destacar que los conductores no tienen un vehículo fijo asociado, sino que serán asignados a cada servicio según la disponibilidad y los requisitos específicos.

Los servicios realizados tendrán asociados tanto ingresos, como gastos operativos en los que se incluyen, por ejemplo, el mantenimiento y combustible necesario para los vehículos, los salarios de los conductores, los seguros, el alquiler de instalaciones y los costos administrativos...

Es importante llevar una correcta gestión de ellos para asegurar el rendimiento de la empresa y su éxito a largo plazo.

3.3. Modelo de datos

La empresa cuenta con su propio suministro de combustible en los almacenes, que ha sido proporcionado por una distribuidora de Tenerife a un precio variable según el mes. A los repostajes con este combustible los denominaremos repostajes internos. Aún así, no siempre es suficiente con estos repostajes y durante algunos trayectos, los vehículos que no disponen del combustible necesario, se ven obligados a repostar en áreas de servicio. A estos repostajes los llamaremos repostajes externos.

Además, los vehículos tienen otros gastos como pagos de seguros, revisiones, reparos en el caso de avería, repuestos... A este tipo de gastos los llamaremos gastos de mantenimiento.

Por otro lado, contamos con la información relativa a los ingresos por cada servicio que realiza el vehículo. Estos ingresos reciben el nombre de facturación.

Toda esta información está almacenada en documentos excel, que comprenden desde el 1 de enero hasta el 17 de noviembre de 2022.

En `RepostajesExternos2022`, encontraremos los datos relacionados con los repostajes externos. Este documento está formado por 14 columnas y 5061 filas de datos. Se trabajará solo con las seis columnas siguientes:

- `Fecha`: las fechas en las que se realizaron los repostajes.
- `Hora`: la hora en la que se realizó el repostaje.
- `Matricula`: la matrícula del vehículo que realizó el repostaje.
- `Litros`: los litros repostados.
- `Kms`: los kilómetros que llevaba recorridos el vehículo en el momento del repostaje.
- `Importe`: el costo que supuso el repostaje.

Aunque cuenta con otra información disponible como:

- `Conductor`: los conductores que realizaron el repostaje externo. Esta columna no se utilizará para el análisis pues cada vehículo no tiene un conductor fijo asociado, así que no nos proporciona información de interés para el seguimiento de la flota
- `Tipo`: el tipo de combustible repostado. Esta información no se utiliza ya que el único tipo de combustible repostado es gasoil.
- `Proveedor`: la estación de servicio en la que se realizó el repostaje. Del mismo modo que el tipo de combustible, solo existe un único proveedor, por lo que no nos proporciona información de interés para el seguimiento de la flota.

`GastosPorMatricula2022` contiene los gastos de mantenimiento por mes de cada vehículo. En estas cifras vienen incluidos repuestos y mano de obra en caso de averías, pagos de ITV, seguros, etc. Este fichero tiene 158 filas de datos y 15 columnas de las que solo usaremos las siguientes:

- `Matricula`: las matrículas de la empresa.
- `1-12`: doce columnas con números del uno al doce que representan cada uno de los meses del año.

En `FacturaciónPorVehículo2022` se encuentran los datos relativos a la facturación de cada vehículo. Este documento tiene 180 filas de datos y su formato es similar al de `GastosPorMatricula2022` ya que cuenta con las mismas columnas. Por tanto, trabajaremos con las mismas que mencionamos para los gastos.

En `Matriculas` encontraremos dos columnas:

- `Matricula`: las matrículas que forman parte de la empresa.
- `Empresa`: las empresas que forman parte del conglomerado a la que pertenece cada matrícula.

El documento `Precios` está formado por dos columnas:

- `Mes`: los nombres de los meses del año.
- `Precio`: precio por litro para cada mes en repostajes internos.

Por último, en `RepostajesInternos2022` están almacenados los datos correspondientes a los repostajes internos que se realizan. Está compuesto por 40 columnas y 16692 filas de datos. Al igual que en los repostajes externos, muchas de estas columnas no nos aportan información de interés para el análisis, así que solo haremos uso de las cuatro columnas siguientes:

- `Fecha`: las fechas y horas en las que se realizaron los repostajes.
- `Nombre`: la matrícula del vehículo que realizó el repostaje.
- `Unid.`: los litros repostados.
- `Kilómetros`: los kilómetros que llevaba recorridos el vehículo en el momento del repostaje.

4

Desarrollo de la solución

En este capítulo se describirá la implementación de todo el proceso de análisis. Veremos cómo podemos acceder a los datos para su recopilación y posterior carga a la base de datos correspondiente. Además, se llevará a cabo su procesamiento, validación y transformación a través de códigos Python según sea necesario. Por último, realizaremos la carga de los datos validados y se mostrarán visualizaciones de los mismos.

4.1. Definición de objetivos

La empresa busca soluciones automatizadas que le ayuden a llevar un seguimiento de la flota.

Les gustaría obtener cuadros de mando que muestren indicadores como la facturación, los gastos y el margen bruto por mes y empresa, así como un progreso por mes de estos indicadores para cada vehículo. Por otro lado, les gustaría ver estos indicadores desglosados por vehículo, diferenciando por ejemplo en el número de litros repostados en sus almacenes y en estaciones de servicio o en el gasto que supusieron los repostajes externos frente a los internos.

Estos cuadros de mando deben tener ciertos filtros con los que se muestre únicamente la información que buscan en cada momento.

4.2. Recolección de la información

Los datos necesarios para el análisis se encuentran almacenados en seis archivos Excel:

- `RepostajesExternos2022`
- `GastosPorMatrícula2022`
- `RepostajesInternos2022`
- `FacturaciónPorVehículo2022`
- `Precios`

- `Matriculas`

Los documentos Excel se encuentran en una carpeta de Google Drive. Debido a la confidencialidad de los datos, debemos tener cuidado y no descargarlos en un equipo local. Por eso, haciendo uso de Putty, crearemos un directorio llamado `datosTFG` con el siguiente comando.

```
mkdir datosTFG
```

Ahí, subiremos nuestros archivos Excel para poder acceder a ellos desde Jupyter Notebooks. En primer lugar, debemos ir al archivo que queremos subir, copiar su enlace para compartir y asegurarnos de que cualquier persona con el enlace puede acceder a él. En segundo lugar, nos situaremos en el directorio que acabamos de crear y donde queremos subir los archivos, para ejecutar el siguiente comando:

```
wget "URL del archivo"
```

Realizaremos este proceso con todos los archivos mencionados anteriormente. Cuando lo hayamos hecho, tendremos a nuestra disposición todos los datos y podremos acceder a ellos para visualizarlos y comprender su contenido.

Como hemos mencionado en el capítulo anterior, será necesario crear una base de datos intermedia a la que llamaremos `bd-ODS`. La crearemos en PostgreSQL a través de PgAdmin. Primero, debemos realizar en PgAdmin la conexión con PostreSQL tal y como se explica en Entorno de trabajo 2.3.

Una vez estemos conectados, hacemos clic con el botón derecho del ratón sobre el nombre del servidor que hemos registrado previamente. Se desplegarán las diferentes opciones disponibles. Como se muestra en la Figura 4.1, seleccionaremos *Create* y luego *Database*. Después, se desplegará una ventana en la que nos pedirán el nombre de la base de datos que queremos crear. Una vez lo hayamos puesto, hacemos clic sobre *Save* y se habrá creado. Luego, realizaremos la conexión a la base de datos a través de la librería psycopg2.

Con el Código 4.1, importamos la librería y rellenamos los campos que aparecen con el nombre de la base de datos con la que queremos conectar, nuestro usuario y contraseña junto con el host y el puerto correspondiente.

