MÁSTER EN BIG DATA. TECNOLOGÍA Y ANALÍTICA AVANZADA.

TRABAJO FIN DE GRADO

AUTOMATIZACIÓN DEL PARSEO DE DOCUMENTOS PDF-XFA (XML FORMS ARCHITECTURE) Y LA POSTERIOR EXPLOTACIÓN ANALÍTICA DE SU INFORMACIÓN

Autor: Ángel Catalán Criado

Director: Manuel Pérez Barajas

Madrid

Junio de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Automatización del parseo de documentos PDF-XFA (XML Forms Architecture) y la posterior explotación analítica de su información

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Ángel Catalán Criado Fecha: 02 / 06 / 2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Manuel Pérez Barajas Fecha: 06 / 06 / 2025

AUTOMATIZACIÓN DEL PARSEO DE DOCUMENTOS PDF-XFA (XML FORMS ARCHITECTURE) Y LA POSTERIOR EXPLOTACIÓN ANALÍTICA DE SU INFORMACIÓN

Autor: Catalán Criado, Ángel.
Director: Manuel Pérez Barajas.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El objetivo de este proyecto es mostrar cómo con analítica de datos y conocimientos de ETL (Extract-Transform-Load) es posible no sólo automatizar tareas (y su consecuente ahorro) sino también poder acceder a la información/datos de una manera rápida y masiva, pudiendo así analizarla y explotarla.

En este proyecto se desarrollará el proceso de parseo de archivos PDF-XFA a un formato tabular, para su posterior uso y análisis.

Palabras clave: ETL, XML, PDF, PDF-XFA, parsear, análisis, XFA

1. Introducción

En el ámbito de la aeronáutica, la recopilación y análisis de eventos operativos es una tarea crítica para garantizar la seguridad y la mejora continua de los sistemas. Uno de los principales mecanismos para reportar estos eventos es a través de los *In Service Event Report Template* (ISER), formularios estandarizados en formato PDF, concretamente basados en la arquitectura *XML Forms Architectur*e (XFA).

Tradicionalmente, el procesamiento de estos formularios ha requerido intervención manual, lo que introduce demoras, riesgos de error humano y demoras en el análisis. Este Trabajo Fin de Máster (TFM) se centra en la automatización de dicho proceso, desarrollando un pipeline que permite extraer de forma estructurada la información contenida en documentos PDF-XFA y transformarla en datos listos para su explotación analítica.

El sistema propuesto emplea tecnologías modernas de procesamiento distribuido, como Apache Spark, y almacenamiento en formato *Delta Table*, lo que permite escalar la solución y mejorar la disponibilidad de la información para su posterior análisis. Con ello, se reduce drásticamente el tiempo necesario para disponer de indicadores clave y se facilita la toma de decisiones basada en datos.

2. Definición del proyecto

Este proyecto consiste en el diseño e implementación de un sistema automatizado capaz de procesar documentos PDF-XFA (los ISER), extrayendo de ellos información semiestructurada (información en XML) procesándola y guardándola con un formato estructurado (tabular, en tablas delta), para su posterior análisis. El objetivo es reducir el esfuerzo manual, mejorar la trazabilidad de los datos y acelerar la disponibilidad de información útil para la toma de decisiones operativas.

3. Descripción del modelo/sistema/herramienta

El sistema desarrollado se estructura como un pipeline de procesamiento de datos dividido en varias etapas funcionales, automatizando la extracción y transformación de la información contenida en los ISER. Las etapas principales son:

1. **Ingestión**: los archivos PDF se recogen manualmente y se depositan en un directorio de entrada.
2. **Parseo**: mediante un script en PySpark, se localiza y extrae el contenido XML embebido en el PDF-XFA. A través de una función recursiva, se identifica la sección de interés y se transforma en una estructura de datos tabular.
3. **Transformación y almacenamiento**: los datos extraídos se normalizan y se almacenan en tablas delta, lo que permite escalabilidad y un procesamiento eficiente.
4. **Exportación**: una vez transformados, los datos se convierten a formato Parquet para facilitar su integración con herramientas de visualización y análisis.
5. **Archivado**: los documentos PDF procesados se trasladan a un repositorio histórico para mantener un control documental y evitar reprocesamientos.

Todo el proceso está orquestado mediante Apache Airflow, lo que permite planificar, monitorizar y escalar su ejecución. Este enfoque permite integrar la solución dentro de una arquitectura de datos más amplia, preparada para futuros casos de uso.

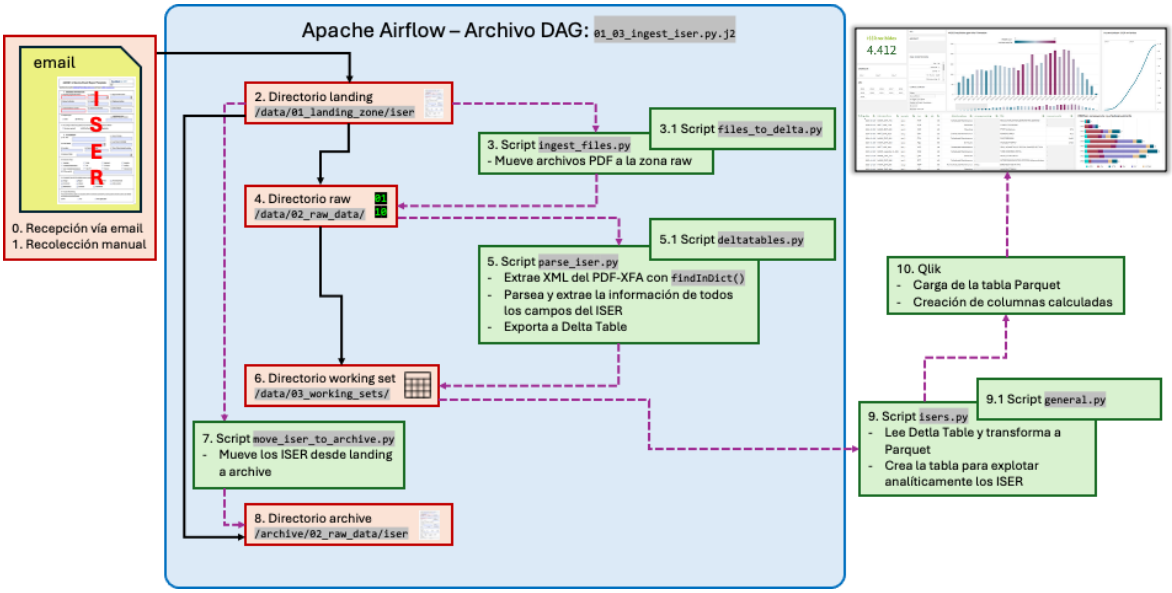

Ilustración 1. Esquema lógico del data pipeline

4. Resultados

La implementación del sistema ha permitido automatizar con éxito el proceso de extracción y guardado de los datos provenientes de los ISER. Gracias al pipeline desarrollado, los datos podrían estar disponibles para su análisis en cuestión de minutos

desde su recepción, reduciendo enormemente los tiempos respecto al método manual anterior.

Además, se ha logrado una mejora en la calidad, trazabilidad y accesibilidad de la información, facilitando la generación de indicadores clave y visualizaciones interactivas para el seguimiento de los eventos en servicio reportados.

Ilustración 2. Imagen general del dashboard desarrollado con Qlik

5. Conclusiones

Este proyecto ha demostrado que es posible automatizar el procesamiento de los ISER (documentos del tipo PDF-XFA). La solución desarrollada mejora indiscutiblemente la disponibilidad, velocidad y fiabilidad de los datos extraídos, permitiendo su explotación analítica casi en tiempo real.

Esta solución no solo reduce drásticamente la carga manual, sino que también impacta directamente en el plano económico (ahorrando los gastos del trabajo de una persona para esta tarea mecánica y sin aporte de valor añadido) y, además, la nueva solución aporta mayor valor a la toma de decisiones operativas.

6. Referencias

[1] Apache Spark. “Documentation.” https://spark.apache.org/docs/latest/

[2] Delta Lake. “What is Delta Lake?” https://delta.io

[3] Apache Airflow. “Airflow Documentation.” https://airflow.apache.org/docs/

[4] Databricks. “Delta Lake Quickstart.” https://docs.databricks.com/aws/en/delta/

[5] Github. “Updating filed in XFA PDF #2809” https://github.com/py-pdf/pypdf/discussions/2809

AUTOMATING THE PARSING OF PDF-XFA (XML FORMS ARCHITECTURE) DOCUMENTS AND THE ANALYTICAL USE OF THEIR EXTRACTED INFORMATION

Author: Catalán Criado, Ángel.
Supervisor: Manuel Pérez Barajas.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

The objective of this project is to demonstrate how, with data analytics and ETL (Extract-Transform-Load) knowledge, it is possible not only to automate tasks (and consequently save time) but also to access information/data quickly and extensively, enabling analysis and exploitation.

This project will develop the process of parsing PDF-XFA files into a tabular format for subsequent use and analysis.

Keywords: ETL, XML, PDF, PDF-XFA, parser, analysis, XFA

1. Introduction

In the field of aeronautics, the collection and analysis of operational events is a critical task to ensure safety and the continuous improvement of systems. One of the main mechanisms for reporting these events is through the In Service Event Report Template (ISER), standardized forms in PDF format, specifically based on the XML Forms Architecture (XFA).

Traditionally, processing these forms has required manual intervention, which introduces delays, the risk of human error, and limited capacity for timely analysis. This Master's Thesis focuses on automating this process by developing a pipeline that extracts the information contained in PDF-XFA documents in a structured way and transforms it into data ready for analytical exploitation.

The proposed system leverages modern distributed processing technologies, such as Apache Spark, and Delta Lake storage, enabling the solution to scale and improving data availability for use with business intelligence tools. As a result, the time required to access key indicators is drastically reduced, facilitating data-driven decision-making.

2. Project definition

This project involves the design and implementation of an automated system capable of processing PDF-XFA documents (specifically ISER forms), extracting semi-structured information (in XML format), processing it, and storing it in a structured format (tabular, using Delta tables) for subsequent analysis. The main objective is to reduce manual effort, improve data traceability, and accelerate the availability of useful information for operational decision-making.

3. Descripción del modelo/sistema/herramienta

The developed system is structured as a data processing pipeline divided into several functional stages, automating the extraction and transformation of the information contained in ISER forms.

1. **Ingestion**: PDF files are manually collected and placed into an input directory.

2. **Parsing**: Using a PySpark script, the embedded XML content within the PDF-XFA is located and extracted. A recursive function identifies the relevant section and transforms it into a tabular data structure.

3. **Transformation and Storage**: The extracted data is normalized and stored in Delta Lake tables, allowing for versioning, auditing, and efficient processing.

4. **Export**: Once transformed, the data is converted to Parquet format to facilitate integration with visualization and analysis tools.

5. **Archiving**: Processed PDF documents are moved to a historical repository to maintain document control and prevent reprocessing.

The entire process is orchestrated using Apache Airflow, which enables scheduling, monitoring, and scalable execution. This approach allows the solution to be integrated into a broader data architecture, ready for future use cases.

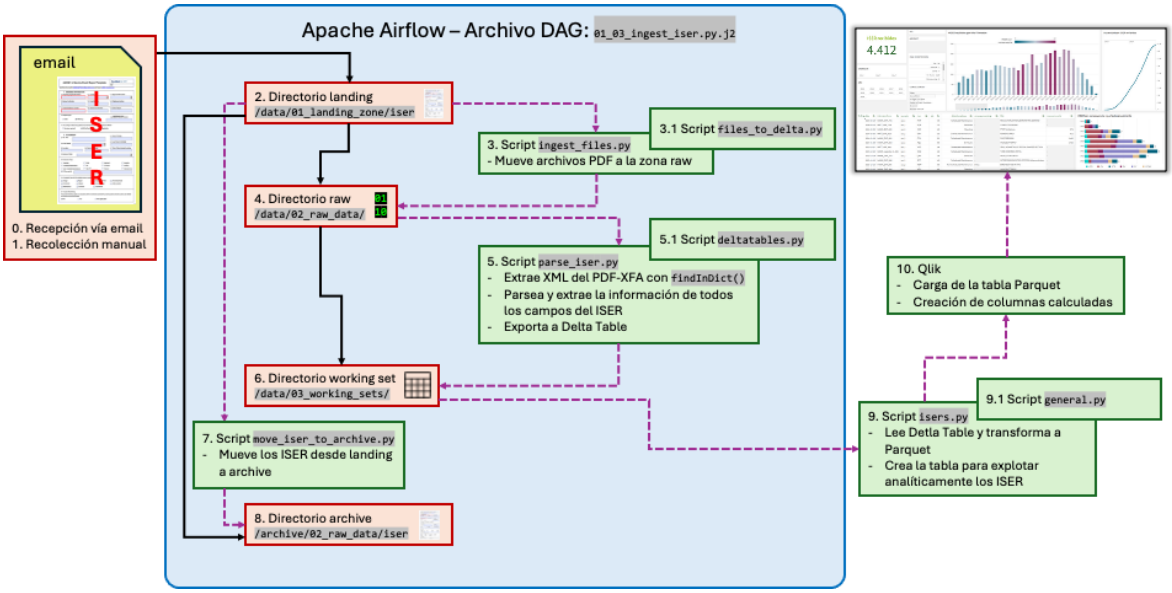

Ilustración 3. Logical overview of the data pipeline

4. Results

The implementation of the system has successfully automated the process of extracting and storing data from ISER forms. Thanks to the developed pipeline, the data can be made available for analysis within minutes of reception, significantly reducing the processing time compared to the previous manual method.

Additionally, the quality, traceability, and accessibility of the information have been improved, facilitating the generation of key performance indicators and interactive visualizations for monitoring the reported in-service events.

Ilustración 4. General view of the dashboard developed with Qlik

5. Conclusions

This project has demonstrated that it is possible to automate the processing of ISER forms (PDF-XFA documents). The developed solution clearly improves the availability, speed, and reliability of the extracted data, enabling near real-time analytical exploitation.

This solution not only drastically reduces the manual workload – directly impacting operational costs by eliminating the need for a person to perform a repetitive, non-value-adding task – but also enhances the quality of operational decision-making.

