# **Documentación del Pipeline de Procesamiento de Datos**

**Objetivo del Documento**: Proveer una descripción clara y concisa de las decisiones tomadas en el diseño y la implementación del pipeline de procesamiento de datos, incluyendo las justificaciones detrás de cada elección. Este documento está diseñado para ser comprensible para personas con o sin un conocimiento técnico profundo.

## **1. Estructura General del Pipeline**

El pipeline fue estructurado para:

* Cargar datos desde múltiples fuentes.
* Procesar y expandir ciertos campos complejos (como event\_data).
* Agregar métricas específicas de negocio (indicadores de clics, vistas y pagos acumulativos).
* Optimizar y exportar el dataset final en distintos formatos (Parquet, JSON, CSV).

Cada fase fue implementada en un módulo separado para mejorar la legibilidad, la mantenibilidad y facilitar futuras expansiones.

## **2. Carga de Datos y Procesamiento Paralelo**

### **Decisiones Tomadas**

* **Uso de concurrent.futures** para cargar los datos en paralelo. Esto permite manejar la carga de múltiples archivos de manera simultánea, optimizando los tiempos de espera en la fase de carga.
* **Separación de clases de carga** (PrintsLoader, TapsLoader, PaysLoader): Cada fuente de datos tiene un módulo de carga específico que encapsula la lógica particular de transformación y limpieza. Esto asegura que los datos estén estandarizados desde el comienzo y permite una fácil modificación de la lógica para cada fuente en el futuro.

### **Justificación**

La carga paralela es crucial en entornos con múltiples fuentes de datos para aprovechar al máximo el hardware disponible y evitar esperas innecesarias. La organización modular permite que el pipeline sea flexible y escalable.

## **3. Expansión y Transformación de Datos**

### **Decisiones Tomadas**

* **Expansión de event\_data**: Se aplicó pd.json\_normalize para expandir columnas de datos JSON en estructuras tabulares. Esto facilitó el análisis y procesamiento de los datos en etapas posteriores.
* **Manejo de columnas faltantes**: Al expandir event\_data, se añadió una lógica para manejar casos donde la columna puede no estar presente, con advertencias en el log.

### **Justificación**

Al estandarizar los datos de event\_data en una estructura tabular, podemos aplicar métodos de agrupación y análisis más eficientes y claros. El manejo cuidadoso de las columnas faltantes asegura que el pipeline no falle ante datos incompletos.

## **4. Agregación de Métricas Utilizando Dask**

### **Decisiones Tomadas**

* **Uso de Dask**: Para procesar grandes volúmenes de datos de forma paralela, se utilizó Dask en la clase MetricsAggregator, lo que permite distribuir el procesamiento de tareas pesadas en múltiples núcleos de la CPU.
* **Implementación de Dask en cálculos específicos**:
  + calculate\_click\_indicator: Se usa para indicar si un usuario ha hecho clic, basándose en la presencia de datos en la columna day\_taps.
  + calculate\_view\_counts: Calcula la cantidad de vistas en un intervalo de tiempo específico.
  + calculate\_click\_counts\_per\_value\_prop: Calcula la cantidad de clics por cada valor de la propiedad (value\_prop).
  + calculate\_cumulative\_payment\_amounts: Agrega los pagos acumulados en un período específico.

### **Justificación**

Dask permite manejar datos que superan la capacidad de memoria RAM, ya que procesa las operaciones en trozos más pequeños en paralelo. Esto optimiza el tiempo de procesamiento en comparación con pandas y evita cuellos de botella, especialmente en entornos con datos grandes.

## **5. Optimización del Dataset y Exportación**

### **Decisiones Tomadas**

* **Optimización de datos nulos y conversiones**: Se eliminan los valores nulos, se convierte clicked a un formato binario, y se eliminan filas irrelevantes donde tanto monto\_total\_pagos como clicked son cero.
* **Exportación en formato Parquet y JSON**: Se optó por Parquet como formato principal por su eficiencia en la compresión y su rapidez en lectura/escritura, especialmente en comparación con JSON. JSON se exporta para facilitar su lectura en sistemas que no soporten Parquet.

### **Justificación**

Parquet es un formato ideal para almacenamiento y análisis de grandes volúmenes de datos gracias a su estructura compacta y columnar. Esto resulta en tiempos de exportación rápidos (13 segundos) en comparación con JSON (20 segundos). El uso de JSON sigue siendo relevante para sistemas que necesiten datos en un formato de texto.

## **6. Validación de Esquemas y Log de Advertencias**

### **Decisiones Tomadas**

* **Validación de columnas**: Se implementa una validación de esquemas para asegurar que las columnas críticas están presentes antes de cada operación de cálculo. En caso de que falten columnas, se genera un error.
* **Mensajes de advertencia y log**: Los logs indican el avance en cada paso del pipeline, y se generan advertencias cuando faltan columnas. Esto permite que el pipeline maneje datos incompletos sin fallar.

### **Justificación**

La validación de esquemas asegura que los cálculos tengan todos los datos necesarios para ser precisos. Los logs detallados ayudan a diagnosticar y solucionar problemas rápidamente, además de facilitar el monitoreo de cada etapa del proceso.

**Resumen**: El pipeline está diseñado para optimizar tanto el tiempo de procesamiento como la calidad de los datos, teniendo en cuenta las limitaciones del hardware disponible (MacBook M1 con 8GB RAM). Cada decisión fue tomada para lograr un equilibrio entre eficiencia, modularidad y facilidad de mantenimiento, asegurando que el pipeline sea escalable y pueda adaptarse a futuros cambios en los volúmenes de datos o en los formatos de entrada.