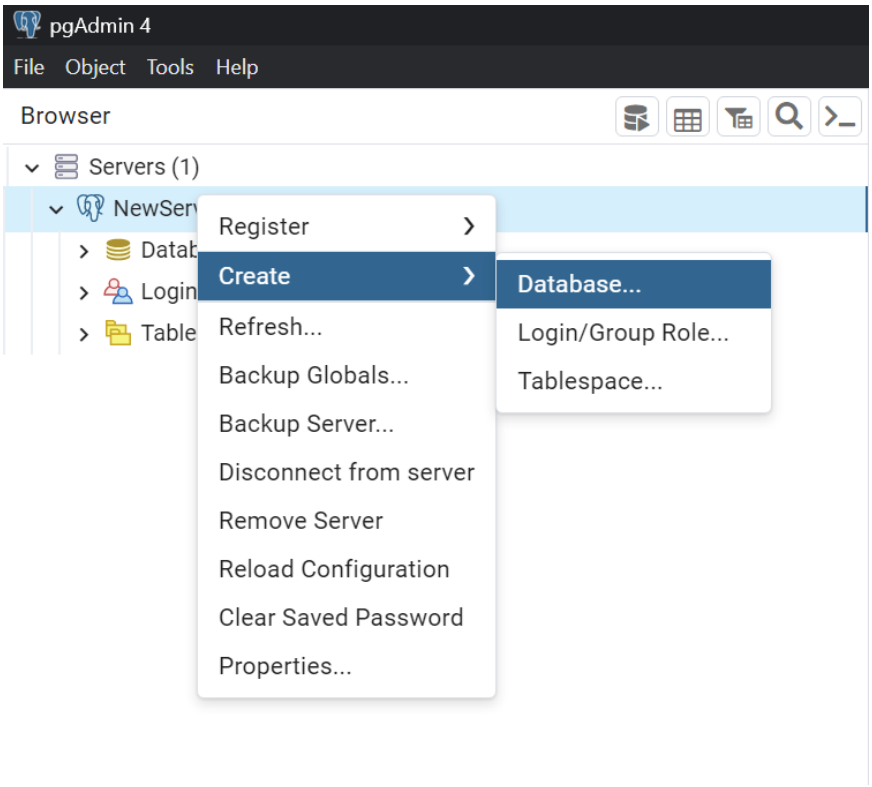


Figura 4.1. Crear base de datos

```
import psycopg2

conexion = psycopg2.connect(
    dbname='bd-ODS',
    user='tania',
    password='contrasena',
    host='localhost',
    port='5432'
)
```

Código 4.1. Conexión bd-ODS

El siguiente paso, será crear las tablas en las que vamos a almacenar nuestros datos, empezando con la de repostajes externos. Para ello, debemos ejecutar el Código 4.2.

Con él, en la línea 11, definimos un cursor a partir de la conexión a la base de datos que hemos creado. Lo utilizaremos para recorrer y manipular filas en los dataframes que extraeremos de las consultas a las tablas que se encuentran en `bd-ODS`. En segundo lugar, definimos una cadena de texto en la que almacenaremos la consulta SQL necesaria para crear la tabla. Además de las columnas mencionadas anteriormente, crearemos una columna adicional

para cada tabla llamada `Validación`, que luego utilizaremos para el proceso de validación de los datos. Luego, se ejecuta con el método `execute` del cursor.

```
cursor = conexion.cursor()

# Tabla de repostajes externos
create_repostex = '''CREATE TABLE repost_ext (
    Fecha DATE NOT NULL,
    Hora TIME NOT NULL,
    Matricula VARCHAR(20) NOT NULL,
    Litros_ext FLOAT NOT NULL,
    Kilometros INTEGER NOT NULL,
    Importe_ext FLOAT NOT NULL,
    Validacion VARCHAR(50)
);'''
```

Código 4.2. Creación de tablas

Haremos lo mismo con todas, cambiando los nombres de las tablas y de las columnas según corresponda, acompañados del tipo de datos que almacena cada una. Luego, cuando estén todas creadas, el siguiente paso será leer los documentos excel y llenar cada una de las tablas con la información que corresponda. Lo haremos como sigue en el Código 4.3 para el caso de los repostajes externos. Con él, iremos recorriendo las filas de la tabla de repostajes externos con la función `iterrows` y las iremos incorporando en las columnas correspondientes de la tabla que creamos en la base de datos. Luego, crearemos un código similar, haciendo los cambios necesarios en los nombres de las tablas y columnas, para el resto de tablas.

```
# Lectura de tablas desde excel
repostex = pd.read_excel('/home/tania/datosSYG/RepostajesExternos2022.
    xlsx')


# Tabla de repostajes externos
for index,row in repostex.iterrows():
    Fecha = row['Fecha']
    Matricula = row['Matricula']
    Hora = row['Hora']
    Litros_ext = row['Litros']
    Kilometros = row['Kms']
    Importe_ext = row['Importe']
    cursor.execute("INSERT INTO repost_ext (Fecha,Matricula,Hora,
    Litros_ext,Kilometros,Importe_ext) VALUES (%s,%s,%s,%s,%s,%s)", (
    Fecha, Matricula,Vehiculo,Hora,Litros_ext,Kilometros,Importe_ext))
```

Código 4.3. Relleno de tablas

Ahora, podremos seguir trabajando cargando los datos desde las tablas que hemos creado usando el Código 4.4 para el caso de repostajes externos y de forma similar para el resto.

```
consulta_repostex = '''
    SELECT Fecha,
        Hora,
        Matricula,
        Litros_ext,
        Kilometros,
        Importe_ext,
        Validacion
    FROM repost_ext;
    '''
cursor.execute(consulta_repostex)
results_ext = cursor.fetchall()

conexion.commit()
cursor.close()
conexion.close()
```

Código 4.4. Carga de tablas

Con el método `execute` ejecutamos el código SQL correspondiente y gracias a `fetchall`, guardamos el resultado en la variable `results_ext`. Por último, con el método `commit` confirmamos los cambios en la base de datos y, finalmente, cerramos el cursor y la conexión con el método `close`.

Por tanto, al terminar esta fase, tendremos una tabla en la base de datos por cada documento excel proporcionado por la empresa.

Finalmente, accedí a cada una de ellas para comprobar que la carga se había completado correctamente. Al visualizar los datos me encontré con algunos inconvenientes que debía resolver como datos erróneos o diferencias de formato a la hora registrar repostajes. Aparecían kilómetros recorridos y litros repostados que excedían límites lógicos así como matrículas incompletas, inexistentes en la empresa o con un formato erróneo, pues algunas tenían espacios al principio o al final y otras no. Además, los kilómetros de un vehículo deben aumentar a medida que pasa el tiempo. En algunos casos, eso no era así.

Por esta razón, además del procesamiento de los datos, se deberá realizar un proceso de validación.

4.3. Procesamiento

Empezaremos con la tabla `FacturaciónPorVehículo2022`. Para realizar el análisis, nos gustaría que tuviera un formato similar al resto de tablas: una columna llamada *Facturacion* que contenga los valores y otra columna llamada *Mes* que contenga los nombres de los meses correspondientes. Además, si queremos hacer análisis futuros con los datos por empresa, deberíamos tener una columna llamada *Empresa*, que nos separe los datos por las mismas. Para ello, vamos a ejecutar el Código 4.5.

```
1  import pandas as pd
2
3  mes = ['Enero','Febrero','Marzo','Abril','Mayo','Junio','Julio',
4  'Agosto','Septiembre','Octubre', 'Noviembre', 'Diciembre']
5
6  columnas_matr = ['Matricula','Empresa']
7  columnas_factu = ['Mes','Matricula','Empresa', 'Facturacion']
8
9  matriculas = pd.DataFrame(results_matriculas, columns = columnas_matr )
10 fact = pd.DataFrame(results_fact, columns = mes)
11
12 fact['Matricula'] = fact['Matricula'].str.strip()
13 factur = pd.merge(fact,matriculas,on=['Matricula'],how='outer')
14 fact_melted = pd.melt(factur, id_vars=['Matricula','Empresa'],
       value_vars= mes, var_name='Mes', value_name='Facturacion')
15 fact_melted = fact_melted.dropna(subset=['Matricula'])
16 fact_melted = fact_melted.fillna(0)
17 factu = fact_melted[columnas_factu]
```

Código 4.5. Preprocesado de facturación

Primero, importamos la librería Pandas y creamos una lista llamada `mes` que contendrá los nombres de los meses. Luego, después de eliminar los posibles espacios existentes en las matrículas de los vehículos para evitar problemas en su asociación, uniremos la tabla `matriculas` haciendo uso de la función `merge`. Así, podremos asignar a cada vehículo su empresa correspondiente para hacer futuros análisis. Al realizar esta unión, si existen empresas en la tabla `matriculas` que no están asociadas a ninguna matrícula de la tabla `fact`, aparecerán valores nulos en la columna `Matricula`, que eliminaremos mediante la función `dropna` en la línea 14. Con la función `melt`, podremos transformar la tabla de ancho a largo, es decir, las columnas que contenían los nombres de los meses ahora pasarán a ser una columna llamada `Mes` en la que aparecerán dichos nombres y los valores de facturación se mostrarán en una columna llamada `Facturacion`. Luego, con `fillna` cambiamos los valores nulos de nuestra tabla por ceros para poder hacer cálculos con ellos en pasos futuros. Por último, escogemos las columnas que utilizaremos para el análisis, que en este caso serán `Matricula`, `Mes`, `Empresa` y `Facturación`.

