

YONILIMAN GALVIS AGUIRRE, JAVIER ALEJANDRO VERGARA ZORRILLA

EXTRACCION, TRANSFORMACIÓN Y CARGA DE DATASET DE PROCESO POR LOTES PARA ENTRENAMIENTO DE MODELO DE PREDICCIÓN POR APRENDIZAJE AUTOMÁTICO

(Marzo 14)

RESUMEN -Los datos obtenidos de procesos de fabricación por lotes son datasets con una estructura compleja. Por esta razón, es necesario preparar adecuadamente los datos para el posterior análisis del proceso y su uso en los procesos de inteligencia artificial.

En los procesos donde se fabrican múltiples tipos de producto, garantizar la calidad final requiere constantes pruebas de calidad que exigen tiempo de producción, costes de mano de obra y el uso de reactivos químicos que producen residuos. Estos residuos a menudo requieren un costoso tratamiento antes de que puedan liberarse al medio ambiente y cumplir las metas de responsabilidad social y medioambiental. La implementación de procesos de Extracción, Transformación y Carga (ETL) es crucial para la preparación de estos datos complejos. ETL permite la integración y limpieza de datos, asegurando que estén en un formato adecuado para el análisis posterior y el desarrollo de modelos de IA y aprendizaje automático.

Utilizar la inteligencia artificial y el aprendizaje automático para desarrollar nuevos conjuntos de herramientas puede mejorar los procesos de fabricación tradicionales, reduciendo costes, horas de mano de obra y el impacto ambiental y de huella de carbono y de esta manera poder aspirar a procesos de cero residuos, optimizando la eficiencia y sostenibilidad de la producción de forma continua.

KEYWORDS – ETL Extracción Transformación Carga Base de Datos Ciencia de Datos Aprendizaje automático Producción por lotes Ingeniería Aplicación de técnicas de ingeniería Inteligencia artificial IA Procesos de fabricación por lotes Calidad del producto Pruebas de calidad Optimización de procesos Reducción de costes Impacto ambiental Cero residuos Integración de datos Limpieza de datos Sostenibilidad

https://github.com/yalves/ETL_project.git

I. INTRODUCCIÓN

La gestión de la calidad en la producción industrial por lotes es esencial para garantizar a los clientes que el producto que adquieren cumple los requisitos necesarios para desempeñar su función, al tiempo que proporciona una adecuada protección de la salud, protege activamente el medio ambiente y mantiene un precio justo. Actualmente, el control de

procesos por lotes está regulado por las normas ISA-88 y la interacción con los sistemas empresariales está regulada por las normas ISA-95. También podemos encontrar propuestas de modelado de sistemas de control distribuido para procesos por lotes basados en la norma IEC-61499, complementando el sistema para prepararlos para la implementación de sistemas abiertos para aplicaciones de inteligencia artificial (IA) y aprendizaje automático.

En 2020, la comunidad europea presentó el informe final oficial Predictive Sensor Data Mining for Product Quality Improvement (PRESED RFCS) como guía para preparar al sector de la producción de acero en la adaptación de herramientas de minería de datos e inteligencia artificial para ayudar a mejorar los procesos de producción y la calidad final del producto. Esta guía puede marcar el inicio del camino a seguir para la producción por lotes. El creciente interés por la aplicación rápida y correcta de tecnologías innovadoras en la producción se basa en la necesidad de desarrollar sistemas de producción más limpios, reducir los costes de producción y minimizar el impacto de los productos en el medio ambiente, cumpliendo así los compromisos sociales de las empresas.

La implementación de procesos de Extracción, Transformación y Carga (ETL) es crucial en este contexto. ETL permite la integración y limpieza de datos complejos obtenidos de los procesos de fabricación, asegurando que estén en un formato adecuado para el análisis y el desarrollo de modelos de IA y aprendizaje automático. Sin embargo, la adquisición y normalización de datos puede ser un desafío significativo debido a la diversidad y complejidad de los datasets. Entender los datos y su estructura es fundamental para optimizar y limpiar los conjuntos de datos, lo que impacta directamente en la capacidad de análisis de los productos y en la aplicación de nuevas tecnologías como la inteligencia artificial.