6. Referencias

[6] Apache Spark. “Documentation.” https://spark.apache.org/docs/latest/

[7] Delta Lake. “What is Delta Lake?” https://delta.io

[8] Apache Airflow. “Airflow Documentation.” https://airflow.apache.org/docs/

[9] Databricks. “Delta Lake Quickstart.” https://docs.databricks.com/aws/en/delta/

[10] Github. “Updating filed in XFA PDF #2809” https://github.com/py-pdf/pypdf/discussions/2809

Índice de la memoria

Capítulo 1. Introducción 5

1.1 Contexto y motivación del proyecto 5

1.2 Objetivo general y objetivos específicos 6

1.2.1 Objetivo general 6

1.2.2 Objetivos específicos 6

Capítulo 2. Estado del arte 8

2.1 Formato PDF-XFA: características y retos 8

2.2 Procesos ETL en Arquitecturas Modernas de Datos 8

2.2.1 Tendencias actuales en ETL 9

2.2.2 ETL distribuido con Apache Spark 9

2.2.3 Delta Lake como Motor de Persistencia 10

2.2.4 Automatización del ETL con Airflow 10

2.3 Tecnologías utilizadas 11

Capítulo 3. Descripción del problema 12

3.1 Formulario ISER (In Service Event Report Template) 13

Capítulo 4. Diseño e implementación 14

4.1 Arquitectura general del sistema 14

4.2 Fases del proceso 15

4.2.1 Ingesta de archivos PFF-XFA 15

4.2.2 Parseo del XML incrustado 15

4.2.3 Transformación y limpieza de datos 17

4.2.4 Almacenamiento en formato Delta Lake 18

4.2.5 Archivado y control de versiones 18

4.2.6 Orquestación con Apache Airflow 18

4.2.7 Control de errores y trazabilidad 19

4.3 Esquema del data pipeline 20

Capítulo 5. Explotación analítica..... 23

5.1 Preparación de los datos para el análisis 23

5.1.1 Tabla iser..... 24

5.2 Anonimizado de los datos 25

5.3 Visualización de la información..... 25

5.3.1 Vista principal del dashboard 26

5.3.2 Ejemplo de uso. Interactividad del dashboard..... 27

5.4 Valor añadido 31

Capítulo 6. Resultados..... 32

6.1 Resultados obtenidos..... 32

6.2 Comparativa con el proceso anterior..... 33

Capítulo 7. Conclusiones y trabajos futuros..... 34

Capítulo 8. Bibliografía..... 36

Capítulo 9. ANEXO – Scripts utilizados..... 37

9.1 01_03_ingest_iser.py.j2: 37

9.2 ingest_files.py 38

9.3 files_to_delta.py 41

9.4 deltatables.py..... 43

9.5 parse_iser.py..... 44

9.6 move_iser_to_archive.py 59

9.7 isers.py 61

9.8 general.py 66

Índice de figuras

Figura 6. Esquema lógico del data pipeline 22
Figura 7. Vista principal del dashboard 27
Figura 8. Paso 1 del ejercicio interactivo 27
Figura 9. Paso 2 del ejercicio interactivo 28
Figura 10. Paso 3 del ejercicio interactivo 29
Figura 11. Paso 4 del ejercicio interactivo – detalle del filtro 29
Figura 12. Paso 4 del ejercicio interactivo 29
Figura 13. Paso 5 del ejercicio interactivo 30
Figura 14. Paso 6 del ejercicio interactivo 31

Índice de tablas

Tabla 1. Información de la tabla iser 25
Tabla 2. Comparativa de procesos 33

Capítulo 1. INTRODUCCIÓN

1.1 CONTEXTO Y MOTIVACIÓN DEL PROYECTO

El trabajo desarrollado en este Trabajo Fin de Máster (TFM), nace de una necesidad dentro de un entorno laboral.

Contexto:

> En un área de la empresa y de una manera regular (casi diariamente), se reciben unos correos electrónicos con un determinado fichero adjunto, llamado ISER (el acrónimo en inglés de *In Service Event Report Template*). El ISER es un documento del tipo PDF con arquitectura XFA (*XML Forms Architecture*).
>
> Para poder utilizar la información recibida en esos PDF, se realiza manualmente una tarea, consistente en abrir cada uno de los ISER recibidos y traspasar la información de todos los campos a un fichero Excel. En ese fichero Excel, cada columna representa un campo del ISER y cada línea un documento ISER.
>
> Sin embargo y debido a obligaciones contractuales con cliente, se trata de un ejercicio vital y necesario que no puede cesar.
>
> Por otro lado, y teniendo en cuenta el estado del arte actual en cuanto a automatización de procesos y ETL (el acrónimo en inglés de *Extract Transform Load*), esta tarea, tal y como es, resulta del todo ineficiente, obsoleta, tediosa, lenta y con un nulo aporte de valor añadido. Además, al tener un factor humano, este proceso es propenso a errores en el momento del traspaso de la información, al copiar y pegar los valores del ISER al Excel.

Motivación:

Crear un proceso con el que obtener toda la información provista en el ISER, de una manera automática, rápida, robusta y eficaz.

Se trabaja entonces en el desarrollo de un proceso con el que poder extraer, transformar y guardar toda la información suministrada en los ISER de una manera automática, prescindiendo del factor humano y por tanto, reduciendo los gastos asociados a esta tarea al mínimo posible. Además, y más importante aún, la persona encargada de realizar esta tarea se puede destinar a otras tareas de mayor valor añadido.

1.2 OBJETIVO GENERAL Y OBJETIVOS ESPECÍFICOS

1.2.1 OBJETIVO GENERAL

El objetivo general de este TFM es el diseño, desarrollo e implementación de un proceso ETL completo que permita automatizar la extracción y transformación de datos desde documentos PDF-XFA hasta su almacenamiento en tablas delta para su posterior análisis y explotación.

1.2.2 OBJETIVOS ESPECÍFICOS

Los objetivos específicos son:

- Demostrar la viabilidad técnica de poder extraer información de archivos del tipo PDF-XFA.

- Desarrollar un proceso/script capaz de extraer automáticamente toda la información proveniente de esos PDF-XFA.
- Transformar toda la información extraída a formato tabular. Es decir, pasar de un dato no estructurado (o semiestructurado si tenemos en cuenta el XML) a un dato estructurado.
- Guardar los datos en formato de tabla delta y parquet.
- Sustituir el anticuado proceso manual por el nuevo flujo automático. Reduciendo tiempos de procesamiento y eliminando errores humanos.

Capítulo 2. ESTADO DEL ARTE

2.1 FORMATO PDF-XFA: CARACTERÍSTICAS Y RETOS

El formato PDF-XFA es una extensión del formato PDF estándar, principalmente diseñado para representar formularios interactivos. A diferencia de un PDF tradicional, los PDF-XFA encapsulan la información en un archivo XML incrustado en su interior. Este XML contine tanto la estructura del formulario (estructura del XML) como los valores introducidos por el usuario (los atributos y elementos del XML).

El principal reto que se plantea es que la mayoría de herramientas estándar de manipulación de archivos PDF (por ejemplo, `PyPDF2`) no consiguen acceder ni interpretar el contenido XML embebido en los PDF-XFA[1]. Por lo tanto, para extraer y convertir a un formato tabular dicho XML, es necesario un enfoque específico, que es el que se desarrolla en este TFM.

2.2 PROCESOS ETL EN ARQUITECTURAS MODERNAS DE DATOS

Los procesos ETL (Extract, Transform, Load) han evolucionado significativamente en las últimas décadas, adaptándose a: cambios en las arquitecturas de datos; necesidades de procesamiento distribuido y la analítica en tiempo (casi) real.

[1] Que este ejercicio no sea trivial, ha sido la causa principal de que esta tarea no se hubiera automatizado con anterioridad, ya que requiere de conocimientos de programación avanzados, más allá de *un simple tutorial en Google*.

Tradicionalmente, el objetivo de un proceso ETL ha sido extraer datos desde sistemas de origen (heterogéneos), transformarlos según necesidades, y cargarlos en un sistema de almacenamiento centralizado como un data warehouse o una RDBMS[2]. Sin embargo, la irrupción del Big Data y las arquitecturas basadas en la nube han provocado nuevos enfoques y herramientas que redefinen el papel del ETL en los sistemas analíticos.

2.2.1 TENDENCIAS ACTUALES EN ETL

Uno de los principales cambios en el paradigma "ETL" ha sido su desplazamiento hacia arquitecturas "ELT" (Extract, Load, Transform), donde los datos se cargan en bruto y se transforman posteriormente utilizando la capacidad computacional del motor de análisis, especialmente en entornos cloud. No obstante, en entornos donde se requiere un mayor control sobre la calidad, el enfoque ETL tradicional sigue siendo dominante.

Asimismo, el uso de frameworks como Apache Airflow ha permitido orquestar flujos de trabajo complejos de una manera sencilla y repetible. Airflow facilita la trazabilidad y reintentos automáticos, lo que es especialmente útil en contextos industriales donde se procesan volúmenes elevados de datos y donde la resiliencia operativa es crítica.

2.2.2 ETL DISTRIBUIDO CON APACHE SPARK

Apache Spark se ha consolidado como uno de los motores de procesamiento distribuido más utilizados en ETL moderno, debido a su capacidad de escalar horizontalmente y de manejar grandes volúmenes de datos en memoria. A través de su interfaz, PySpark[3], es posible diseñar transformaciones complejas que incluyen parsing, limpieza y enriquecimiento de

[2] Relational Data Base Management System
[3] PySpark nace de la combinación de Python y Spark, Es uno de los lenguajes de programación dominantes en el momento actual.

datos, reduciendo significativamente los tiempos de ejecución respecto a tecnologías tradicionales.

2.2.3 DELTA LAKE COMO MOTOR DE PERSISTENCIA

Delta Lake representa otra innovación clave en la evolución del ETL moderno. Se trata de un framework de almacenamiento transaccional sobre Apache Parquet, que permite la escritura concurrente, control de versiones, manejo de esquemas y operaciones ACID sobre datos almacenados en sistemas distribuidos. Esta tecnología permite simplificar el pipeline de datos, eliminando pasos intermedios y favoreciendo la gobernanza.

En la solución desarrollada en este trabajo, los datos extraídos de los PDF se almacenan en tablas Delta.

2.2.4 AUTOMATIZACIÓN DEL ETL CON AIRFLOW

La automatización del flujo ETL se realiza mediante DAGs (Directed Acyclic Graphs) en Apache Airflow. En el sistema desarrollado, se ha diseñado un DAG que orquesta la ejecución secuencial de tres tareas principales: ingestión de los archivos PDF, parsing de su contenido y archivo de los documentos procesados. Cada tarea se ejecuta mediante operadores de Spark (`SparkSubmitOperator`), lo que permite paralelizar la ejecución cuando sea necesario y escalar el procesamiento con control.

Este enfoque facilita la mantenibilidad del data pipeline, así como su integración futura en entornos más complejos donde coexistan múltiples fuentes de datos y diferentes ventanas temporales de procesamiento.

2.3 TECNOLOGÍAS UTILIZADAS

- Python/PySpark: lenguaje principal para el desarrollo de los scripts.
- Apache Airflow: herramienta de orquestación.
- Delta Lake: Framework de almacenamiento.
- Librerías específicas: `xml.etree.ElementTree` por ejemplo, para el parseo de XML.

Capítulo 3. DESCRIPCIÓN DEL PROBLEMA

En el entorno operativo de muchas organizaciones, la gestión de formularios digitales es un proceso clave para la recopilación de datos estructurados. Sin embargo, cuando dichos formularios están encapsulados en formatos como PDF-XFA, acceder y reutilizar su contenido se convierte en un desafío técnico y operativo.

En el caso que nos ocupa, los formularios ISER, recibidos en formato PDF-XFA, contienen datos relevantes en un XML embebido dentro del propio archivo. La extracción de dicha información no es directa ni accesible mediante herramientas convencionales, lo que obliga a recurrir a métodos más avanzados para poder acceder a los datos.

El procedimiento habitual consistía en abrir individualmente cada PDF y, campo a campo, transferirlos manualmente, copiándolos y pegándolos a una hoja de Excel. Este enfoque, aunque funcional en contextos de bajo volumen (no es el caso), presenta múltiples desventajas:

- Bajo rendimiento: el proceso manual limita el volumen de documentos que pueden procesarse en un tiempo razonable.
- Riesgo de error humano: la operación de copiar y pegar datos es susceptible de introducir errores.
- Falta de trazabilidad: no se dispone de un histórico estructurado ni de un control formal sobre los datos procesados.
- Desaprovechamiento del capital humano: se destina tiempo de personal cualificado a tareas mecánicas de escaso valor añadido.

Este contexto plantea la necesidad de diseñar un sistema automatizado, reproducible y escalable que permita procesar grandes volúmenes de formularios PDF-XFA, accediendo a través de tecnologías de procesamiento de datos como Python a los datos contenidos en el XML, y almacenarlos en una estructura que habilite su posterior análisis y explotación masiva.

3.1 FORMULARIO ISER (IN SERVICE EVENT REPORT TEMPLATE)

El documento ISER es, como ya se viene diciendo, un formulario en PDF con el cual los distintos operadores reportan eventos en servicio[4]. El formulario permite reportar información variada como pude ser:

- información general del informador;
- fecha del evento;
- si hay relación o no con eventos reportados anteriormente;
- descripciones del evento;
- posibles consecuencias y/o daños generados;
- etc.

Por motivos de confidencialidad, no se adjunta en el presente trabajo el documento ISER, ya que se trata de un formulario oficial usado en la industria.

[4] En el contexto de este TFM, los eventos en servicio representan fallos de avión que han provocado "eventos". Y por supuesto, no ocurren en todos los vuelos.

Capítulo 4. DISEÑO E IMPLEMENTACIÓN

4.1 ARQUITECTURA GENERAL DEL SISTEMA

El proceso desarrollado tiene como objetivo automatizar el procesamiento de ficheros PDF-XFA (XML Forms Architecture), facilitando su transformación a formatos estructurados adecuados para su análisis. El flujo completo se orquesta con Apache Airflow y se basa en procesamiento distribuido mediante Apache Spark, almacenamiento en formato Delta Lake y organización de los datos por directorios:

1. `/data/01_landing_zone/`
2. `/data/02_raw_data/`
3. `/data/03_working_sets/`
4. `/archive/02_raw_data/`

El sistema se compone de los siguientes bloques:

- **Ingesta manual de archivos PDF**. Se depositan en el directorio de "landing".
- **Transformaciones mediante scripts desarrollado en PySpark**, que parsean el contenido XML y lo estructuran.
- **Almacenamiento de los datos parseados** en formato Delta Lake.
- **Archivado de los ISER procesados** para mantener el control de versiones.
- **Orquestación completa** mediante un DAG de Apache Airflow.

4.2 FASES DEL PROCESO

Todos los scripts utilizados se pueden encontrar en el capítulo de *Anexos*, al final del documento.

4.2.1 INGESTA DE ARCHIVOS PFF-XFA

Los archivos PDF llegan inicialmente a través de correo electrónico. La recolección es manual: una persona descarga los archivos y los deposita en el directorio de entrada `/data/01_landing_zone/iser`. Este directorio representa la zona de "landing" del proceso, desde donde se inicia el pipeline.

La ejecución del proceso puede lanzarse manualmente o programarse mediante Airflow, utilizando el DAG `01_03_ingest_iser`.