Ya que `GastosPorMatricula2022` tenía un formato parecido, utilizaremos una implementación similar a la mostrada en el Código 4.5, a excepción de algunos cambios como el nombre del dataframe `fact` por `gast` y la columna `Facturacion` por `GastosMant`. Sin embargo, tanto el archivo `RepostajesExternos2022` como el archivo `RepostajesInternos2022` ya tenían este formato, por lo que no vamos a necesitar las mismas transformaciones. En el caso de `RepostajeExternos2022` vamos a necesitar el Código 4.6.

```
import datetime

columnas_ext = ['Fecha','Hora','Matricula','Litros_ext','Kilometros',
                'Importe_ext','Validado']

mes_num = {1: 'Enero', 2: 'Febrero', 3: 'Marzo', 4: 'Abril', 5: 'Mayo',
           6: 'Junio', 7: 'Julio', 8: 'Agosto', 9: 'Septiembre',
           10: 'Octubre', 11: 'Noviembre', 12: 'Diciembre'}

repostex = pd.DataFrame(results_ext, columns = columnas_ext)
repostex['Fecha'] = pd.to_datetime(repostex['Fecha'])
repostex['Fecha'] = repostex['Fecha'].dt.strftime('%Y-%m-%d')
repostex['Año'] = repostex['Fecha'].dt.year
repostex['Mes'] = repostex['Fecha'].dt.month.map(mes_num)
repostex['Repostaje'] = 'Ext'
```

Código 4.6. Preprocesado de repostajes externos

En él, primero importamos la librería `datetime`, ya que la necesitaremos para hacer modificaciones en la columna fecha, creamos un diccionario llamado `mes_num` que asigna a cada número de mes del uno al doce su nombre correspondiente en español y una lista con las columnas de la tabla que utilizaremos para el análisis. En la línea 11, convertimos la columna fecha a objeto *datetime* y, luego, en la siguiente línea, le damos un nuevo formato. Además, creamos una nueva columna llamada año que almacenará los años de las fechas, una columna mes que almacenará los nombres de los meses en español haciendo uso del diccionario creado previamente y una columna llamada `Repostaje` que contendrá la cadena "ext" para que así, podamos identificar de qué tipo de repostaje se trata en pasos futuros.

Luego, con el Código 4.7 haremos las modificaciones de `RepostajesInternos2022`.

```
columnas_int=['Fecha','Matricula','Litros_int','Kilometros','Validado']
columnas_pr = ['Mes', 'Precios']

repostin = pd.DataFrame(results_int, columns = columnas_int )
precio = pd.DataFrame(results_precios, columns = columnas_pr )

repostin['Fecha_hora'] = pd.to_datetime(repostin['Fecha'])
repostin['Hora'] = repostin['Fecha_hora'].dt.strftime('%H:%M')
repostin['Fecha'] = repostin['Fecha_hora'].dt.strftime('%Y-%m-%d')
repostin['Mes'] = repostin['Fecha'].dt.month.map(mes_num)
repostin['Año'] = repostin['Fecha'].dt.year
repostin = pd.merge(repostin,precio, how='outer', on=['Mes'])
repostin['Importe_int'] = repostin['Litros_int']*repostin['Precios']
repostin['Repostaje'] = 'Int'
```

Código 4.7. Preprocesado de repostajes internos

Con este código, en primer lugar, creamos dos variables que contengan las columnas que utilizaremos para cada tabla y luego, cargamos los datos de ambas en las variables `repostin` y `precio`.

En segundo lugar, además de crear una columna `Repostaje` y extraer el nombre del mes y el año de cada fecha como hemos hecho anteriormente, debemos tener en cuenta que en este caso la columna `Fecha` contiene tanto la fecha como la hora, así que la dividiremos en dos columnas: `Fecha`, en la que únicamente se mostrará la fecha y `Hora`, en la que se mostrará la hora en la que se realizó el repostaje. Por último, asociamos a cada mes su precio por litro correspondiente que aparece en la tabla `precios` con la función `merge`. De esta forma, podremos obtener el gasto en repostajes internos multiplicando la cantidad de litros repostados por su precio correspondiente, el cual almacenaremos en una nueva columna llamada `Importe_int`.

4.4. Validación

La validación es muy importante en un proceso de análisis de datos ya que la calidad de los mismos influye en la confiabilidad y precisión de los resultados. Como mencioné anteriormente, durante el procesamiento de los datos, me encontré con algunos erróneos, incoherentes o faltantes. Es por esto, que fue necesario llevar a cabo un proceso de validación de datos.

Se realiza gracias a un esquema de validación, que nos permitirá clasificar nuestros datos en "válidos" o "inválidos", tal y como se muestra en la Figura 4.2. Los datos que hayan cumplido con los requisitos de validación los consideraremos válidos y podremos seguir trabajando con ellos, mientras que cuando encontremos un dato inválido, lo dejaremos indicado de tal forma que la empresa podrá decidir cómo solucionar estos errores y asegurar la integridad, precisión y confiabilidad de los datos utilizados.

En el caso de los datos de los repostajes, debemos comprobar tanto los kilómetros como los litros, pues son los datos más sensibles a la aparición de errores. Además, debemos asegurarnos de que las matrículas que aparecen son correctas.

En cuanto a los kilómetros, debemos comprobar que, para cada matrícula, a medida que avanza el tiempo el número de kilómetros recorridos nunca disminuye. Además, los valores registrados deben ser menores a un millón, ya que son tomados desde cuentakilómetros que, como máximo, pueden alcanzar esa cifra.

Referente a los litros, el depósito de una guagua tiene una capacidad máxima de mil quinientos litros, por lo que los valores para este campo no pueden exceder esa cifra.

Para llevar a cabo la validación de los datos, utilizaremos la librería Pandera de Python. Con ella, definiremos un esquema de validación para cada columna en la que se verifiquen todas las condiciones que se han mencionado.

Empezaremos uniendo ambas tablas de repostajes ordenando los datos por fechas, para que así, podamos aplicar el esquema de validación a todos los datos independientemente de si son internos o externos. Lo haremos con el Código 4.8.

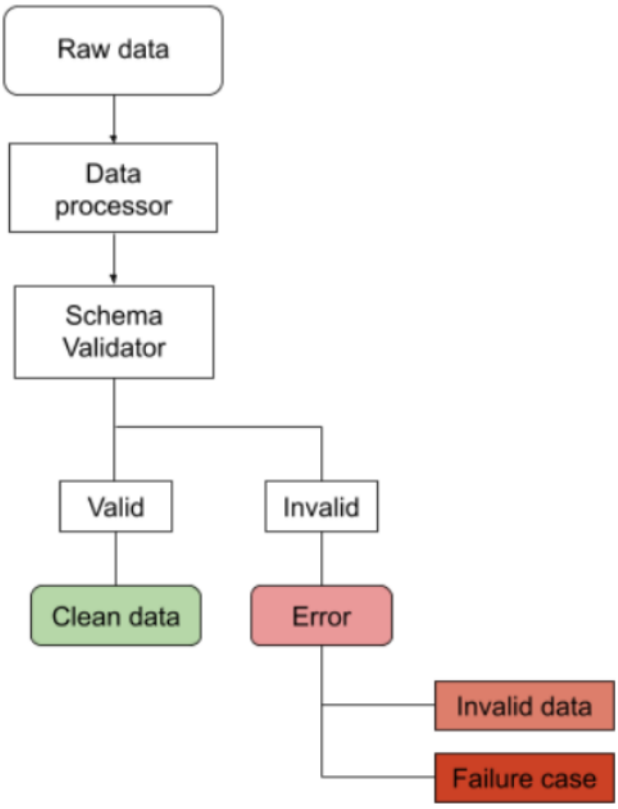


Figura 4.2. Esquema validación de datos
Fuente: https://analyticsindiamag.com/a-hands-on-guide-to-pandera-a-statistical-dataframe-testing-toolkit/

```
1 repost = pd.concat([repostex, repostin])
2 repost = repost.sort_values(by=['Fecha','Hora'])
3 repost = pd.merge(repost,matriculas, how='outer', on=['Matricula'])
4 repost['Indices'] = range(0, len(repost))
5 repost_group = repost.groupby(['Matricula','Indices']).sum(numeric_only
    =False)
6 repost_group['Diferencia'] = repost_group.groupby('Matricula')['
    Kilometros'].diff()
7 repost_group['Diferencia_inv'] = repost_group.groupby('Matricula')['
    Kilometros'].diff(-1)
```