4.2.2 PARSEO DEL XML INCRUSTADO

Una vez copiados los archivos en la zona de "landing", el script `ingest_files.py` se encarga de moverlos a la zona raw, aplicando una lógica de categorización por fecha y origen. Después, el script `parse_iser.py` procesa/parsea los archivos, extrayendo el XML interno de los formularios PDF-XFA.

Este parseo convierte el contenido semiestructurado de los XML en contenido estructurado en forma de DataFrames de Spark.

4.2.2.1 Función para extraer la información de los XML de los PDF-XFA

Para poder acceder a la información de los XML dentro de los archivos PDF-XFA, se desarrolla una función, que se encuentra dentro del script `parse_iser.py`.

Esta función se llama `findInDict()` y, debido a su importancia, se explica a continuación:

- Script:

```
def findInDict(needle, haystack):
    """
    This function look for an specific object inside of a dictionary.
    The goal for this use case id to search an specific 'tag' inside of the PDF-
XFA structure
    """
    for key in haystack:
        try:
            value = haystack[key]
        except Exception as e:
            raise e
        if key == needle:
            return value
        if isinstance(value, dict):
            x = findInDict(needle, value)
            if x is not None:
                return x
    return None
```

- Propósito: La función busca recursivamente una clave (`needle`) dentro de un diccionario potencialmente anidado (`haystack`). Es decir, puede haber (o no) diccionarios dentro de diccionarios, y se desea localizar una clave concreta sin saber su profundidad.
- Parámetros:
  - `needle`: la clave que se desea buscar.
  - `haystack`: el diccionario (potencialmente anidado) dónde se realizará la búsqueda.
- Funcionamiento:
  - Recorre todas las claves del diccionario `haystack`.
  - Intenta obtener el valor asociado a cada clave. Si hay un error (poco probable en este contexto), se propaga con `raise e`.
  - Compara si la clave actual (`key`) es igual a la buscada (`needle`). Si es así, devuelve el valor directamente.

- Si el valor asociado a la clave actual es otro diccionario, entra en él recursivamente con `findInDict()`.
- Si el resultado de esa búsqueda recursiva no es `None`, significa que se encontró el valor, y se retorna.
- Si después de recorrer todas las claves no se encuentra la deseada, devuelve `None`.

- Ejemplo práctico:

  Supongamos el siguiente diccionario llamado haystack:

```
haystack = {
    "a": {
        "b": {
            "c": "valor buscado"
        }
    }
}
needle = "c"
```

  En este ejemplo, si usamos la función `findInDict("c", haystack)`, devolvería: `"valor buscado"`.

En el contexto del caso de uso que ocupa este trabajo, los formularios PDF-XFA son PDF basados em XML y al abrirlos con bibliotecas como `PyPDF2` o similares, su contenido puede transformarse en una estructura de diccionarios anidados. Entra en juego entonces la función descrita, que se usa para buscar concretamente el elemento `/XFA`, dentro de esa jerarquía.

4.2.3 TRANSFORMACIÓN Y LIMPIEZA DE DATOS

Durante el parseo, se aplican transformaciones para convertir el XML a una estructura tabular, generando columnas normalizadas. Estas operaciones aseguran la coherencia de los datos antes de persistirlos.

El resultado se almacena como tablas delta, siendo coherente con la lógica usada para otros data pipelines también creados en este mismo entorno Big Data (pero que se quedan fuera del alcance de este trabajo).

4.2.4 ALMACENAMIENTO EN FORMATO DELTA LAKE

Los datos transformados se persisten/guardan en el directorio de `/data/03_working_sets/iser/`, utilizando delta table como formato de almacenamiento. Esto proporciona capacidades de versionado, transacciones ACID[5] y/o rendimiento escalable. La gestión de estas tablas se centraliza en los scripts `deltatables.py` y `files_to_delta.py`.

Por tanto, la tecnología o framework de almacenamiento es Delta Lake.

4.2.5 ARCHIVADO Y CONTROL DE VERSIONES

Tras completar el parseo, los archivos PDF originales se mueven a una zona de archivado en `/archive/02_raw_data/iser`. El script `move_iser_to_archive.py` realiza esta operación, renombrando los archivos con información de versión y fecha, lo que permite mantener trazabilidad completa del histórico.

4.2.6 ORQUESTACIÓN CON APACHE AIRFLOW

El DAG de Airflow (`01_03_ingest_iser.py.j2`) define el flujo de tareas del pipeline:

1. Captura de versión.
2. Ingesta inicial.
3. Parseo y transformación.

[5] ACID es acrónimo de Atomicidad (Atomicity), Consistencia (Consistency), Aislamiento (Isolation) y Durabilidad (Durability).

4. Archivado.

Cada etapa está definida mediante operadores `SparkSubmitOperator`, lo que permite ejecutar scripts Spark con parámetros específicos y configuración personalizada de recursos (memoria, núcleos, etc.).

Por otro lado, el DAG no tiene una frecuencia programada (`schedule_interval=None`), por lo que puede lanzarse manualmente o podría integrarse con *triggers* externos en el futuro. Que sea así es una decisión consensuada por las partes implicadas en este proceso.

4.2.7 CONTROL DE ERRORES Y TRAZABILIDAD

La mayoría de los scripts incluyen el uso de la librería `logging`, lo que permite registrar mensajes de depuración, información o error en consola. Esto facilita el monitoreo básico y el diagnóstico de problemas. Actualmente no hay alertas automáticas, pero el uso de Airflow y logs permite identificar fallos de ejecución de manera visual.

4.3 ESQUEMA DEL DATA PIPELINE

Se representa a continuación un esquema para poder mostrar visualmente la lógica de todo el procesado de datos.

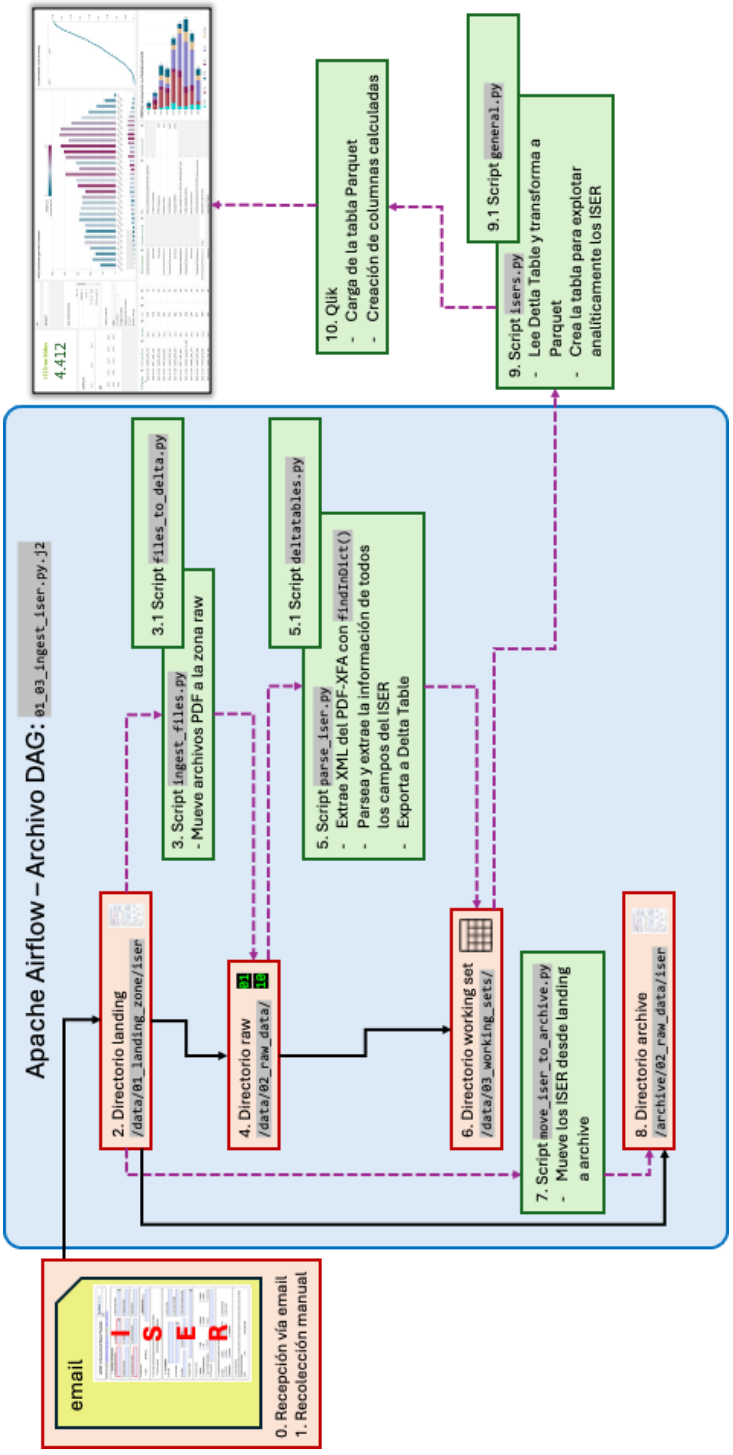
Apache Airflow – Archivo DAG: 01_03_ingest_iser.py.j2
email
ISER
0. Recepción vía email
1. Recolección manual
2. Directorio landing /data/01_landing_zone/iser
3. Script ingest_files.py - Mueve archivos PDF a la zona raw
3.1 Script files_to_delta.py
4. Directorio raw /data/02_raw_data/
5. Script parse_iser.py - Extrae XML del PDF-XFA con findInDict() - Parsea y extrae la información de todos los campos del ISER - Exporta a Delta Table
5.1 Script deltatables.py
6. Directorio working set /data/03_working_sets/
7. Script move_iser_to_archive.py - Mueve los ISER desde landing a archive
8. Directorio archive /archive/02_raw_data/iser
9. Script isers.py - Lee Detla Table y transforma a Parquet - Crea la tabla para explotar analíticamente los ISER
9.1 Script general.py
10. Qlik - Carga de la tabla Parquet - Creación de columnas calculadas
4.412

Figura 1. Esquema lógico del data pipeline

Capítulo 5. EXPLOTACIÓN ANALÍTICA

Una vez los datos han sido transformados y almacenados en formato tabla delta, pasan a ser una fuente estructurada y confiable para su análisis.

El objetivo principal de esta fase es facilitar la consulta ágil y visual de la información contenida en los ISER, ofreciendo soporte a la toma de decisiones técnicas tanto para el cliente interno (dentro de la empresa) como el cliente externo (los propios operadores).

5.1 PREPARACIÓN DE LOS DATOS PARA EL ANÁLISIS

El script `iser.py` accede a la *tabla delta* en la que se encuentran todos los datos de todos los campos del ISER[6] y realiza una serie de procesados y transformaciones creando una nueva tabla, conteniendo sólo la información requerida para resolver el caso de uso y con una estructura acorde a las expectativas del cliente. Este nuevo dataset se guarda en formato parquet, facilitando así la integración con herramientas de visualización, ya que este formato está ampliamente soportado por plataformas como Power BI, Qlik, Spotfire, Tableau, etc.

Para este trabajo y por motivos operacionales internos en la empresa, el dashboard se ha desarrollado en Qlik[7].

[6] Se recuerda que, en este punto, cada ISER representa una fila de la tabla delta y cada columna es un elemento del formulario.
[7] Qlik es una plataforma de Business Intelligence (BI) y visualización de datos que permite transformar datos crudos en análisis interactivos y dashboards visuales.

Una vez cargados los datos en Qlik, se pueden realizar transformaciones en los mismos, dentro del entorno de desarrollo de la propia herramienta. Facilitando así la personalización del resultado final, sin afectar a la tabla origen (que podría ser usada por otros usuarios para otros casos de uso diferentes).

5.1.1 TABLA ISER

Tras el procesado, la tabla llamada `iser` y usada para crear los dashboards, queda así:

Nombre	*Data type*	*Descripción*	*Extra*
internal_ref_no	text	Referenica del ISER	Primary key
title	text	Descripción del evento	
finding_date	date	Fecha en la que se produjo el evento	
operator	text	Operador de la aeronave	
msn	text	Nombre del avión	
tn1	text	Matrícula del avión	
ata	text	Completa (Parquet + BI)	
detection_phase	text	Fase en la que se detectó el evento	
consequence	text	Consecuencia producida por el evento	En formato acrónimo
consequence_long	text	Consecuencia producida por el evento	En formato extendido
msn_comments	text	Comentarios extra sobre el evento	

Tabla 1. Información de la tabla iser

5.2 ANONIMIZADO DE LOS DATOS

Los datos utilizados en este trabajo y que se muestran en la sección que viene a continuación, han sido anonimizados para garantizar la confidencialidad de la información. Esto es debido a la criticidad de los mismos, ya que los datos originales representarían eventos en servicio de aviones militares reales de distintas fuerzas aéreas (gobiernos). Por lo tanto, en el presente trabajo no se muestra ningún tipo de dato sensible o que pudiera dar lugar a identificar una fuerza aérea o avión, cumpliendo así con las políticas internas de la empresa y con los principios de protección de datos.

5.3 VISUALIZACIÓN DE LA INFORMACIÓN

El archivo Parquet con la información procesada de los ISER es la única fuente de datos usada para la construcción de los dashboards desarrollados. Una vez en la herramienta, se crea una nueva columna calculada para poder tener el valor de la columna `consequence` no sólo en formato acrónimo sino también en formato extendido.

Estas visualizaciones interactivas permiten a los usuarios consultar eventos reportados, pudiendo aplicar distintos filtros, que modifican dinámicamente la información mostrada.