Código 4.8. Unión de repostajes

Haciendo uso de la función `concat`, unimos ambas tablas de repostajes y ordenamos los datos por fecha y hora en las líneas uno y dos. Luego, en la siguiente línea, le unimos la tabla `matriculas` de tal forma que a cada uno de los vehículos se le asocie su empresa correspondiente. Creamos una columna llamada `Indices` que asociará un número del cero al número total de filas menos uno, de tal forma que, al agrupar nuestros datos por matrícula en la línea cinco, no

se sumen los valores numéricos de nuestra tabla sino que sigan siendo repostajes independientes. Por último, para comprobar que los kilómetros son crecientes para cada vehículo a medida que pasa el tiempo, crearemos dos nuevas columnas más: `Diferencia`, en la que se almacenarán las diferencias de cada kilómetro con el anterior para cada vehículo y `Diferencia_inv`, que contendrá las diferencias de cada kilómetro con el siguiente para cada vehículo. Con ellas, podremos ver cuántos kilómetros ha realizado un vehículo de un repostaje a otro y, además, nos servirán para ir comprobando que el kilómetro actual siempre es mayor que el anterior y menor que el siguiente.

Empezaremos la validación con en el Código 4.9, importando las librerías necesarias para llevarla a cabo.

```
1 import pandera as pa
2 from pandera import Column, Check
3 from pandera.errors import SchemaErrors
```

Código 4.9. Importación de librerías

Luego, utilizaremos el Código 4.10 con el que, a través de la librería `pandera`, crearemos el esquema de validación para nuestros datos.

```
def validar_datos_repostajes(df,m):
    matriculas_disp = m['Matricula'].str.strip().unique()
    max_kms = 1000000
    max_litros = 1700

    schema = pa.DataFrameSchema({
        'Matricula': Column(pa.String, Check.isin(matriculas_disp,
            error='La matricula no es correcta'),
                            nullable=False),
        'Fecha': Column(nullable=False),
        'Hora': Column(nullable=False),
        'Kilometros': Column(pa.Int, nullable=False, checks=[
            Check(lambda x: x > 0,element_wise=True,
                error='Km debe ser mayor que 0'),
            Check(lambda x: x < max_kms , element_wise=True,
                error = 'Km excede el limite')
        ]),
        'Litros_ext': Column(pa.Float, nullable=True, checks=[
            Check(lambda x: x >= 0, element_wise=True,
                error='Litro externo debe ser mayor que 0'),
            Check(lambda x: x < max_litros , element_wise=True,
                error='Litro externo excede el limite')
        ]),
        'Litros_int': Column(pa.Float, nullable=True, checks=[
            Check(lambda x: x >= 0, element_wise=True,
                error='Litro interno debe ser mayor que 0'),
            Check(lambda x: x < max_litros , element_wise=True,
                error='Litro excede el limite')
        ]),
        'Diferencia': Column(pa.Float, nullable=True, checks=[
            Check(lambda x: x > 0, element_wise=True,
                error='Km debe ser creciente')
        ]),
        'Diferencia_inv': Column(pa.Float, nullable=True, checks=[
            Check(lambda x: x < 0, element_wise=True,
                error='Revisar km, posible error')
        ])
    })

    try:
        validation_result = schema.validate(df, lazy=True)
    except SchemaErrors as err:
        errores = err.failure_cases
        new_df = err.data
        error_index = errores['index'].str.split(',',expand=True)[1].
            str.replace(')','',regex=False).str.strip().fillna(-1).
            astype(float).round()
        diccionario_errores = dict(zip(error_index, errores['check']))
        new_df['Validado'] = new_df['Indice'].map(diccionario_errores).
            fillna('Validado')
    return new_df

repost_val = validar_datos_repostajes(repost_group,matriculas)
```

Código 4.10. Validación de datos

En este código creamos una función llamada `validar_datos_repostajes` que recibirá dos dataframes, el que queremos validar y en el que se almacenan las matrículas de los vehículos disponibles. En primer lugar, definimos el valor máximo que pueden alcanzar los kilómetros, el máximo de litros y una lista que contenga las matrículas disponibles. Pasamos al esquema de validación que comprobará que se cumplan las siguientes condiciones para cada columna:

- `Matrícula`: debe ser una cadena no nula y debe estar en la lista de matrículas disponibles. Si estas condiciones no se cumplen se mostrará el mensaje de error *"La matrícula no es correcta"*.
- `Fecha` y `Hora`: no pueden ser nulas.
- `Kilómetros`: debe ser un número entero que sea mayor que cero y menor que el límite definido, en este caso, menor que un millón. Si estas condiciones no se cumplen se mostrará el mensaje de error *"Km excede el límite"* o *"Km debe ser mayor que 0' según corresponda"*.
- `Litros ext` y `Litros int`: debe ser un número decimal mayor que cero y menor que el límite definido, en este caso, menor que mil setecientos. Si estas condiciones no se cumplen se mostrará el siguiente mensaje de error *"Litro debe ser mayor que 0"* o *"Litro excede el límite"*, según corresponda.
- `Diferencia`: la diferencia entre cada kilómetro y su anterior debe ser siempre positiva para cada vehículo para asegurarnos de que son crecientes. Si esta condición no se cumple se mostrará el mensaje de error *"Km debe ser creciente"*.
- `Diferencia inv`: del mismo modo, la diferencia entre cada kilómetro y su siguiente debe ser siempre negativa. Esta condición es un extra para la comprobación de que los kilómetros sean crecientes ya que, puede haber un kilómetro erróneo que cumpla la condición anterior pero, si esta condición no se verifica, puede ayudarnos a identificarlo, pues es un indicador de que se ha producido un aumento y descenso repentino. Por esta razón, si no se cumple se mostrará el mensaje de error *"Revisar km, posible error"*.

Una vez comprobadas todas estas condiciones, si no se cumple alguna de ellas, se lanza una excepción del tipo `SchemaErrors` y se almacenan los casos erróneos en la variable `errores`. De esta variable, extraemos el índice de cada caso erróneo para poder buscarlo en `new_df`, que es una copia del dataframe inicial. Por último, se crea un diccionario que relaciona cada índice numérico con el tipo de error que ha ocurrido y así, podremos añadirlo en la columna `Validado` creada inicialmente.

En conclusión, después de ejecutar este código obtendremos una nueva tabla de datos que contendrá en la columna `Validado` su error en el caso de que lo hubiera, para que así la empresa pueda indentificarlo fácilmente y decida qué hacer con ellos. De lo contrario, si cumple todas las condiciones aparecerá "Validado".

Finalmente, nos quedaremos con los datos que hayan sido validados y desagruparemos el dataframe como se muestra en el Código 4.11.

```
repost_valid = repost_val[repost_val['Validado'] == 'Validado']
repost_des = repost_valid.reset_index()
```

Código 4.11. Recopilación de datos validados

Una vez lo hayamos hecho, crearemos una nueva base de datos en PostgreSQL llamada `bd-DW` que será nuestro DW. En ella podremos crear las nuevas tablas procesadas de repostajes, gastos de mantenimiento y facturación y luego rellenarlas con su información correspondiente, tal y como hicimos con la base `bd-ODS`.