5.3.1 VISTA PRINCIPAL DEL DASHBOARD

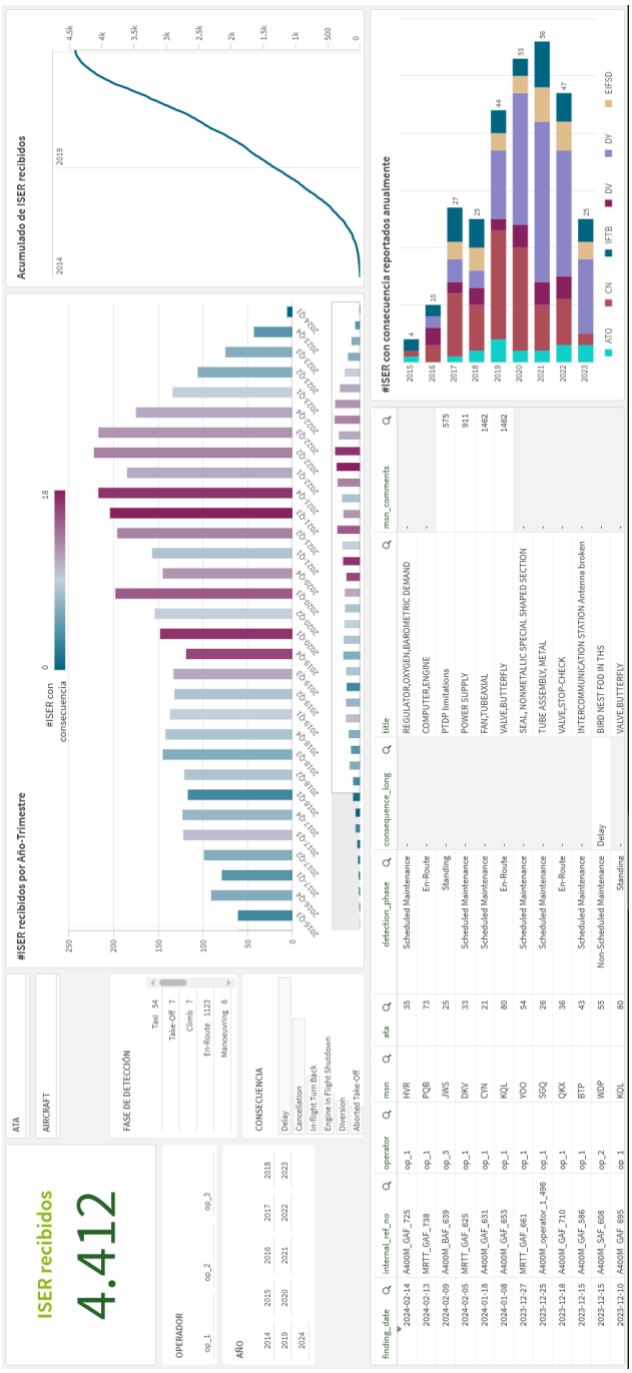

Figura 2. Vista principal del dashboard

5.3.2 EJEMPLO DE USO. INTERACTIVIDAD DEL DASHBOARD

Se va a exponer a continuación cómo evoluciona la información mostrada en el dashboard según se van aplicando filtros. Todo esto es interactivo e independiente del orden en el que los filtros se apliquen, permitiendo al usuario acceder rápidamente a una información valiosa y que sería inabarcable si la otra opción fuera ir abriendo PDF uno a uno.

1. Filtrar para que se vea información solamente del operador **op_1**:

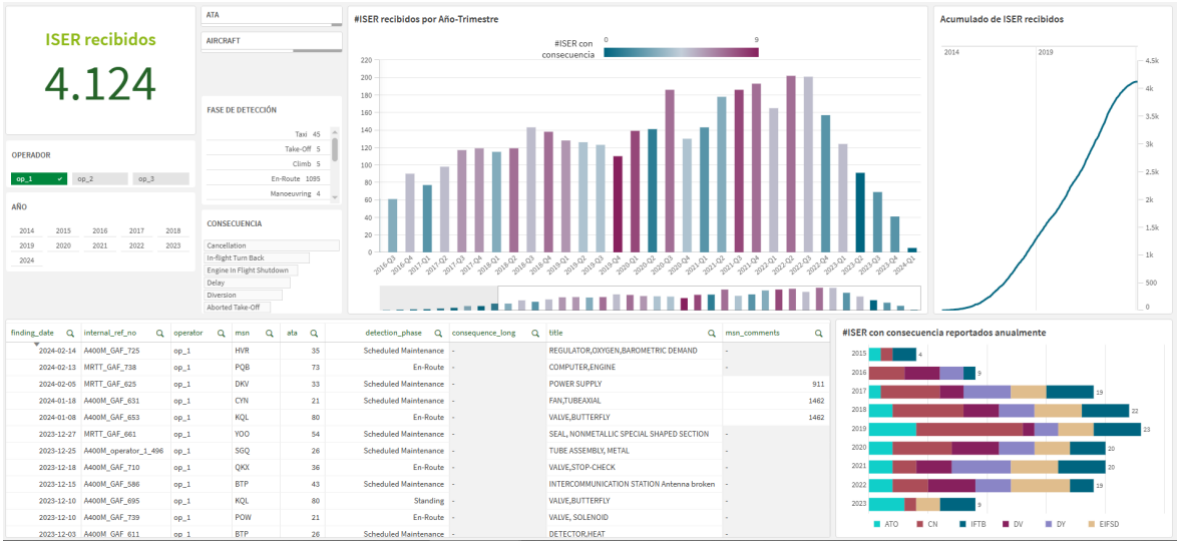

Figura 3. Paso 1 del ejercicio interactivo

2. Filtrar por ISER que hayan tenido lugar en los años **2017**, **2019**, **2022**, **2023** o **2024**:

Figura 4. Paso 2 del ejercicio interactivo

3. Filtrar por ISER que han tenido la consecuencia **Cancellation** o **Aborted Take-Off**:

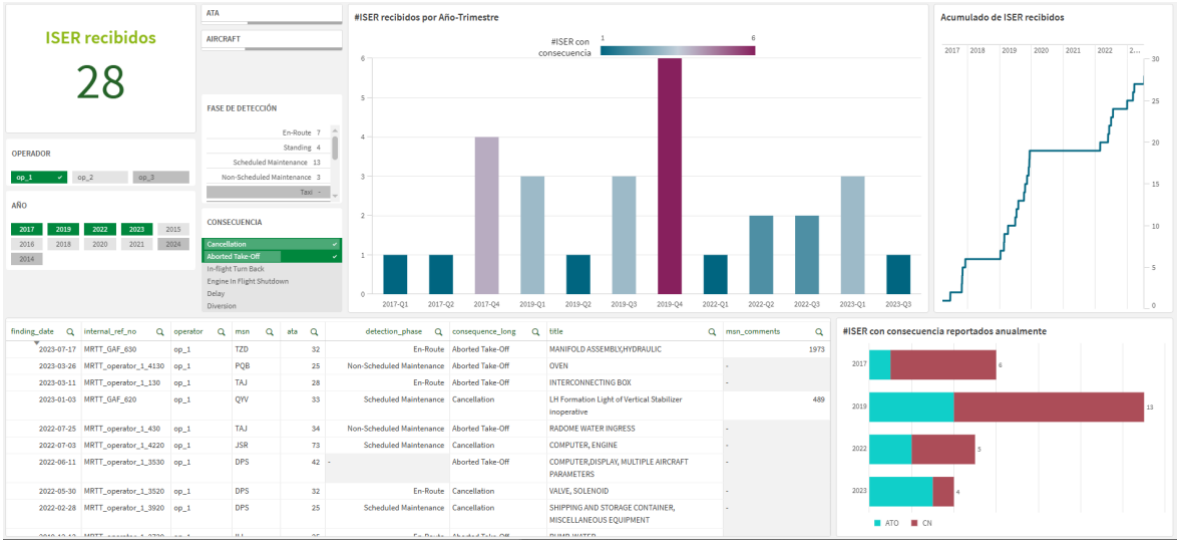

Figura 5. Paso 3 del ejercicio interactivo

4. Filtrar por el TOP-3 de aviones que hayan reportado más ISER con las condiciones actuales:

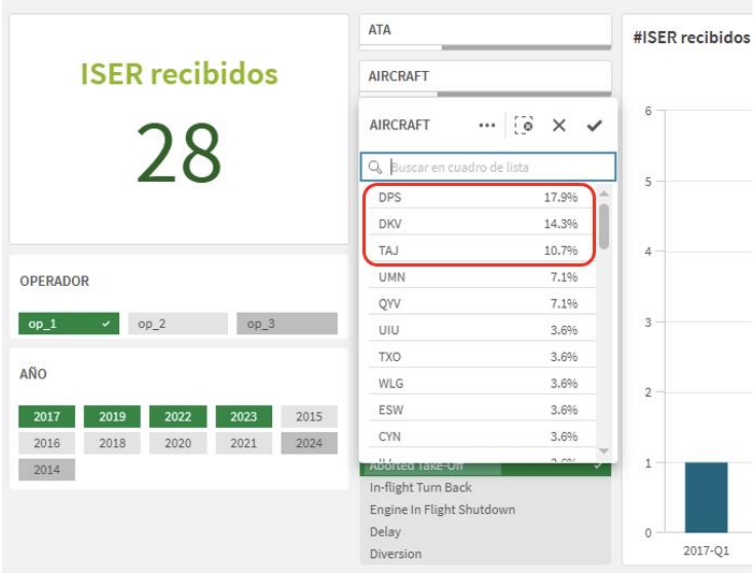

Figura 6. Paso 4 del ejercicio interactivo – detalle del filtro

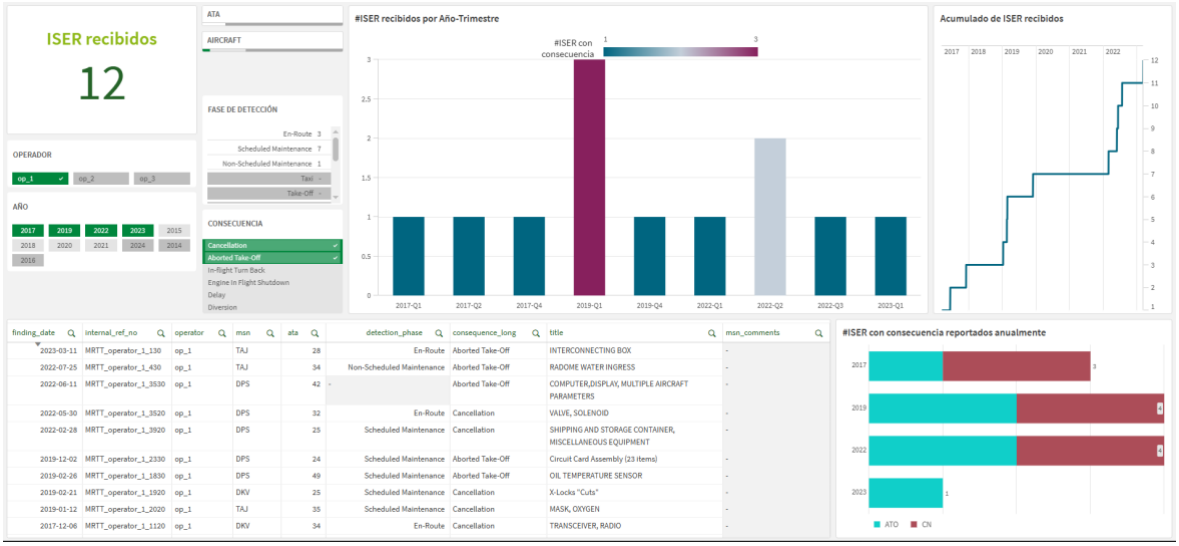

Figura 7. Paso 4 del ejercicio interactivo

5. Filtrar por ISER que se hayan detectado durante el **mantenimiento programado**:

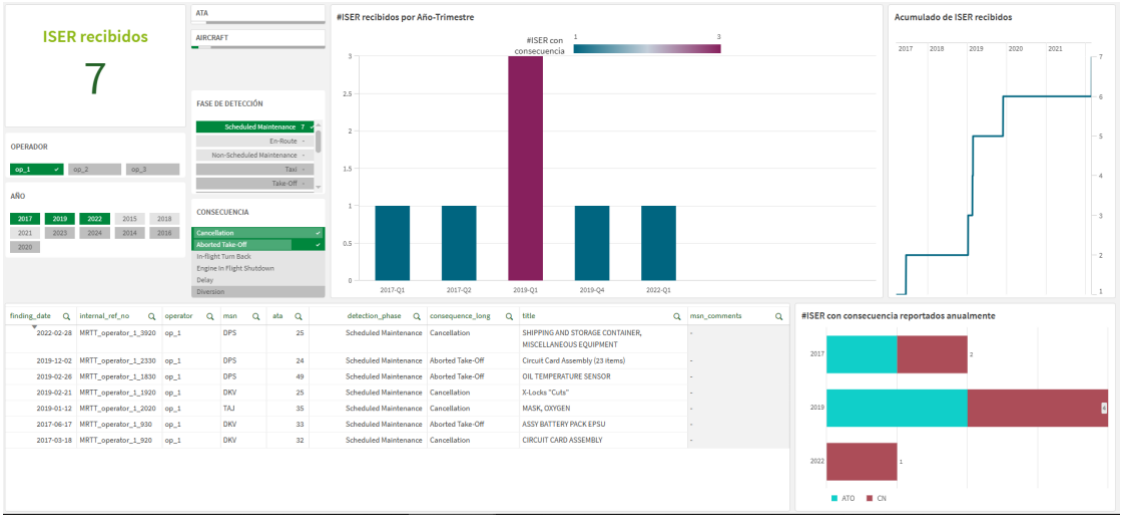

Figura 8. Paso 5 del ejercicio interactivo

6. También se pueden aplicar filtros clicando sobre elementos de las visualizaciones. En este caso, se clica en el gráfico *#ISER recibidos por Año-Tirmestre* sobre la barra

de color morado, que representa el año trimestre en el que se han recibido más ISER.

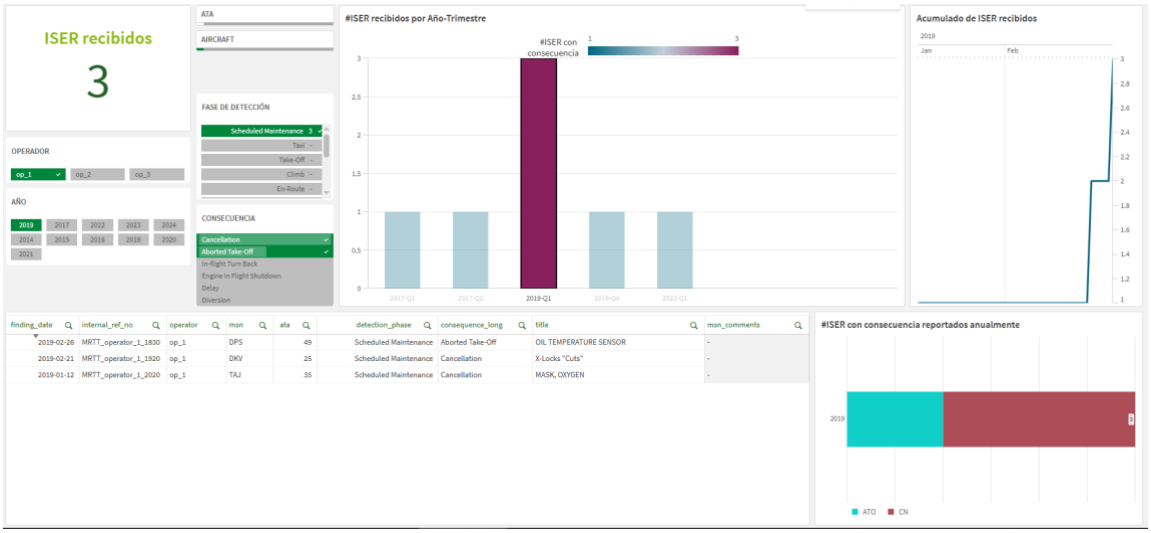

Figura 9. Paso 6 del ejercicio interactivo

5.4 VALOR AÑADIDO

Los principales usuarios de esta información son perfiles técnicos dentro de la empresa, responsables del análisis de incidencias, mantenimiento de flotas, ingenieros de fiabilidad…

La automatización del proceso ha permitido que los datos estén disponibles en el mismo día de la recepción del formulario, lo que supone una mejora significativa en los tiempos de acceso a la información frente al proceso manual anterior.

Aunque la explotación analítica se mantiene similar en cuanto a contenidos, el cambio en la velocidad, consistencia y fiabilidad del acceso a los datos ha incrementado notablemente su valor. La información ahora es más accesible, menos propensa a errores y está preparada para su consumo por parte de herramientas modernas de análisis y visualización.

Capítulo 6. RESULTADOS

6.1 RESULTADOS OBTENIDOS

La automatización del proceso de extracción y tratamiento de la información contenida en los formularios ISER (PDF-XFA) ha permitido transformar un flujo de trabajo manual, lento y propenso a errores, en un data pipeline eficiente, trazable y escalable. Entre los principales logros se incluyen:

- **Reducción del tiempo de disponibilidad de los datos**: antes, el proceso completo podía tardar varios días desde la recepción del PDF hasta su análisis; ahora, los datos están listos en cuestión de minutos tras la ejecución del pipeline.

- **Mayor fiabilidad y calidad de los datos**: al eliminar tareas manuales, se ha reducido significativamente el riesgo de errores humanos en el copiado y formateo de la información.

- **Automatización**: se han definido tareas orquestadas para ingesta, parseo y archivado, lo que permite una ejecución repetible y auditada.

- **Estandarización del almacenamiento**: el uso de tablas delta permite una gestión más eficiente y segura de los datos, facilitando la consulta posterior desde herramientas de análisis.

- **Ahorro de costes y optimización de recursos humanos**: anteriormente, esta tarea (manual) de extracción de la información de los ISER se externalizaba. Con la automatización, se ha eliminado esa necesidad, lo que supone un ahorro económico directo. Además, la persona que antes dedicaba esfuerzos a esta labor repetitiva

puede ahora centrarse en tareas de mayor valor añadido, como el análisis avanzado de los datos o la mejora de procesos.

6.2 COMPARATIVA CON EL PROCESO ANTERIOR

Aspecto	*Antiguo proceso*	*Nuevo proceso*
Tipo	Manual	Automático
Tiempo de procesamiento	1 ISER ~ 1 hora	100 ISER ~ ↓5 minutos
Intervención humana	Completa	Mínima
Trazabilidad	Limitada	Completa (logs, Airflow…)
Escalabilidad	Baja	Alta
Integración	Parcial (Excel)	Completa (Parquet + BI)

Tabla 2. Comparativa de procesos

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Como conclusiones a este Trabajo Fin de Máster, considero que uno de los principales hitos que me propuse al inicio de este – que quedase reflejada la aplicabilidad real de los conocimientos adquiridos en el *Máster de Big Data. Tecnología y Analítica avanzadas* en un entorno laboral – ha quedado sobradamente cumplido. En concreto, este trabajo refleja la capacidad de aplicar técnicas, procesos y conocimiento de las áreas de *Data Engineering* y *Data Analytics*.

El problema era claro: había un proceso altamente automatizable y con una repercusión directa tanto en el plano económico como en el de la eficiencia. La solución, también: aplicar un proceso de ETL interviniendo en todas las etapas del proceso, haciéndolo lo más eficiente posible y con el objetivo claro de terminar con un trabajo manual, que además de ser propenso a errores, consumía un recurso valiosísimo, el trabajo de una persona.

Además de lo mencionado, al proceso de ETL se añadió un trabajo de *Data Analytics*, procesando la información y creando un dashboard con la aplicación Qlik. Permitiendo así poder acceder y explotar la información de una manera rápida, precisa, interactiva y eficaz. Ahorrando tiempos de búsqueda y análisis de la información.

En cuanto a hitos principales cabe destacar:

- La correcta extracción de datos embebidos en documentos del tipo PDF-XFA.
- Procesamiento de archivos XML, transformando la información de semiestructurada a estructurada.
- Almacenamiento eficiente en formato delta, facilitando tanto el control de versiones como la explotación analítica.

- La creación de un dashboard, permitiendo obtener indicadores clave, comparativas y tendencias de forma casi inmediata.
- La automatización del proceso.

En definitiva, el trabajo realizado demuestra cómo con la combinación de técnicas avanzadas de parseo, procesamiento, almacenamiento y automatización, se puede transformar significativamente un proceso en un entorno laboral – y no laboral – mejorándolo en todos sus aspectos.

Capítulo 8. BIBLIOGRAFÍA

[11] Apache Spark. "Documentation." https://spark.apache.org/docs/latest/

[12] Delta Lake. "What is Delta Lake?" https://delta.io

[13] Apache Airflow. "Airflow Documentation." https://airflow.apache.org/docs/

[14] Databricks. "Delta Lake Quickstart." https://docs.databricks.com/aws/en/delta/

[15] Github. "Updating filed in XFA PDF #2809" https://github.com/py-pdf/pypdf/discussions/2809

Capítulo 9. ANEXO – SCRIPTS UTILIZADOS

9.1 01_03_INGEST_ISER.PY.J2:

Funcionalidad: Este script es el DAG de Apache Airflow. La extensión .py.j2 es porque se trata de una plantilla de Jinja2. Define el flujo de trabajo para el procesado de los ISER, dividiendo el proceso en tres tareas principales: ingestión, parseo y archivado.

```
#!/usr/bin/env python3
{% include "shared.py.j2" %}

ISER_LANDING_DIR = f"/data/01_landing_zone/iser"

ISER_RAW_DATA_DIR = f"/data/02_raw_data"

ISER_SETS_DIR = f"/data/03_working_sets"

ISER_ARCHIVE_DIR = f"/archive/02_raw_data/iser"

with DAG(
    dag_id="01_03_ingest_iser",
    description="Ingest and parse ISER information",
    schedule_interval=None,
    default_args=DEFAULT_ARGS,
    tags=["iser_parser"],
) as dag:

    version_task = PythonOperator(task_id="capture_version",
python_callable=capture_version, dag=dag)

    ingest_iser = SparkSubmitOperator(
        task_id="ingest_iser",
        application_args=[
            "-f", "ingest.ingest_files", "--",
            "--source-dir", ISER_LANDING_DIR,
            "--target-dir", ISER_RAW_DATA_DIR,
            "--category", "iser",
        ],
        conf={**COMMON_SPARK_CONFIG, **AIRFLOW_SPARK_CONFIG},
        **SUBMIT_OPERATOR_ARGS,
```

```
    )

    version_task >> ingest_iser

    parse_iser = SparkSubmitOperator(
        task_id="parse_iser",
        application_args=[
            "-f", "ingest.parse_iser", "--",
            "--source-dir", ISER_RAW_DATA_DIR,
            "--target-dir", ISER_SETS_DIR,
        ],
        conf={**COMMON_SPARK_CONFIG, **AIRFLOW_SPARK_CONFIG,
**{"spark.executor.memory": "20G", "spark.executor.instances": "1",
"spark.executor.cores": "4"}},
        **SUBMIT_OPERATOR_ARGS,
    )

    ingest_iser >> parse_iser

    archive_iser = SparkSubmitOperator(
        task_id="move_iser_files_to_archive",
        application_args=[
            "-f", "ingest.move_iser_to_archive", "--",
            "--source-dir", ISER_LANDING_DIR,
            "--target-dir", ISER_ARCHIVE_DIR,
            "--pattern", "*.pdf",
        ],
        conf={**COMMON_SPARK_CONFIG, **AIRFLOW_SPARK_CONFIG,
**{"spark.executor.instances": "1"}},
        **SUBMIT_OPERATOR_ARGS,
    )

    parse_iser >> archive_iser
```

9.2 INGEST_FILES.PY

Funcionalidad**:** Esta etapa se encarga de mover los archivos PDF con formularios ISER desde el directorio de landing (`/data/01_landing_zone/iser`) hacia la zona de datos brutos o raw (`/data/02_raw_data`). Esta acción garantiza que los datos estén disponibles en una ubicación centralizada y estructurada para su posterior procesamiento.

Parámetros relevantes:

- `--source-dir`: Directorio de entrada con los archivos originales.
- `--target-dir`: Directorio de destino.
- `--category`: Categoría del proceso (`iser`).

```
import argparse
import logging
import os
from pathlib import Path
from typing import Literal

from pyspark.sql import SparkSession

from files_to_delta import ingest_files_to_delta
from spark import create_spark_session


def ingest_files(
    spark: SparkSession,
    source_dir: Path,
    target_dir: Path,
    category: Literal["iser"],
    create_marker: bool,
) -> None:
    known_file_types = {
        "iser": [
            ("iser", r".+\.pdf"),
        ]
    }

    for file_type, pattern in known_file_types[category]:
        logging.info(f"Processing {source_dir} with file types {file_type}.")
        delta_file_path = target_dir / category / f"{file_type}.delta"
        new_files = ingest_files_to_delta(
            spark=spark,
            source_dir=source_dir,
            delta_file_path=delta_file_path,
            pattern=pattern,
        )

        # Mark the original file for processing by the next task
```

```
        if create_marker:
            for file in new_files:
                (file.parent / f".{file.name}.{category}").touch()
                logging.info(f"Created marker file for {file}")


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="""Ingest unpacked Files:
        Find files in source dir.  Append the matching raw binary content as a
row to the respective delta file in raw data dir.
        Then, set a flag file that the file has been processed."""
    )
    parser.add_argument(
        "--source-dir",
        type=Path,
        help="Ingest Directory containing the zip files.'.",
        required=True,
    )
    parser.add_argument(
        "--target-dir",
        type=Path,
        help="Raw Data Directory.'.",
        required=True,
    )
    parser.add_argument(
        "--category",
        type=str,
        help="File category.",
        required=True,
        choices=["iser"],
    )
    parser.add_argument(
        "--marker",
        type=bool,
        help="Create or don't create marker file. Use --no-marker when reading
archived data.",
        required=False,
        default=True,
        action=argparse.BooleanOptionalAction,
    )
    args = parser.parse_args()

    spark = create_spark_session()

    logging.basicConfig(
        level=logging.getLevelName(os.environ.get("LOGGING", "INFO").upper()),
        format="%(asctime)-15s %(levelname)s %(message)s",
```

```
    )

    ingest_files(spark, args.source_dir, args.target_dir, args.category,
args.marker)
```

9.3 FILES_TO_DELTA.PY

Funcionalidad: Convierte archivos Parquet procesados en tablas delta. Tiene en cuenta además que sólo se procesen nuevos archivos.

```
import logging
import re
import zlib
from collections.abc import Iterator
from pathlib import Path

import pyspark.sql.functions as F
import pyspark.sql.types as T
from pyspark.sql import SparkSession

from deltatables import set_table_properties


def get_existing_files(spark, delta_file_path) -> dict[str, set[int]]:
    return (
        {
            row.file_name: row.crcs
            for row in spark.read.format("delta")
            .load(delta_file_path.as_posix())
            .groupBy("file_name")
            .agg(F.collect_set("crc").alias("crcs"))
            .collect()
        }
        if delta_file_path.exists()
        else {}
    )


def get_new_files(source_dir: Path, pattern: str, existing_files: dict[str,
set[int]]) -> list[Path]:
    rx = re.compile(pattern)
    return [
        file
        for file in source_dir.glob("*")
```

```
        if file.is_file() and rx.match(file.name) and file.name not in
existing_files
    ]


def read_files(files: list[Path]) -> Iterator[tuple[str, int, bytes, str]]:
    for file in files:
        data = file.read_bytes()
        crc = zlib.crc32(data)
        yield file.name, crc, data, file.as_posix()


def ingest_files_to_delta(
    spark: SparkSession, source_dir: Path, delta_file_path: Path, pattern: str,
batch_size: int = 1000
) -> list[Path]:
    logging.info(f"Ingesting {source_dir} with file pattern {pattern} to
{delta_file_path}")

    existing_files = get_existing_files(spark, delta_file_path)
    new_files = get_new_files(source_dir, pattern, existing_files)

    logging.info(f"Ingesting {len(new_files)} new files from {source_dir}")

    schema = T.StructType(
        [
            T.StructField("file_name", T.StringType(), False),
            T.StructField("crc", T.LongType(), False),
            T.StructField("bytes", T.BinaryType(), False),
            T.StructField("ingest_file", T.StringType(), False),
        ]
    )
    batches = [new_files[i : i + batch_size] for i in range(0, len(new_files),
batch_size)]

    df = (
        spark.sparkContext.parallelize(batches)
        .flatMap(read_files)
        .toDF(schema)
        .dropDuplicates(subset=["file_name", "crc"])
        .withColumn("ingest_date", F.current_timestamp())
    )

    n_new_files = df.count()
    if n_new_files > 0:
        df.write.format("delta").mode("append").save(delta_file_path.as_posix())

    # Ensure log retention is set as soon as possible.
```

```
    set_table_properties(spark, delta_file_path)

    logging.info(
        f"Completed ingesting {source_dir} with file pattern {pattern}. Appended
{n_new_files} rows to {delta_file_path}."
    )

    return new_files
```

9.4 DELTATABLES.PY

Función: Define funciones para la escritura de datos con el framework Delta Lake.

```
import logging
from pathlib import Path

from pyspark.sql import SparkSession

logger = logging.getLogger(__name__)

DEFAULT_PROPERTIES = {
    "delta.deletedFileRetentionDuration": "interval 180 days",
    "delta.logRetentionDuration": "interval 365 days",
    "delta.enableChangeDataFeed": "true",
}


def set_table_properties(
    spark: SparkSession, delta_table_path: Path, required_properties: dict[str,
str] | None = None
) -> None:
    if not delta_table_path.exists():
        logger.warning(f"{delta_table_path} does not exists. Skipping.")
        return

    tblproperties = spark.sql(f"SHOW TBLPROPERTIES delta.`{delta_table_path}`")

    required_properties = required_properties or DEFAULT_PROPERTIES

    for key, required_value in required_properties.items():
        actual_property = tblproperties.filter(tblproperties.key ==
key).select("value").collect()
```

```
        if len(actual_property) == 0:
            logger.warning(
                f"Config '{key}' for delta table {delta_table_path} is not yet
set. Setting property to required value '{required_value}'."
            )
            spark.sql(f'ALTER TABLE delta.`{delta_table_path}` SET
TBLPROPERTIES({key} = "{required_value}");')

        elif len(actual_property) == 1:
            actual_value = actual_property[0].value

            if actual_value == required_value:
                logger.info(
                    f"Config '{key}' for delta table {delta_table_path} is set to
'{required_value}' as required. No need to set property."
                )
            else:
                logger.warning(
                    f"Config '{key}' for delta table {delta_table_path} is set to
'{actual_value}' but required is '{required_value}'. Setting property to required
value."
                )
                spark.sql(f'ALTER TABLE delta.`{delta_table_path}` SET
TBLPROPERTIES({key} = "{required_value}");')

        else:
            raise RuntimeError(
                f"Found unexpected number of entries for config {key} in table
properties for table {delta_table_path}"
            )
```