4.5. Análisis y visualizaciones

Hay una variedad de herramientas de visualización de datos disponibles. Empezaremos haciendo algunas visualizaciones básicas con la librería `plotly` de Python y luego, gracias a `Dash`, podremos elaborar cuadros de mando interactivos en los que la empresa podrá seleccionar y filtrar únicamente los datos que desea visualizar. En primer lugar, haciendo uso del Código 4.12, haremos las agrupaciones de datos necesarias para preparar la información que queremos mostrar.

```
columnas_union = ['Mes','Matricula','Empresa','Facturacion',
        'GastosMant','Importe_ext','Importe_int', 'Litros_ext',
        'Litros_int','Diferencia']

union1 = pd.concat([factu,gastos])
        .groupby(['Mes','Matricula','Empresa']).sum().reset_index()

union2 = pd.concat([union1,repost_des])
        .groupby(['Mes','Matricula','Empresa']).sum().reset_index()
union2 = union2[columnas_union]
union2 = union2.groupby(['Mes','Matricula','Empresa']).
        sum(numeric_only
        =False)
union2.rename(columns={'Diferencia':'Kilometros'}, inplace=True)
union2['GastosTotal'] = union2['GastosMant'] + union2['Importe_ext'] +
                    union2['Importe_int']
union2['LitrosTotal'] = union2['Litros_ext'] +
                     union2['Litros_int']
union2['MargenBruto'] = union2['Facturacion'] - union2['GastosTotal']
union2 = union2.reset_index()
union2['Mes'] = pd.Categorical(union2['Mes'], categories=mes,
                ordered=True)
```

Código 4.12. Unión para visualizaciones

Primero, concatenamos las tablas de facturación y gastos y luego, a esa tabla le concatenamos los datos de repostajes validados. Después de seleccionar únicamente las columnas con las que vamos a trabajar, hacemos una agrupación por mes, matrícula y empresa. Por último, hacemos algunas operaciones entre columnas para obtener otros indicadores que nos interesa representar como: los gastos totales (suma de los gastos por repostajes y gastos de mantenimiento), los litros totales (suma de todos los litros repostados) o el margen bruto (diferencia entre la facturación de cada vehículo y sus correspondientes gastos totales). Una vez hemos calculado los indicadores que queremos representar, empezaremos con las primeras visualizaciones.

Lo haremos con el Código 4.13.

```
import plotly
import plotly.express as px

mes_sel = 'Enero'
empresa_sel = 'nombre de la empresa'
num_sel = 10

df_gastos = union2[(union3['Empresa'] == empresa_sel)
            & (union3['Meses'] == mes_sel)].
            sort_values(by='GastosTotal', ascending=False)
top_gastos = df_gastos.head(num_sel)
fig = px.bar(top_gastos, x='Matriculas', y='GastosTotal',
    labels={'Matriculas':'Matrículas', 'GastosTotal':'Gastos'})
fig.update_layout(title='Top ', str(num_sel),
                  'vehículos con más gastos')
fig.show()

df_fact = union2[(union2['Empresa'] == empresa_sel)
        & (union2['Meses'] == mes_sel)].
        sort_values(by='Facturacion', ascending=False)
top_fact = df_fact.head(num_sel)
fig = px.bar(top_fact, x='Matriculas', y='Facturacion',
     labels={'Matriculas': 'Matrículas', 'Facturacion': 'Facturacion'})
fig.update_layout(title='Top ',str(num_sel),
                  ' vehículos con más facturación')
fig.show()

df_mar = union2[(union2['Empresa'] == empresa_sel)
        & (union2['Meses'] == mes_sel)].
        sort_values(by='MargenBruto', ascending=False)
top_mar = df_mar.head(num_sel)
fig = px.bar(top_mar, x='Matriculas', y='MargenBruto',
     labels={'Matriculas':'Matrículas','MargenBruto': 'Margen bruto'})
fig.update_layout(title='Top ',str(num_sel),
                  ' vehículos con más margen bruto')
fig.show()
```

Código 4.13. Visualizaciones

Después de importar las librerías necesarias en las líneas 1 y 2, creamos tres nuevas variables en las que almacenaremos el mes y la empresa de la que queremos visualizar los datos así como el número de vehículos que queremos mostrar. Creamos para cada visualización una copia del dataframe que contendrá únicamente los datos del mes y empresa elegidos que recibirán el nombre de `df_gastos`, `df_fact` o `df_gan` según corresponda. Luego, nos quedamos únicamente con el número de valores que hayamos elegido y, finalmente, los representamos. El resultado se muestra en la Figura 4.3. Debido a la confidencialidad de los datos, las matrículas de los vehículos aparecerán difusas en todas las visualizaciones.