9.5 PARSE_ISER.PY

Funcionalidad: A partir de los archivos almacenados en raw, esta tarea ejecuta el proceso de extracción de datos. El parseo se realiza mediante PySpark, transformando la información contenida en los formularios PDF (formato XFA) en estructuras tabulares almacenadas en `/data/03_working_sets`. Esta transformación facilita su posterior análisis y explotación.

```
"""This file is used to parse the ISER XFA/PDF files."""

import argparse
```

```
import io
import xml.etree.ElementTree as ET
from pathlib import Path

import pypdf
import pyspark.sql.functions as F
from core.processing import delta_stream_foreach_batch
from core.storage.persistence import DeltaArtifact
from pyspark.sql import DataFrame, SparkSession
from pyspark.sql.types import BooleanType, DateType, FloatType, IntegerType,
StringType, StructField, StructType

from spark import create_spark_session

from deltatables import set_table_properties


def findInDict(needle, haystack):
    """
    This function look for an specific object inside of a dictionary.
    The goal for this use case id to search an specific 'tag' inside of the PDF-
XFA structure
    """
    for key in haystack:
        try:
            value = haystack[key]
        except Exception as e:
            raise e
        if key == needle:
            return value
        if isinstance(value, dict):
            x = findInDict(needle, value)
            if x is not None:
                return x
    return None


def get_xml_from_PDF(iser_file: io.BytesIO) -> str:
    """
    The ISER is an specific type of PDF, is a XFA-PDF.
    XFA stands for 'XML Format Architecture' and it means that inside of the PDF
metadata, there is a XML indicating all the structure and information.
    The objective of this function is to get the information of the XML inside of
the ISER file (using the 'findInDict' function).
    """
    # Read the PDF with the library pypdf
    pdf = pypdf.PdfReader(iser_file)
```

```
    # Use the function findInDict() to search for the tag '/XFA' into the PDF
struture
    xfa = findInDict("/XFA", pdf.resolved_objects)
    # Once found the XFA dictionary, find the 9th element that is where the
structure and data is
    return xfa[9].get_object().get_data()


def parse_iser_xml(iser_xml: str) -> dict[str, None]:
    """
    This function takes as input the XML from the ISER XFA-PDF.
    Then, it creates a list of dicts, one per tag and with its information/tag-
value.
    It returns that list of dicts with the data.
    """
    # List with all the possible tags inside of the ISER XML structure. This has
been done manually and it is checked it is OK and complete.
    tags = [
        "Org_Name",
        "Org_Country",
        "Appr_Ref",
        "Submitter",
        "Mail",
        "Telephone",
        "Internal_Ref_No",
        "Issue",
        "Report_Date",
        "Report_Type",
        "Initial_Report_Date",
        "Occurrence",
        "Title_or_Summary",
        "Date_of_Finding",
        "ATA",
        "ISE_Time",
        "Location",
        "Departure",
        "Destination",
        "Flight_Nature",
        "Detect_Phase",
        "Detect_Phase_Other",
        "Cause_Design",
        "Cause_Operational",
        "Cause_HumanFactors",
        "Cause_UnapprovedParts",
        "Cause_Repair",
        "Cause_NotDetermined",
        "Cause_Production",
        "Cause_Fatigue",
```

```
        "Cause_Maintenance",
        "Cause_Corrosion",
        "Cause_Other",
        "Cause_Text",
        "System_Monitoring",
        "References1",
        "References2",
        "References3",
        "References4",
        "References5",
        "References6",
        "Consequence_Delay",
        "Delay_Time",
        "Delay_Text",
        "Consequence_Cancellation",
        "Cancellation_Text",
        "Consequence_Turn_Back",
        "Turn_Back_Text",
        "Consequence_Diversion",
        "Diversion_Text",
        "Consequence_Aborted_TakeOff",
        "Aborted_TakeOff_Text",
        "Consequence_Eng_ShutDown",
        "Aircraft",
        "AC_SN",
        "Operator",
        "Registration",
        "AC_TSN",
        "AC_CSN",
        "Engine_Pos_1",
        "Engine_Pos_2",
        "Engine_Pos_3",
        "Engine_Pos_4",
        "Eng_SN_Pos_1",
        "Eng_SN_Pos_2",
        "Eng_SN_Pos_3",
        "Eng_SN_Pos_4",
        "Eng_PN_Pos_1",
        "Eng_PN_Pos_2",
        "Eng_PN_Pos_3",
        "Eng_PN_Pos_4",
        "Eng_AC_Pos",
        "Eng_Event_UncontainedFailure",
        "Eng_Event_Fire",
        "Eng_Event_Shutdown",
        "Eng_Event_LOTC_LOPC",
        "Eng_Event_Other",
        "Eng_TSN_Pos_1",
```

```
        "Eng_TSN_Pos_2",
        "Eng_TSN_Pos_3",
        "Eng_TSN_Pos_4",
        "Eng_TSO_Pos_1",
        "Eng_TSO_Pos_2",
        "Eng_TSO_Pos_3",
        "Eng_TSO_Pos_4",
        "Eng_TSR_Pos_1",
        "Eng_TSR_Pos_2",
        "Eng_TSR_Pos_3",
        "Eng_TSR_Pos_4",
        "Eng_CSN_Pos_1",
        "Eng_CSN_Pos_2",
        "Eng_CSN_Pos_3",
        "Eng_CSN_Pos_4",
        "Eng_CSO_Pos_1",
        "Eng_CSO_Pos_2",
        "Eng_CSO_Pos_3",
        "Eng_CSO_Pos_4",
        "Eng_CSR_Pos_1",
        "Eng_CSR_Pos_2",
        "Eng_CSR_Pos_3",
        "Eng_CSR_Pos_4",
        "Propel_Model",
        "Propel_SN",
        "Propel_PN",
        "Propel_Event_Overspeed",
        "Propel_Event_Overtorque",
        "Propel_Event_Un_Pitch_C",
        "Propel_Event_Un_Pitch_Lock",
        "Propel_Event_Other",
        "Propel_AC_Pos",
        "Propel_TSN",
        "Propel_TSO",
        "Propel_TSR",
        "Propel_CSN",
        "Propel_CSO",
        "Propel_CSR",
        "Side",
        "ZoneSubZone",
        "Frame1",
        "Rib1",
        "Stringer1",
        "Frame2",
        "Rib2",
        "Stringer2",
        "DamageType",
        "DamageOther",
```

```
        "Comp_Manuf",
        "Comp_Name",
        "Comp_NCAGE",
        "Comp_PN",
        "Comp_SN",
        "Comp_PDesignation",
        "Comp_Date_Manuf",
        "Comp_E_TSO",
        "Comp_TSN",
        "Comp_TSO",
        "Comp_CSN",
        "Comp_CSO",
        "DFDR",
        "Narrative_1",
        "Narrative_2",
        "Narrative_3",
        "Correct_Act",
    ]
    # Create a empty list to collect here dicts with data, one per 'XML tag'
    data_from_dict = {}
    # As the XML source is text/string, we use directly root
    root = ET.fromstring(iser_xml)
    # Iterate through all the XML-PDF structure to get the information from every
"tag"
    for item1 in root.findall("./{http://www.xfa.org/schema/xfa-
data/1.0/}data/Stream/Item"):
        """
        The "Location" tag is repeated inside of the XML structure. To deal with
it, the second "Location" tag is changed to "Side".
        """
        location = 0
        for child1 in item1:
            for item2 in child1.iter():
                if item2.tag in tags:
                    if item2.tag == "Location" and location == 0:
                        data_from_dict = {**data_from_dict, item2.tag:
item2.text}
                        location += 1
                    elif item2.tag == "Location" and location == 1:
                        data_from_dict = {**data_from_dict, "Side": item2.text}
                    else:
                        data_from_dict = {**data_from_dict, item2.tag:
item2.text}
    return data_from_dict


def parse_iser_file(iser_file: io.BytesIO) -> list[dict[str, None]]:
    """
```

```
    This function uses get_xml_from_PDF() to open the ISER PDF file and take the
XML information to then use parse_iser_xml() to transform all that information
    into a table
    """
    # Get the XML from the PDF file
    xml_info = get_xml_from_PDF(iser_file)
    # Takes the XML and returns the data within a dictionary
    return [parse_iser_xml(xml_info)]


def parse_iser_files(iser: DataFrame) -> DataFrame:
    """
    This function generate an output with all the information inside of the
(XFA)-PDF files.
    """
    # Initiate spark
    spark = SparkSession.getActiveSession()
    if not spark:
        raise RuntimeError("No active SparkSession")
    # Define the schema except for boolean and date data type columns
    iser_schema = StructType(
        [
            StructField("Org_Name", StringType(), True),
            StructField("Org_Contry", StringType(), True),
            StructField("Appr_Ref", StringType(), True),
            StructField("Submitter", StringType(), True),
            StructField("Mail", StringType(), True),
            StructField("Telephone", StringType(), True),
            StructField("Internal_Ref_No", StringType(), True),
            StructField("Issue", StringType(), True),
            StructField("Report_Date", StringType(), True),
            StructField("Report_Type", StringType(), True),
            StructField("Initial_Report_Date", StringType(), True),
            StructField("Occurrence", StringType(), True),
            StructField("Title_or_Summary", StringType(), True),
            StructField("Date_of_Finding", StringType(), True),
            StructField("ATA", StringType(), True),
            StructField("ISE_Time", StringType(), True),
            StructField("Location", StringType(), True),
            StructField("Departure", StringType(), True),
            StructField("Destination", StringType(), True),
            StructField("Flight_Nature", StringType(), True),
            StructField("Detect_Phase", StringType(), True),
            StructField("Detect_Phase_Other", StringType(), True),
            StructField("Cause_Design", StringType(), True),
            StructField("Cause_Operational", StringType(), True),
            StructField("Cause_HumanFactors", StringType(), True),
            StructField("Cause_UnapprovedParts", StringType(), True),
```

```
            StructField("Cause_Repair", StringType(), True),
            StructField("Cause_NotDetermined", StringType(), True),
            StructField("Cause_Production", StringType(), True),
            StructField("Cause_Fatigue", StringType(), True),
            StructField("Cause_Maintenance", StringType(), True),
            StructField("Cause_Corrosion", StringType(), True),
            StructField("Cause_Other", StringType(), True),
            StructField("Cause_Text", StringType(), True),
            StructField("System_Monitoring", StringType(), True),
            StructField("References1", StringType(), True),
            StructField("References2", StringType(), True),
            StructField("References3", StringType(), True),
            StructField("References4", StringType(), True),
            StructField("References5", StringType(), True),
            StructField("References6", StringType(), True),
            StructField("Consequence_Delay", StringType(), True),
            StructField("Delay_Time", StringType(), True),
            StructField("Delay_Text", StringType(), True),
            StructField("Consequence_Cancellation", StringType(), True),
            StructField("Cancellation_Text", StringType(), True),
            StructField("Consequence_Turn_Back", StringType(), True),
            StructField("Turn_Back_Text", StringType(), True),
            StructField("Consequence_Diversion", StringType(), True),
            StructField("Diversion_Text", StringType(), True),
            StructField("Consequence_Aborted_TakeOff", StringType(), True),
            StructField("Aborted_TakeOff_Text", StringType(), True),
            StructField("Consequence_Eng_ShutDown", StringType(), True),
            StructField("Aircraft", StringType(), True),
            StructField("AC_SN", StringType(), True),
            StructField("Operator", StringType(), True),
            StructField("Registration", StringType(), True),
            StructField("AC_TSN", StringType(), True),
            StructField("AC_CSN", StringType(), True),
            StructField("Engine_Pos_1", StringType(), True),
            StructField("Engine_Pos_2", StringType(), True),
            StructField("Engine_Pos_3", StringType(), True),
            StructField("Engine_Pos_4", StringType(), True),
            StructField("Eng_SN_Pos_1", StringType(), True),
            StructField("Eng_SN_Pos_2", StringType(), True),
            StructField("Eng_SN_Pos_3", StringType(), True),
            StructField("Eng_SN_Pos_4", StringType(), True),
            StructField("Eng_PN_Pos_1", StringType(), True),
            StructField("Eng_PN_Pos_2", StringType(), True),
            StructField("Eng_PN_Pos_3", StringType(), True),
            StructField("Eng_PN_Pos_4", StringType(), True),
            StructField("Eng_AC_Pos", StringType(), True),
            StructField("Eng_Event_UncontainedFailure", StringType(), True),
            StructField("Eng_Event_Fire", StringType(), True),
```

```
            StructField("Eng_Event_Shutdown", StringType(), True),
            StructField("Eng_Event_LOTC_LOPC", StringType(), True),
            StructField("Eng_Event_Other", StringType(), True),
            StructField("Eng_TSN_Pos_1", FloatType(), True),
            StructField("Eng_TSN_Pos_2", FloatType(), True),
            StructField("Eng_TSN_Pos_3", FloatType(), True),
            StructField("Eng_TSN_Pos_4", FloatType(), True),
            StructField("Eng_TSO_Pos_1", FloatType(), True),
            StructField("Eng_TSO_Pos_2", FloatType(), True),
            StructField("Eng_TSO_Pos_3", FloatType(), True),
            StructField("Eng_TSO_Pos_4", FloatType(), True),
            StructField("Eng_TSR_Pos_1", FloatType(), True),
            StructField("Eng_TSR_Pos_2", FloatType(), True),
            StructField("Eng_TSR_Pos_3", FloatType(), True),
            StructField("Eng_TSR_Pos_4", FloatType(), True),
            StructField("Eng_CSN_Pos_1", IntegerType(), True),
            StructField("Eng_CSN_Pos_2", IntegerType(), True),
            StructField("Eng_CSN_Pos_3", IntegerType(), True),
            StructField("Eng_CSN_Pos_4", IntegerType(), True),
            StructField("Eng_CSO_Pos_1", IntegerType(), True),
            StructField("Eng_CSO_Pos_2", IntegerType(), True),
            StructField("Eng_CSO_Pos_3", IntegerType(), True),
            StructField("Eng_CSO_Pos_4", IntegerType(), True),
            StructField("Eng_CSR_Pos_1", IntegerType(), True),
            StructField("Eng_CSR_Pos_2", IntegerType(), True),
            StructField("Eng_CSR_Pos_3", IntegerType(), True),
            StructField("Eng_CSR_Pos_4", IntegerType(), True),
            StructField("Propel_Model", StringType(), True),
            StructField("Propel_SN", StringType(), True),
            StructField("Propel_PN", StringType(), True),
            StructField("Propel_Event_Overspeed", StringType(), True),
            StructField("Propel_Event_Overtorque", StringType(), True),
            StructField("Propel_Event_Un_Pitch_C", StringType(), True),
            StructField("Propel_Event_Un_Pitch_Lock", StringType(), True),
            StructField("Propel_Event_Other", StringType(), True),
            StructField("Propel_AC_Pos", StringType(), True),
            StructField("Propel_TSN", FloatType(), True),
            StructField("Propel_TSO", FloatType(), True),
            StructField("Propel_TSR", FloatType(), True),
            StructField("Propel_CSN", IntegerType(), True),