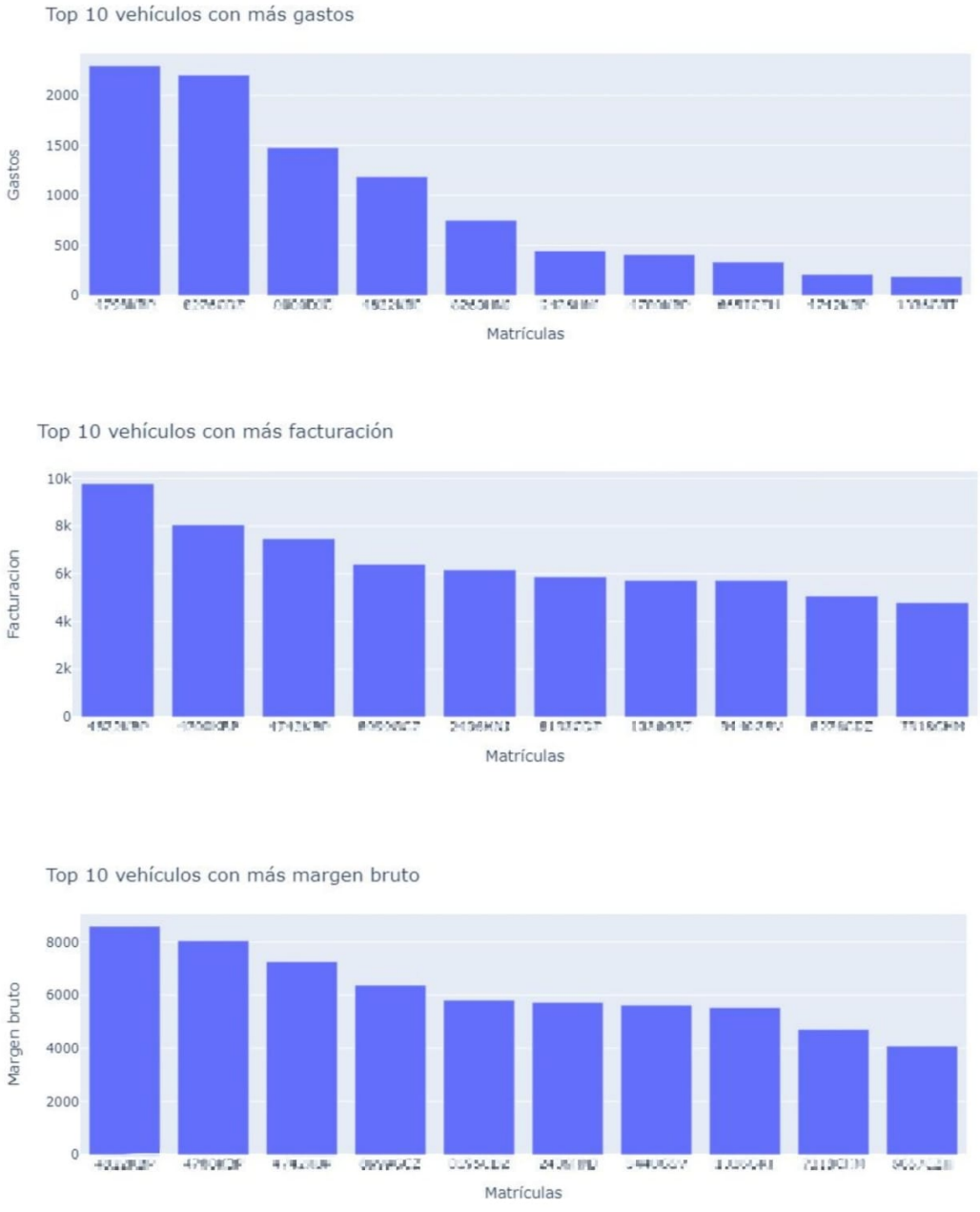


Figura 4.3. Gráficos Plotly

El siguiente paso, es la creación de un cuadro de mando con el que podamos interactuar para seleccionar la información que deseemos de esas visualizaciones. Lo crearemos con Dash, usando el Código 4.14.

```
import dash
from dash import html, dcc
from dash import Input, Output

empresas = union2['Empresa'].unique()

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("Cuadro de mando"),
    html.Label("Seleccione la empresa:"),
    dcc.Dropdown(
        id='dropdown-empresa',
        options=[{'label': empresa, 'value': empresa}
        for empresa in empresas],
        value=empresas[0]
    ),
    html.Br(),
    html.Label("Seleccione el mes:"),
    dcc.Dropdown(
        id='dropdown-mes',
        options=[{'label': mes, 'value': mes}
        for mes in union2['Meses'].unique()],
        value=union2['Meses'].unique()[0]
    ),
    html.Br(),
    html.Label("Seleccione el número de vehículos a visualizar:"),
    dcc.Slider(
        id='slider-valor',
        min=5,
        max=30,
        step=5,
        value=5,
        marks={
            5: '5',
            10: '10',
            15: '15',
            20: '20',
            25: '25',
            30: '30'
        }
    ),
    html.Div([
        dcc.Graph(id='grafico-gastos'),
        dcc.Graph(id='grafico-fact'),
        dcc.Graph(id='grafico-margen')
    ])
])
```

```
56
57 @app.callback([
58     Output(component_id='grafico-gastos', component_property='figure'),
59     Output(component_id='grafico-fact', component_property='figure'),
60     Output(component_id='grafico-margen', component_property='figure')
       ],
61     [Input(component_id='dropdown-empresa',
62     component_property='value'),
63     Input(component_id='dropdown-mes', component_property='value'),
64     Input(component_id='slider-valor', component_property='value')])
65
66 def update_graficos(empresa, mes, valor):
67     df_gas = union2.loc[union2['Empresa'] == empresa].
68             sort_values(by='GastosTotal', ascending=False)
69     df_gas = df_gas.loc[df_gas['Meses'] == mes]
70     top_gas = df_gas.head(valor)
71     fig_gastos = px.bar(top_gas, x='Matriculas', y='GastosTotal',
72                 labels={'Matriculas': 'Matrículas',
73                         'GastosTotal': 'Gastos'})
74     fig_gastos.update_layout(title='Top ' + str(valor) +
75                              ' vehículos con más gastos')
76
77     df_fact = union2.loc[union2['Empresa'] == empresa].
78             sort_values(by='Facturacion', ascending=False)
79     df_fact = df_fact.loc[df_fact['Meses'] == mes]
80     top_fact = df_fact.head(valor)
81     fig_fact = px.bar(top_fact, x='Matriculas', y='Facturacion',
82                 labels={'Matriculas': 'Matrículas',
83                         'Facturacion': 'Facturación'})
84     fig_fact.update_layout(title='Top ' + str(valor) +
85                         ' vehículos con más facturación')
86
87     df_mar = union2.loc[union2['Empresa'] == empresa].
88             sort_values(by='MargenBruto', ascending=False)
89     df_mar = df_mar.loc[df_mar['Meses'] == mes]
90     top_mar = df_mar.head(valor)
91     fig_mar = px.bar(top_mar, x='Matriculas', y='MargenBruto',
92             labels={'Matriculas': 'Matrículas',
93                     'MargenBruto': 'Margen bruto'})
94     fig_mar.update_layout(title='Top ' + str(valor) +
95                         ' vehículos con más margen bruto')
96
97     return fig_gastos, fig_fact, fig_mar
98
99
100 if __name__ == '__main__':
101     app.run_server(port=12000, debug=True, use_reloader=False)
```

Código 4.14. Cuadro de mando: Indicadores por vehículo y mes

El código comienza importando las bibliotecas necesarias y define una variable llamada `empresas`, que almacena los valores únicos de la columna `Empresa`. Luego, en la línea 7, se crea una instancia de la aplicación Dash, que es un marco de trabajo de Python para crear aplicaciones web interactivas. A continuación,

se define la interfaz de usuario de la aplicación en la línea 9. La interfaz de usuario está escrita en HTML y define la estructura y los elementos de la página, como encabezados, etiquetas, menús desplegables y deslizadores.

Luego, se define la función `update_graficos`, que se ejecutará cuando ocurran cambios en la interfaz de usuario. Esta función se utiliza como un `callback`, es decir, una función que se ejecuta en respuesta a un acto específico en la aplicación, como los cambios en los menús desplegables y el deslizador. Dentro de la función `update_graficos`, filtramos nuestros datos únicamente con los datos seleccionados en los menús desplegables y el deslizador. El resultado son tres conjuntos de datos filtrados: `top_gas`, `top_fact` y `top_gan`.

Con ellos, se crean tres figuras de gráficos utilizando la librería `Plotly Express`. Cada figura representa un gráfico de barras con los datos que se mostraban en la Figura 4.3 y su diseño se irá actualizando para agregar un título basado en el valor seleccionado en el deslizador. La función `update_graficos` devuelve las tres figuras de gráficos como salidas. Se define un decorador `@app.callback`, que vincula la función `update_graficos` con las salidas y las acciones de la interfaz de usuario, que hará que se actualicen cuando ocurran cambios en las selecciones especificadas. Por último, en la línea 101, se ejecuta la aplicación en un servidor local. Antes de ejecutar el código, debemos crear conexión al puerto 12000, tal y como lo hicimos anteriormente en la Sección 2.1, cambiando en la Figura 2.2 los datos del puerto por 12000.

Finalmente, el resultado se muestra en la Figura 4.4.

Figura 4.4. Cuadro de mando: Facturación, gastos y margen bruto

Por otro lado, otro de los objetivos principales era obtener una visualización interactiva de la evolución de estos indicadores para cada vehículo por mes. Para ello, usaremos el Código 4.15.

```
matriculas = union2['Matricula'].unique()
app = dash.Dash(__name__)
app.layout = html.Div([
    html.H1('Evolución por matrícula'),
    dcc.Dropdown(
        id='matr-dropdown',
        options=[{'label': matricula, 'value': matricula}
        for matricula in matriculas],
        value= matriculas[0]
    ),
    html.Div(id='sel-matricula'),
    html.Div([
        dcc.Graph(id='grafico-fact'),
        dcc.Graph(id='grafico-gastos'),
        dcc.Graph(id='grafico-margen')
    ])
])

@app.callback([
    Output(component_id='grafico-fact', component_property='figure'),
    Output(component_id='grafico-gastos', component_property='figure'),
    Output(component_id='grafico-margen', component_property='figure'),
    Output(component_id='sel-matricula', component_property='children')
    ],
    [Input(component_id='matr-dropdown', component_property='value')])

def update_graficos(matricula):
    data = union2[union2['Matricula'] == matricula]
    empresa = data['Empresa'].iloc[0]

    data_fact = data.groupby('Mes')['Facturacion'].sum().reset_index()
    data_gast = data.groupby('Mes')['GastosTotal'].sum().reset_index()
    data_gan = data.groupby('Mes')['MargenBruto'].sum().reset_index()

    fig_fact = px.line(data_fact, x='Mes', y='Facturacion',
        title=f'Facturación por mes para la matrícula {matricula}')
    fig_gast = px.line(data_gast, x='Mes', y='GastosTotal',
        title=f'Gastos por mes para la matrícula {matricula}')
    fig_gan = px.line(data_gan, x='Mes', y='MargenBruto',
        labels={'MargenBruto':'Margen bruto'},
        title=f'Margen bruto por mes para la matrícula {matricula}')

    seleccion = html.P(f"Selección: {matricula} | Empresa: {empresa}")

    return fig_fact, fig_gast, fig_gan, seleccion

if __name__ == '__main__':
    app.run_server(port=12000,debug=True,use_reloader=False)
```

Código 4.15. Cuadro de mando: Evolución por vehículo

Es muy similar al código anterior a excepción de algunos cambios. En este cuadro de mando aparecerá al inicio un seleccionable para elegir el vehículo del que queremos visualizar los datos y, al seleccionarla, nos indicará a qué empresa pertenece. Además, los gráficos que mostraremos esta vez serán de líneas. El resultado se muestra en la Figura 4.5.

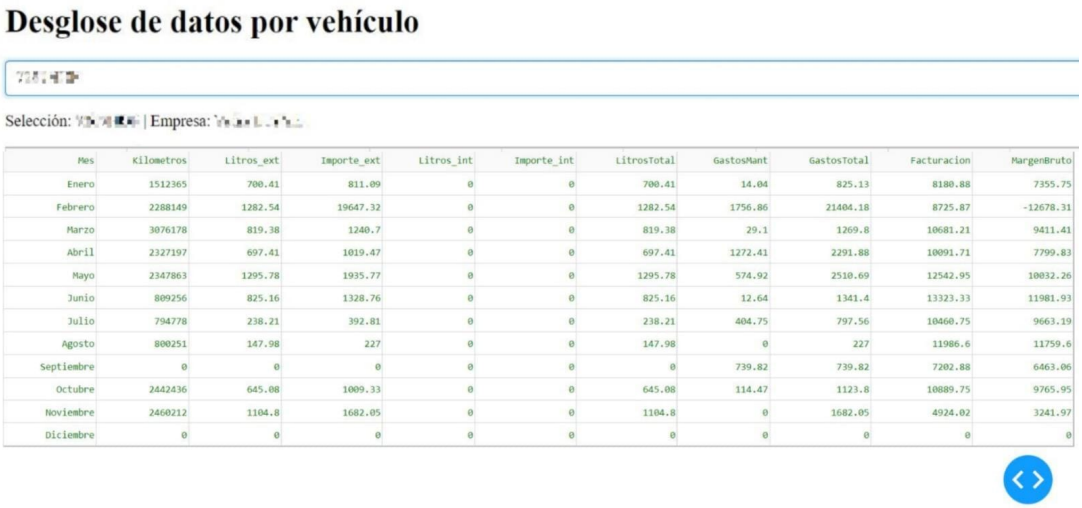
Figura 4.5. Evolución por vehículo: Facturación, gastos y margen bruto

Por último, para completar nuestros objetivos, la empresa quería visualizar los datos desglosados por cada vehículo y mes. Es decir, mostrar para el vehículo seleccionado datos como los litros repostados, diferenciando entre los repostados en sus almacenes o en estaciones de servicio externas, el número de kilómetros recorridos, el desglose de gastos totales en gastos de mantenimiento y de repostajes... Para ello, haremos otro cuadro de mando ejecutando el Código 4.16.

```
import dash_table

columnas_df = ['Mes','Kilometros','Litros ext','Importe ext','Litros
    int','Importe int','LitrosTotal','GastosMant','GastosTotal','
    Facturacion','MargenBruto']

app = dash.Dash(__name__)

df = union2[columnas_df]

table = dash_table.DataTable(
    id='tabla-data',
    columns=[{"name": i, "id": i} for i in df.columns],
    data=df.to_dict('records'),
    style_table={
        'border': '1px solid gray',
        'backgroundColor': '#3D9970',
        'color': 'green'
    }
)

app.layout = dash.html.Div(
    children=[
    dash.html.H1(children='Desglose de datos por vehículo'),
    dcc.Dropdown(
        id='matricula-dropdown',
        options=[{'label': matricula, 'value': matricula}
        for matricula in matriculas],
        value= matriculas[0]
    ),
    html.Div(id='sel-matricula'),
    table
])

@app.callback([
    Output(component_id='sel-matricula',component_property='children'),
    Output(component_id='tabla-data', component_property='data')],
    [Input(component_id='matricula-dropdown',
    component_property='value')]
)

def update_graficos(matricula):
    df = union2[union2['Matricula'] == matricula]
    filtered_data = df.to_dict('records')
    empresa = df['Empresa'].iloc[0]

    seleccion = html.P(f"Selección: {matricula} | Empresa: {empresa}")

    return seleccion,filtered_data

if __name__ == '__main__':
    app.run_server(port=12000,debug=True,use_reloader=False)
```

Código 4.16. Cuadro de mando: Desglose de datos

Después de importar el módulo necesario, definimos un dataframe con las columnas que vamos a representar únicamente y seguimos un proceso similar a los códigos anteriores. En este caso, la función `update_graficos` se encarga de filtrar la tabla de datos según la matrícula del vehículo que se haya seleccionado. Este código, tiene como resultado el cuadro de mando que se muestra en la Figura 4.6.



Mes	Kilometros	Litros_ext	Importe_ext	Litros_int	Importe_int	LitrosTotal	GastosMant	GastosTotal	Facturacion	MargenBruto
Enero	1512365	700.41	811.09	0	0	700.41	14.04	825.13	8180.88	7355.75
Febrero	2288149	1282.54	19647.32	0	0	1282.54	1756.86	21404.18	8725.87	-12678.31
Marzo	3076178	819.38	1240.7	0	0	819.38	29.1	1269.8	10681.21	9411.41
Abril	2327197	697.41	1019.47	0	0	697.41	1272.41	2291.88	10091.71	7799.83
Mayo	2347863	1295.78	1935.77	0	0	1295.78	574.92	2510.69	12542.95	10032.26
Junio	809256	825.16	1328.76	0	0	825.16	12.64	1341.4	13323.33	11981.93
Julio	794778	238.21	392.81	0	0	238.21	404.75	797.56	10460.75	9663.19
Agosto	800251	147.98	227	0	0	147.98	0	227	11986.6	11759.6
Septiembre	0	0	0	0	0	0	739.82	739.82	7202.88	6463.06
Octubre	2442436	645.08	1009.33	0	0	645.08	114.47	1123.8	10889.75	9765.95
Noviembre	2460212	1104.8	1682.05	0	0	1104.8	0	1682.05	4924.02	3241.97
Diciembre	0	0	0	0	0	0	0	0	0	0

Figura 4.6. Cuadro de mando: Desglose de datos por vehículo

Finalmente, gracias a este análisis y a los cuadros de mando resultantes, la empresa podrá obtener un seguimiento de la flota de vehículos de la empresa a lo largo del año 2022. Nos brindan información valiosa sobre cómo está funcionando y cómo pueden hacer cambios de tal forma que garanticen la mejora continua de la gestión de la flota y el crecimiento de la empresa.

5

Conclusiones

Este Trabajo de Fin de Grado ha sido un proyecto integral de análisis de datos que ha destacado por su enfoque en el uso de datos reales proporcionados por una empresa de transporte discrecional de Tenerife. La utilización de datos reales ha permitido obtener una visión más precisa y contextualizada de las operaciones, recursos y el rendimiento de la empresa. El objetivo principal ha sido realizar un análisis exhaustivo de estos datos y crear cuadros de mando interactivos con el fin de proporcionar a la empresa un punto de partida para implantar un proceso de BI.

El proyecto abarcó varias etapas. Tras un primer contacto con la empresa, se fijaron los objetivos del proyecto y se obtuvo información sobre los datos disponibles para el análisis como dónde están almacenados, qué información contienen, qué suponen para la empresa... Una vez tenía todos estos conceptos claros, el siguiente paso fue identificar qué herramientas informáticas eran necesarias para llevar a cabo cada una de las tareas del proyecto. Realicé un estudio profundo sobre las herramientas disponibles, teniendo en cuenta sus ventajas y desventajas frente a otras similares. Esto supuso un gran reto para mí, pues tuve que aprender a manejar herramientas que desconocía así como aprender nuevos lenguajes de programación.

Después de elegir las herramientas y configurar el entorno de trabajo, empezó el proceso de análisis. En las primeras visualizaciones de los datos veo que no hay demasiados campos de estudio y que nuestro análisis va a estar bastante limitado por esto. Además, logro identificar varios errores a causa del error humano al añadir registros en los archivos de origen. Estos datos erróneos podrían afectar a la confiabilidad de nuestros resultados finales, por lo que considero necesario realizar un proceso de validación de datos.

Se creó un esquema de validación con el que se comprobaba la verificación de ciertas condiciones para cada uno de los campos del estudio. Si no lo hacían, se mostraba el mensaje de error correspondiente de tal forma que la empresa pudiera acceder al origen de datos y tomar una decisión al respecto. En caso contrario, quedaban identificados como datos validados y podíamos proceder

con su análisis. De esta forma, se excluían del análisis los registros completos que tuvieran algún dato erróneo. Esto afecta de alguna forma a nuestro estudio ya que reduce la fiabilidad de los datos mostrados y nos limita a la hora de calcular otros indicadores. Por tanto, en lugar de eliminar una fila no válida, se podría eliminar únicamente el valor erróneo para luego poder realizar una imputación de valores basada en parámetros estadísticos de tal forma que no se desvíen demasiado de los valores reales y validados. Esto se deja indicado como una propuesta de mejora.

Una vez se validaron los datos, se realizó un análisis profundo de ellos utilizando diversas técnicas matemáticas y de visualización. Se diseñaron y crearon cuadros de mando interactivos que presentaban los resultados del análisis de manera clara y accesible. Estos cuadros de mando proporcionaron a la empresa una herramienta para monitorizar y evaluar continuamente su desempeño, así como para tomar decisiones en tiempo real basadas en datos confiables y actualizados.

En general, realizar este proyecto de análisis de datos reales ha supuesto un reto para mí y me ha hecho tener una mayor comprensión de este campo en el mercado laboral.