            StructField("Propel_CSO", IntegerType(), True),
            StructField("Propel_CSR", IntegerType(), True),
            StructField("Side", StringType(), True),
            StructField("ZoneSubZone", StringType(), True),
            StructField("Frame1", StringType(), True),
            StructField("Rib1", StringType(), True),
            StructField("Stringer1", StringType(), True),
            StructField("Frame2", StringType(), True),
```

```
            StructField("Rib2", StringType(), True),
            StructField("Stringer2", StringType(), True),
            StructField("DamageType", StringType(), True),
            StructField("DamageOther", StringType(), True),
            StructField("Comp_Manuf", StringType(), True),
            StructField("Comp_Name", StringType(), True),
            StructField("Comp_NCAGE", StringType(), True),
            StructField("Comp_PN", StringType(), True),
            StructField("Comp_SN", StringType(), True),
            StructField("Comp_PDesignation", StringType(), True),
            StructField("Comp_Date_Manuf", StringType(), True),
            StructField("Comp_E_TSO", StringType(), True),
            StructField("Comp_TSN", FloatType(), True),
            StructField("Comp_TSO", FloatType(), True),
            StructField("Comp_CSN", IntegerType(), True),
            StructField("Comp_CSO", IntegerType(), True),
            StructField("DFDR", StringType(), True),
            StructField("Narrative_1", StringType(), True),
            StructField("Narrative_2", StringType(), True),
            StructField("Narrative_3", StringType(), True),
            StructField("Correct_Act", StringType(), True),
        ]
    )
    # Define the schema for the desired output
    iser_schema_processed = StructType(
        [
            StructField("Org_Name", StringType(), True),
            StructField("Org_Contry", StringType(), True),
            StructField("Appr_Ref", StringType(), True),
            StructField("Submitter", StringType(), True),
            StructField("Mail", StringType(), True),
            StructField("Telephone", StringType(), True),
            StructField("Internal_Ref_No", StringType(), True),
            StructField("Issue", StringType(), True),
            StructField("Report_Date", DateType(), True),
            StructField("Report_Type", StringType(), True),
            StructField("Initial_Report_Date", StringType(), True),
            StructField("Occurrence", StringType(), True),
            StructField("Title_or_Summary", StringType(), True),
            StructField("Date_of_Finding", StringType(), True),
            StructField("ATA", StringType(), True),
            StructField("ISE_Time", StringType(), True),
            StructField("Location", StringType(), True),
            StructField("Departure", StringType(), True),
            StructField("Destination", StringType(), True),
            StructField("Flight_Nature", StringType(), True),
            StructField("Detect_Phase", StringType(), True),
            StructField("Detect_Phase_Other", StringType(), True),
```

```
            StructField("Cause_Design", BooleanType(), True),
            StructField("Cause_Operational", BooleanType(), True),
            StructField("Cause_HumanFactors", BooleanType(), True),
            StructField("Cause_UnapprovedParts", BooleanType(), True),
            StructField("Cause_Repair", BooleanType(), True),
            StructField("Cause_NotDetermined", BooleanType(), True),
            StructField("Cause_Production", BooleanType(), True),
            StructField("Cause_Fatigue", BooleanType(), True),
            StructField("Cause_Maintenance", BooleanType(), True),
            StructField("Cause_Corrosion", BooleanType(), True),
            StructField("Cause_Other", BooleanType(), True),
            StructField("Cause_Text", StringType(), True),
            StructField("System_Monitoring", StringType(), True),
            StructField("References1", StringType(), True),
            StructField("References2", StringType(), True),
            StructField("References3", StringType(), True),
            StructField("References4", StringType(), True),
            StructField("References5", StringType(), True),
            StructField("References6", StringType(), True),
            StructField("Consequence_Delay", BooleanType(), True),
            StructField("Delay_Time", StringType(), True),
            StructField("Delay_Text", StringType(), True),
            StructField("Consequence_Cancellation", BooleanType(), True),
            StructField("Cancellation_Text", StringType(), True),
            StructField("Consequence_Turn_Back", BooleanType(), True),
            StructField("Turn_Back_Text", StringType(), True),
            StructField("Consequence_Diversion", BooleanType(), True),
            StructField("Diversion_Text", StringType(), True),
            StructField("Consequence_Aborted_TakeOff", BooleanType(), True),
            StructField("Aborted_TakeOff_Text", StringType(), True),
            StructField("Consequence_Eng_ShutDown", BooleanType(), True),
            StructField("Aircraft", StringType(), True),
            StructField("AC_SN", StringType(), True),
            StructField("Operator", StringType(), True),
            StructField("Registration", StringType(), True),
            StructField("AC_TSN", StringType(), True),
            StructField("AC_CSN", StringType(), True),
            StructField("Engine_Pos_1", StringType(), True),
            StructField("Engine_Pos_2", StringType(), True),
            StructField("Engine_Pos_3", StringType(), True),
            StructField("Engine_Pos_4", StringType(), True),
            StructField("Eng_SN_Pos_1", StringType(), True),
            StructField("Eng_SN_Pos_2", StringType(), True),
            StructField("Eng_SN_Pos_3", StringType(), True),
            StructField("Eng_SN_Pos_4", StringType(), True),
            StructField("Eng_PN_Pos_1", StringType(), True),
            StructField("Eng_PN_Pos_2", StringType(), True),
            StructField("Eng_PN_Pos_3", StringType(), True),
```

```
            StructField("Eng_PN_Pos_4", StringType(), True),
            StructField("Eng_AC_Pos", StringType(), True),
            StructField("Eng_Event_UncontainedFailure", BooleanType(), True),
            StructField("Eng_Event_Fire", BooleanType(), True),
            StructField("Eng_Event_Shutdown", BooleanType(), True),
            StructField("Eng_Event_LOTC_LOPC", BooleanType(), True),
            StructField("Eng_Event_Other", BooleanType(), True),
            StructField("Eng_TSN_Pos_1", FloatType(), True),
            StructField("Eng_TSN_Pos_2", FloatType(), True),
            StructField("Eng_TSN_Pos_3", FloatType(), True),
            StructField("Eng_TSN_Pos_4", FloatType(), True),
            StructField("Eng_TSO_Pos_1", FloatType(), True),
            StructField("Eng_TSO_Pos_2", FloatType(), True),
            StructField("Eng_TSO_Pos_3", FloatType(), True),
            StructField("Eng_TSO_Pos_4", FloatType(), True),
            StructField("Eng_TSR_Pos_1", FloatType(), True),
            StructField("Eng_TSR_Pos_2", FloatType(), True),
            StructField("Eng_TSR_Pos_3", FloatType(), True),
            StructField("Eng_TSR_Pos_4", FloatType(), True),
            StructField("Eng_CSN_Pos_1", IntegerType(), True),
            StructField("Eng_CSN_Pos_2", IntegerType(), True),
            StructField("Eng_CSN_Pos_3", IntegerType(), True),
            StructField("Eng_CSN_Pos_4", IntegerType(), True),
            StructField("Eng_CSO_Pos_1", IntegerType(), True),
            StructField("Eng_CSO_Pos_2", IntegerType(), True),
            StructField("Eng_CSO_Pos_3", IntegerType(), True),
            StructField("Eng_CSO_Pos_4", IntegerType(), True),
            StructField("Eng_CSR_Pos_1", IntegerType(), True),
            StructField("Eng_CSR_Pos_2", IntegerType(), True),
            StructField("Eng_CSR_Pos_3", IntegerType(), True),
            StructField("Eng_CSR_Pos_4", IntegerType(), True),
            StructField("Propel_Model", StringType(), True),
            StructField("Propel_SN", StringType(), True),
            StructField("Propel_PN", StringType(), True),
            StructField("Propel_Event_Overspeed", BooleanType(), True),
            StructField("Propel_Event_Overtorque", BooleanType(), True),
            StructField("Propel_Event_Un_Pitch_C", BooleanType(), True),
            StructField("Propel_Event_Un_Pitch_Lock", BooleanType(), True),
            StructField("Propel_Event_Other", BooleanType(), True),
            StructField("Propel_AC_Pos", StringType(), True),
            StructField("Propel_TSN", FloatType(), True),
            StructField("Propel_TSO", FloatType(), True),
            StructField("Propel_TSR", FloatType(), True),
            StructField("Propel_CSN", IntegerType(), True),
            StructField("Propel_CSO", IntegerType(), True),
            StructField("Propel_CSR", IntegerType(), True),
            StructField("Side", StringType(), True),
            StructField("ZoneSubZone", StringType(), True),
```

```
            StructField("Frame1", StringType(), True),
            StructField("Rib1", StringType(), True),
            StructField("Stringer1", StringType(), True),
            StructField("Frame2", StringType(), True),
            StructField("Rib2", StringType(), True),
            StructField("Stringer2", StringType(), True),
            StructField("DamageType", StringType(), True),
            StructField("DamageOther", StringType(), True),
            StructField("Comp_Manuf", StringType(), True),
            StructField("Comp_Name", StringType(), True),
            StructField("Comp_NCAGE", StringType(), True),
            StructField("Comp_PN", StringType(), True),
            StructField("Comp_SN", StringType(), True),
            StructField("Comp_PDesignation", StringType(), True),
            StructField("Comp_Date_Manuf", StringType(), True),
            StructField("Comp_E_TSO", StringType(), True),
            StructField("Comp_TSN", FloatType(), True),
            StructField("Comp_TSO", FloatType(), True),
            StructField("Comp_CSN", IntegerType(), True),
            StructField("Comp_CSO", IntegerType(), True),
            StructField("DFDR", StringType(), True),
            StructField("Narrative_1", StringType(), True),
            StructField("Narrative_2", StringType(), True),
            StructField("Narrative_3", StringType(), True),
            StructField("Correct_Act", StringType(), True),
        ]
    )
    # Create an empty dataframe in wich store all the parsed data
    ISERs_spark_df_processed = spark.createDataFrame([], iser_schema_processed)
    # Iterate through all the ISER files
    for row in iser.collect():
        # Create the dataframe using the data and the schema
        ISER_spark_df =
spark.createDataFrame(parse_iser_file(io.BytesIO(row["bytes"])), iser_schema)
        # Change the data types of the spark dataframe for the boolean and date
columns
        ISER_spark_df = (
            ISER_spark_df.withColumn("Report_Date",
F.col("Report_Date").cast(DateType()))
            .withColumn("Cause_Design",
F.col("Cause_Design").cast(BooleanType()))
            .withColumn("Cause_Operational",
F.col("Cause_Operational").cast(BooleanType()))
            .withColumn("Cause_HumanFactors",
F.col("Cause_HumanFactors").cast(BooleanType()))
            .withColumn("Cause_UnapprovedParts",
F.col("Cause_UnapprovedParts").cast(BooleanType()))
```

```
            .withColumn("Cause_Repair",
F.col("Cause_Repair").cast(BooleanType()))
            .withColumn("Cause_NotDetermined",
F.col("Cause_NotDetermined").cast(BooleanType()))
            .withColumn("Cause_Production",
F.col("Cause_Production").cast(BooleanType()))
            .withColumn("Cause_Fatigue",
F.col("Cause_Fatigue").cast(BooleanType()))
            .withColumn("Cause_Maintenance",
F.col("Cause_Maintenance").cast(BooleanType()))
            .withColumn("Cause_Corrosion",
F.col("Cause_Corrosion").cast(BooleanType()))
            .withColumn("Cause_Other", F.col("Cause_Other").cast(BooleanType()))
            .withColumn(
                "Consequence_Delay",
                F.when(F.col("Consequence_Delay") == "1",
True).otherwise(False).cast(BooleanType()),
            )
            .withColumn("Consequence_Cancellation",
F.col("Consequence_Cancellation").cast(BooleanType()))
            .withColumn("Consequence_Turn_Back",
F.col("Consequence_Turn_Back").cast(BooleanType()))
            .withColumn("Consequence_Diversion",
F.col("Consequence_Diversion").cast(BooleanType()))
            .withColumn("Consequence_Aborted_TakeOff",
F.col("Consequence_Aborted_TakeOff").cast(BooleanType()))
            .withColumn("Consequence_Eng_ShutDown",
F.col("Consequence_Eng_ShutDown").cast(BooleanType()))
            .withColumn("Eng_Event_UncontainedFailure",
F.col("Eng_Event_UncontainedFailure").cast(BooleanType()))
            .withColumn("Eng_Event_Fire",
F.col("Eng_Event_Fire").cast(BooleanType()))
            .withColumn("Eng_Event_Shutdown",
F.col("Eng_Event_Shutdown").cast(BooleanType()))
            .withColumn("Eng_Event_LOTC_LOPC",
F.col("Eng_Event_LOTC_LOPC").cast(BooleanType()))
            .withColumn("Eng_Event_Other",
F.col("Eng_Event_Other").cast(BooleanType()))
            .withColumn("Propel_Event_Overspeed",
F.col("Propel_Event_Overspeed").cast(BooleanType()))
            .withColumn("Propel_Event_Overtorque",
F.col("Propel_Event_Overtorque").cast(BooleanType()))
            .withColumn("Propel_Event_Un_Pitch_C",
F.col("Propel_Event_Un_Pitch_C").cast(BooleanType()))
            .withColumn("Propel_Event_Un_Pitch_Lock",
F.col("Propel_Event_Un_Pitch_Lock").cast(BooleanType()))
            .withColumn("Propel_Event_Other",
F.col("Propel_Event_Other").cast(BooleanType()))
```

```
        )
        # Join all the iser-dataframes into one
        ISERs_spark_df_processed = ISERs_spark_df_processed.union(ISER_spark_df)
    return ISERs_spark_df_processed


def parse_iser(spark: SparkSession, source_dir: Path, target_dir: Path):
    source = source_dir / "iser" / "iser.delta"
    target = target_dir / "iser" / "iser.delta"

    delta_stream_foreach_batch(
        spark=spark,
        stream_input=DeltaArtifact(path=source),
        batch_inputs=[],
        output=target,
        function=parse_iser_files,
    )

    # Ensure log retention is set as soon as possible.
    set_table_properties(spark, target)


if __name__ == "__main__":
    # Create Spark session
    spark = create_spark_session()

    # Define arguments to use from DAG file "01_03_ingest_iser.py.j2"
    parser = argparse.ArgumentParser(
        description="""Parse ISER files:
       This process takes the ISER PDF files and parses its XML data into a
tabular format to then
       save it as a delta table.
       """
    )
    parser.add_argument("--source-dir", type=Path, help="Path where the ISER
files are stored.", required=True)
    parser.add_argument(
        "--target-dir", type=Path, help="Path where the dataset with the info
parsed is stored.", required=True
    )
    args = parser.parse_args()

    parse_iser(spark=spark, source_dir=args.source_dir,
target_dir=args.target_dir)
```