5.1. Trabajos futuros

Uno de los problemas encontrados fue la escasez de datos para ampliar el campo de estudio. Es por esto que, registrar nuevos datos como la fecha de fabricación del vehículo, el número de asientos ofertados de cada vehículo frente a los asientos que se terminan ocupando en cada viaje o información sobre las rutas que realiza cada uno, a priori, no supondría un gran esfuerzo para ella y ayudaría a enriquecer nuestros análisis y resultados. Con estos datos podríamos identificar patrones o relaciones entre el consumo medio del vehículo y su edad, calcular un porcentaje de ocupación medio por cada vehículo o emplear algoritmos de optimización para determinar las rutas más rentables para el vehículo en cada viaje. Además, se podrían utilizar sensores y sistemas de seguimiento de flotas para recopilar información en tiempo real sobre la ubicación de los vehículos o sistemas de información geográfica (GIS) para analizar y visualizar la información de tráfico y rutas.

De la misma forma, haber tenido datos de años anteriores habría sido de gran ayuda para el seguimiento de la flota, pues podríamos llevar una comparativa para ver cómo evoluciona el consumo de sus recursos a más largo plazo.

Finalmente, han de mencionarse algunas líneas de trabajo a seguir que ayudarían a la empresa. Se podrían realizar análisis más avanzados utilizando modelos de regresión, de series temporales o de aprendizaje automático con el fin de predecir necesidades de mantenimiento o fallos en el vehículo, costos operativos u ocupación, entre otras.

Bibliografía

[1] R Core Team. *Python: an interpreted, object-oriented, high-level programming language with dynamic semantics*. 2023. https://www.python.org/.

[2] Universidad de La Laguna. IaaS: Infraestructura como Servicio. [Año de consulta: 2023]. Disponible en: https://www.ull.es/servicios/stic/category/iaas/

[3] PuTTY: a free SSH and Telnet client. [Año de consulta: 2023]. Disponible en: https://www.putty.org/

[4] Jupyter: Interactive Computing. [Año de consulta: 2023]. Disponible en: https://jupyter.org/

[5] Pandas - Python Data Analysis Library. [Año de consulta: 2023]. Disponible en: https://pandas.pydata.org/

[6] Datetime: Basic date and time types.[Año de consulta: 2023]. Disponible en: https://docs.python.org/3/library/datetime.html

[7] Pandera: A light-weight, flexible and expressive pandas data validation library. [Año de consulta: 2023]. Disponible en: https://pypi.org/project/pandera/

[8] Plotly Python Open Source Graphing Library.[Año de consulta: 2023]. Disponible en: https://plotly.com/python/

[9] Psycopg2 - PostgreSQL database adapter for Python. [Año de consulta: 2023]. Disponible en: https://www.psycopg.org/

[10] Dash: Analytical web applications for Python. [Año de consulta: 2023]. Disponible en: https://plotly.com/dash/

[11] PostgreSQL: The world's most advanced open source database. [Año de consulta: 2023]. Disponible en: https://www.postgresql.org/

[12] SQL: Language to create database [Año de consulta: 2023]. Disponible en: https://www.tutorialspoint.com/sql/sql-overview.htm

[13] PgAdmin: The Open Source administration and development platform for PostgreSQL. [Año de consulta: 2023]. Disponible en: https://www.pgadmin.org/

[14] DataPrix. Introducción a SQL. [Año de consulta: 2023]. Disponible en: https://www.dataprix.com/es/book/export/html/441

[15] DataPrix. Data Warehousing y Metodología Hefesto: Proceso BI. [Año de consulta: 2023]. Disponible en: https://www.dataprix.com/es/data-warehousing-y-metodologia-hefesto/13-proceso-bi

[16] Real Python. Building Interactive Dashboards with Plotly and Dash. [Año de consulta: 2023]. Disponible en: https://realpython.com/python-dash/

[17] Shahriar Akter, Samuel Fosso Wamba, Angappa Gunasekaran, Rameshwar Dubey, Stephen J. Childe. (2016). How to improve firm performance using big data analytics capability and business strategy alignment. International Journal of Production Economics, Volumen(182). [Año de consulta: 2023]. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0925527316302110#bib66

[18] Arafat Salih Aydiner, Ekrem Tatoglu, Erkan Bayraktar, Selim Zaim, Dursun Delen. (2019). Business analytics and firm performance: The mediating role of business process performance. Journal of Business Research, Volumen(96). [Año de consulta: 2023]. Disponible en: https://www-sciencedirect-com.accedys2.bbtk.ull.es/science/article/pii/S0148296318305800

[19] Liebowitz, Jay. (2013). *Big data and business analytics*. CRC Press.

[20] Stubbs, Evan. (2011). *The value of business analytics: Identifying the path to profitability*. John Wiley & Sons.

[21] Pierson, Lillian. (2015). *Data Science*. For Dummies.

[22] Sullivan, Rob. (2012). *Introduction to data mining for the life sciences*. Springer.

Business Intelligence applied to a transport company

Tania Melián Expósito
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101223940@ull.edu.es

Abstract

The importance of data in business decision making has become increasingly apparent in recent years, specifically in transport companies. This project addresses the complete process of data analysis apply to real data provided by a transport company in Tenerife. It ranges from the identification of the necessary data, its collection and upload to databases to its processing and validation using different computer tools. Finally, dashboards have been created that will allow the company to access a global vision of the state of its fleet and its performance.
There were challenges that had to be overcome like the existence of erroneous or incomplete data and a scarcity of data available to carry out a deeper analysis. For this reason, suggestions and future lines of work are proposed, such as carrying out predictive analyzes to anticipate possible problems and needs of the fleet or the exploration of new fields in order to enrich the analysis to continue promoting the growth and optimization of the business.

1. Development environment

We will use a virtual machine of the IaaS of the University of La Laguna to which we will connect through PuTTy 1(b). The extraction, processing and analysis of the data will be carried out in Jupyter Lab 1(d), specifically in Notebooks, using the Python 1(a) language and its libraries. In addition, we will use PostgreSQL 1(c) and PgAdmin to create and manage databases.

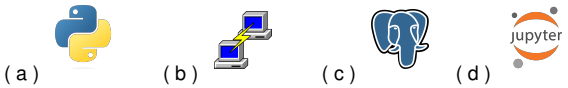


Figure 1: *Computer tools*

2. Business Intelligence

Business intelligence is the process by which companies use statistical methods and technologies to analyze data in order to gain new insights and improve strategic decision making. It is made up of the phases shown in the Figure 2.

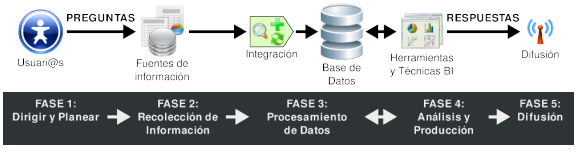


Figure 2: *BI phases*

Fuente: https://www.dataprix.com/es/book/export/html/441

3. Solution development

- *Lead and plan*: At this stage, the company tells us that it wants to view indicators such as billing, spending and gross margin for each vehicle and explains how it wants to see them.
- *Data collection*: The data is stored in six Excel documents. We access, visualize and understand them. They contain information on the external and internal refueling carried out by the vehicles and on their expenses, billing and companies to which they belong.
- *Data processing*: We create a database and all the information is stored in it. Then, it is processed and the format is changed. In this process, we find erroneous, incomplete or missing data so we have to carry out a validation process. We do it with de Python library Pandera.
- *Analysis and production*: Once the data has been validated, we move on to the representation of the correct ones. The necessary calculations are made to create the indicators that we wanted to see. Using the Python Dash library, the dashboards requested by the company are created. Are shown in the Figure 3 and Figure 4.

Figure 3: *Dashboard: Indicators*

Figure 4: *Dashboard: Evolution by vehicle*

- *Sharing*: Finally, users, using different tools, will be able to explore the data they want in a simple and intuitive way thanks to the dashboard that we have created. In this way, they will be able to carry out a detailed monitoring of their performance and evolution whenever they wish.

References

[1] Sullivan, Rob. (2012). *Introduction to data mining for the life sciences*. Springer.

[2] DataPrix. Introducción a SQL. [Consulta: 2023]. Disponible en: https://www.dataprix.com/es/book/export/html/441

[3] Liebowitz, Jay. (2013). *Big data and business analytics*. CRC Press.