9.6 MOVE_ISER_TO_ARCHIVE.PY

Funcionalidad: Una vez parseados los datos, los archivos PDF originales son movidos a un directorio de archivo (`/archive/02_raw_data/iser`). Esto permite conservar el histórico de documentos procesados, evitando su reprocesamiento accidental y garantizando la trazabilidad.

```
import argparse
import hashlib
import logging
import os
import shutil
from pathlib import Path


def archive_iser_files(data_dir: Path, target_dir: Path, pattern: str):
    """
    Creates a list of all the files that are going to be moved and then iterates
through each of one to move it.
    """
    for file_path in data_dir.glob(pattern):
        archive_iser_file(file_path, target_dir)


def archive_iser_file(file_path: Path, target_dir: Path) -> None:
    """
    Move ISER.pdf file to archive, set permissions to readonly, print logs about
what it is being done.
    """
    logging.info(f"Processing {file_path} ...")

    hash = compute_md5(file_path)

    archive_path = target_dir / file_path.name
    archive_path.parent.mkdir(parents=True, exist_ok=True)

    if archive_path.exists():
        # Check if the file is already archived by comparing MD5 hashes
        if hash == compute_md5(archive_path):
            logging.info(f"File {file_path} is already archived to
{archive_path}. Skipping.")
        else:
```

```
            raise RuntimeError(f"File {file_path} exists in archive but with
different content.")
    else:
        # Copy the file to the archive
        shutil.copy2(file_path, archive_path)
        if not hash == compute_md5(archive_path):
            raise RuntimeError(f"Copied {archive_path} does not have the same
hash.")
        logging.info(f"Copied {file_path} to {archive_path}.")

    # Set the copied file to read-only for user and group
    archive_path.chmod(0o440)
    logging.info(f"Set {archive_path} to read-only.")

    # Remove the original file
    file_path.unlink()
    logging.info(f"Deleted {file_path} after copying to {archive_path}")


def compute_md5(file_path: Path, chunk_size: int = 4096) -> str:
    """
    Compute the MD5 hash of a file.
    """
    hash = hashlib.md5()
    with open(file_path, "rb") as f:
        while chunk := f.read(chunk_size):
            hash.update(chunk)
    return hash.hexdigest()


if __name__ == "__main__":
    logging.basicConfig(
        level=logging.getLevelName(os.environ.get("LOGGING", "INFO").upper()),
        format="%(asctime)-15s %(levelname)s %(message)s",
    )

    parser = argparse.ArgumentParser(
        description="""Archive ISER files:
        Copy ISER(.pdf) files to the archive, make the copy readonly,
        and set a flag file that the file has been archived."""
    )
    parser.add_argument("--source-dir", type=Path, help="Path of data folder to
copy from.", required=True)
    parser.add_argument("--target-dir", type=Path, help="Path of the archive
folder to copy to.", required=True)
    parser.add_argument(
        "--pattern",
        type=str,
```

```
        help="Pattern of files to be archived. They are going to be PDF files, so
'*.pdf'",
        required=True,
        default="*.pdf",
    )
    args = parser.parse_args()

    archive_iser_files(data_dir=args.source_dir, target_dir=args.target_dir,
pattern=args.pattern)
```

9.7 ISERS.PY

Funcionalidad: Aplica transformaciones, uniones y limpieza de datos sobre los conjuntos procesados previamente. Esta tabla es el resultado analítico final del pipeline y la utilizada para generar los dashboards.

```
from functools import partial

import pyspark.sql.functions as F
from pyspark.sql import DataFrame

from general import apply_functions_sequentially


def create_isers(
    isers_table: DataFrame,
) -> DataFrame:
    """
    Generates the iser table. This table is a selection of fields from the In
Service Event Reports (the ISER).

    · Consequence column processed to not only keep in the dataset only the most
restrictive but convert the text information into an acronym
    · Detection Phase column processed to add the free text value when category
Other is marked
    · Selection of columns from the whole ISER working dataset to the desired
data product
    · Rename of columns to fit standards

    :param isers_table: DataFrame containing all the data in the ISER pdf files
(not processed, in a raw format)
```

```
    :return: A dataframe containing a selection of fields with the main
information about the In-Service reports and needed for the use case

    >>> spark = getfixture('spark')
    >>> from datetime import datetime as dt
    >>> isers_table = spark.createDataFrame([
    ...     Row(Internal_Ref_No="ref_1", Title_or_Summary="title_1",
Date_of_Finding=dt(2025,1,1,0,0,0), Operator="operator_1", AC_SN="msn_1",
ATA="ata_1", Detect_Phase="phase_1", Detect_Phase_Other=None,
Consequence_Delay=True, Consequence_Cancellation=True,
Consequence_Turn_Back=True, Consequence_Diversion=False,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False,
Registration="tail_no_1"),
    ...     Row(Internal_Ref_No="ref_2", Title_or_Summary="title_2",
Date_of_Finding=dt(2025,1,2,0,0,0), Operator="operator_1", AC_SN="msn_1",
ATA="ata_1", Detect_Phase="Other", Detect_Phase_Other="phase_other",
Consequence_Delay=False, Consequence_Cancellation=False,
Consequence_Turn_Back=False, Consequence_Diversion=False,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False,
Registration="tail_no_1"),
    ...     ])
    >>> create_isers(isers_table).show(truncate=False)
    +---------------+-------+-------------------+----------+-----+-----+---------
-+-----------+---------------+
    |internal_ref_nb|title  |finding_date       |operator  |msn  |ata
|ac_tail_no|consequence|detection_phase|
    +---------------+-------+-------------------+----------+-----+-----+---------
-+-----------+---------------+
    |ref_1          |title_1|2025-01-01 00:00:00|operator_1|msn_1|ata_1|tail_no_1
|IFTB       |phase_1        |
    |ref_2          |title_2|2025-01-02 00:00:00|operator_1|msn_1|ata_1|tail_no_1
|null       |phase_other    |
    +---------------+-------+-------------------+----------+-----+-----+---------
-+-----------+---------------+
    <BLANKLINE>
    """

    isers_table_columns = [
        "Internal_Ref_No",
        "Title_or_Summary",
        "Date_of_Finding",
        "Operator",
        "AC_SN",
        "ATA",
        "Detect_Phase",
        "Detect_Phase_Other",
        "Consequence_Delay",
        "Consequence_Cancellation",
```

```
        "Consequence_Turn_Back",
        "Consequence_Diversion",
        "Consequence_Aborted_TakeOff",
        "Consequence_Eng_ShutDown",
        "Registration",
    ]

    columns_to_drop = [
        "Detect_Phase",
        "Detect_Phase_Other",
        "Consequence_Delay",
        "Consequence_Cancellation",
        "Consequence_Turn_Back",
        "Consequence_Diversion",
        "Consequence_Aborted_TakeOff",
        "Consequence_Eng_ShutDown",
    ]

    new_columns_dict = {
        "Internal_Ref_No": "internal_ref_nb",
        "Title_or_Summary": "title",
        "Date_of_Finding": "finding_date",
        "Operator": "operator",
        "AC_SN": "msn",
        "ATA": "ata",
        "Registration": "ac_tail_no",
    }

    return (
        apply_functions_sequentially(
            isers_table.select(isers_table_columns),
            [
                process_consequence,
                process_detection_phase,
                partial(DataFrame.withColumnsRenamed, colsMap=new_columns_dict),
            ],
        )

).drop(*columns_to_drop).write.mode("append").parquet("/data/04_data_products/iser")


def process_consequence(isers_table: DataFrame) -> DataFrame:
    """
    This function checks all the consequences reported in the ISER file creating a new column called "consequence" in which is selected only the most restrictive one.
    The order of restriction is as follows:
```

```
        1. Engine In Flight Shut Down (EIFSD)
        2. Aborted Take Off (ATO)
        3. Diversion (DV)
        4. In Flight Turn Back (IFTB)
        5. Cancellation (CN)
        6. Delay (DY)

    :param isers_table: DataFrame containing the information about the
consequence of the In-Service event
    :return: A dataframe with a new column called "consequence" containing only
the most restrictive consequence reported in the ISER file

    >>> spark = getfixture('spark')
    >>> from datetime import datetime as dt
    >>> isers_table = spark.createDataFrame([
    ...     Row(Consequence_Delay=True, Consequence_Cancellation=True,
Consequence_Turn_Back=True, Consequence_Diversion=True,
Consequence_Aborted_TakeOff=True, Consequence_Eng_ShutDown=True),
    ...     Row(Consequence_Delay=True, Consequence_Cancellation=True,
Consequence_Turn_Back=True, Consequence_Diversion=True,
Consequence_Aborted_TakeOff=True, Consequence_Eng_ShutDown=False),
    ...     Row(Consequence_Delay=True, Consequence_Cancellation=True,
Consequence_Turn_Back=True, Consequence_Diversion=True,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False),
    ...     Row(Consequence_Delay=True, Consequence_Cancellation=True,
Consequence_Turn_Back=True, Consequence_Diversion=False,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False),
    ...     Row(Consequence_Delay=True, Consequence_Cancellation=True,
Consequence_Turn_Back=False, Consequence_Diversion=False,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False),
    ...     Row(Consequence_Delay=True, Consequence_Cancellation=False,
Consequence_Turn_Back=False, Consequence_Diversion=False,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False),
    ...     Row(Consequence_Delay=False, Consequence_Cancellation=False,
Consequence_Turn_Back=False, Consequence_Diversion=False,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False),
    ...     Row(Consequence_Delay=False, Consequence_Cancellation=True,
Consequence_Turn_Back=False, Consequence_Diversion=True,
Consequence_Aborted_TakeOff=False, Consequence_Eng_ShutDown=False),
    ...     ])
    >>> process_consequence(isers_table).show()
    +-----------------+----------------------+--------------------+----------
----------+--------------------------+----------------------+-----------+

|Consequence_Delay|Consequence_Cancellation|Consequence_Turn_Back|Consequence_Div
ersion|Consequence_Aborted_TakeOff|Consequence_Eng_ShutDown|consequence|
    +-----------------+----------------------+--------------------+----------
----------+--------------------------+----------------------+-----------+
```

ERROR: USE THE HOME TAB TO APPLY TÍTULO 1 TO THE TEXT THAT YOU WANT TO APPEAR HERE.


```
    |              true|                     true|                  true|
true|                        true|                      true|      EIFSD|
    |              true|                     true|                  true|
true|                        true|                     false|       ATO|
    |              true|                     true|                  true|
true|                       false|                     false|        DV|
    |              true|                     true|                  true|
false|                       false|                     false|       IFTB|
    |              true|                     true|                 false|
false|                       false|                     false|         CN|
    |              true|                    false|                 false|
false|                       false|                     false|         DY|
    |             false|                    false|                 false|
false|                       false|                     false|       null|
    |             false|                     true|                 false|
true|                       false|                     false|        DV|
    +------------------+------------------------+--------------------+-----------
----------+----------------------------+------------------------+-----------+
    <BLANKLINE>
    """

    return isers_table.withColumn(
        "consequence",
        F.coalesce(
            F.when(F.col("Consequence_Eng_ShutDown"), "EIFSD"),
            F.when(F.col("Consequence_Aborted_TakeOff"), "ATO"),
            F.when(F.col("Consequence_Diversion"), "DV"),
            F.when(F.col("Consequence_Turn_Back"), "IFTB"),
            F.when(F.col("Consequence_Cancellation"), "CN"),
            F.when(F.col("Consequence_Delay"), "DY"),
        ),
    )


def process_detection_phase(isers_table: DataFrame) -> DataFrame:
    """
    This function catches the text value introduced in the ISER file when the
detection phase is marked as "Other".

    :param isers_table: DataFrame containing the information about the phase in
which the event was detected
    :return: A dataframe with a new column called "detection_phase" containing
the phase value marked in the ISER. If the category Other is marked, then the
function keeps the free text value provided

    >>> spark = getfixture('spark')
    >>> isers_table = spark.createDataFrame([
    ...     Row(Detect_Phase="phase_1", Detect_Phase_Other=None),
```

```
    ...     Row(Detect_Phase="Other", Detect_Phase_Other="Other text"),
    ...     Row(Detect_Phase="phase_2", Detect_Phase_Other="No should appear"),
    ...    ])
    >>> process_detection_phase(isers_table).show(truncate=False)
    +------------+------------------+---------------+
    |Detect_Phase|Detect_Phase_Other|detection_phase|
    +------------+------------------+---------------+
    |phase_1     |null              |phase_1        |
    |Other       |Other text        |Other text     |
    |phase_2     |No should appear  |phase_2        |
    +------------+------------------+---------------+
    <BLANKLINE>
    """

    return isers_table.withColumn(
        "detection_phase",
        F.when(F.col("Detect_Phase") == "Other",
F.col("Detect_Phase_Other")).otherwise(F.col("Detect_Phase")),
    )
```

9.8 GENERAL.PY

Funcionalidad: Contiene funciones auxiliares usadas en `isers.py`.

```
from collections.abc import Callable
from functools import reduce
from typing import Any, TypeVar

from pyspark.sql import SparkSession

T1 = TypeVar("T1")
T2 = TypeVar("T2")
T3 = TypeVar("T3")


def compose(f: Callable[[T1], T2], g: Callable[[T2], T3]) -> Callable[[T1], T3]:
    """Compose two callable objects
    >>> from functools import partial
    >>> def f(x: float, precision: int) -> float:
    ...     return round(x, precision)
    >>> def g(x: float) -> str:
    ...     return str(x)
    >>> h = compose(partial(f, precision=0), g)
```

```
    >>> h(3.14159)
    '3.0'
    """

    def composed(x: T1) -> T3:
        return g(f(x))

    return composed


def apply_functions_sequentially(obj, funcs: list[Callable]) -> Any:
    """Composes a list of functions and applys them to obj
    >>> def add_one(x: int) -> int:
    ...     return x+1
    >>> def sub_one(x: int) -> int:
    ...     return x-1
    >>> apply_functions_sequentially(
    ...     0,
    ...     [
    ...         add_one,
    ...         sub_one,
    ...         add_one,
    ...     ]
    ... )
    1
    """
    return reduce(compose, funcs)(obj)
```