Cuando se termina un lote de producción, es necesario tomar muestras del producto, gestionar adecuadamente estas muestras y trasladarlas a un lugar adecuado donde se puedan

realizar las pruebas de laboratorio necesarias para demostrar el estado del producto. El producto y el equipo de fabricación se conservarán mientras duren las pruebas y, una vez finalizadas éstas, y en función de los resultados, se tomará una decisión sobre la eliminación del producto.

Si el producto es óptimo, continuará el proceso para formar el producto final. Si el producto no es óptimo, pero puede ser reprocesado para mejorar la calidad, se hará así. Si el producto debe ser desechado porque no es óptimo y no hay posibilidad de mejorarlo, se eliminará. Las pruebas de laboratorio requieren la introducción de distintos tipos de muestras de productos, reactivos y material de laboratorio, así como horas de trabajo para realizar las pruebas. Al final de las pruebas, las muestras de producto contaminadas por reactivos y restos no utilizados, y el equipo de laboratorio usado deben ser desinfectados. Todo este proceso genera residuos que deben eliminarse adecuadamente, añadiendo costes al producto para cubrir el tratamiento adecuado de los residuos y las horas de trabajo empleadas entre el muestreo, las pruebas de laboratorio y los sistemas de gestión de datos para registrar y garantizar la trazabilidad del producto.

Un módulo de machine learning que sea capaz de reconocer las variables predictoras de las pruebas de calidad puede predecir con antelación los resultados de laboratorio y, al enfatizar aquellos que pueden ser decisivos en la liberación del producto, puede lograr una disminución significativa en la frecuencia de dichas pruebas, reduciéndolas a las necesarias para validar y corregir el correcto funcionamiento del módulo. Esto causará el impacto positivo deseado en la producción. Los resultados de la predicción también pueden ser oportunos una vez cumplidas las variables predictoras y sin esperar al final del lote, minimizando el número de lotes rechazados o insalvables.

Todo lo anterior reducirá positivamente el impacto sobre el medio ambiente, reducirá el coste de producción, aumentará los beneficios o mejorará la competitividad del producto y permitirá que la cantidad de horas de trabajo dedicadas al control de calidad se empleen en diseñar nuevos productos o mejorar los existentes.

II. ENTENDIENDO LOS PROCESOS

A. Tren De Produccion

Desde los años 90, la norma ANSI/ISA 88 ha sido la base para desarrollar procesos y piezas de software de control para la fabricación por lotes. Esta norma ha permitido la creación de procesos estandarizados que comparten similitudes profundas, independientemente del producto fabricado, el país donde se ubique la empresa, la disparidad en los equipos industriales o los conceptos empresariales propios de la industria.

Una ventaja significativa de la ANSI/ISA 88 es la organización de los registros obtenidos del proceso. Al cumplir con este estándar internacional, los registros comparten conceptos, grupos de variables y divisiones,

creando un campo fértil para la implementación de sistemas de inteligencia artificial (IA) que pueden ayudar a la producción, ser escalables y propagarse adecuadamente en diversas industrias.

Para entender este papel, es necesario comprender el concepto de tren de producción o unidad. Según el estándar, una unidad es el conjunto de equipos de procesamiento y control necesarios para desarrollar actividades mayores de proceso, operando relativamente independientes unas de otras.

Imagen 1: Tren de Producción

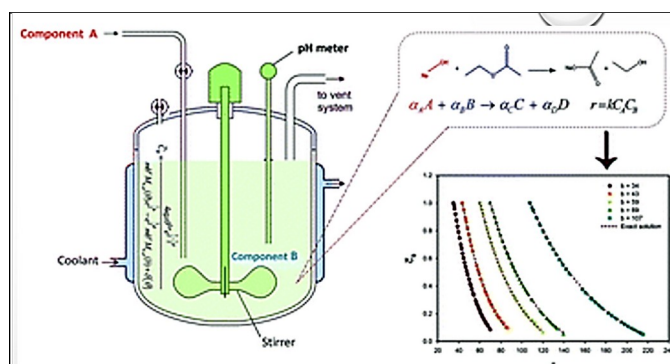


Image source: <https://www.x-mol.net/paper/article/5795495>

En resumen, el tren de producción es el conjunto de equipos necesarios para fabricar un lote de producto de forma total e independiente. Aunque puede haber equipos compartidos, estos solo trabajan para fabricar un lote a la vez. Estos equipos incluyen tanques, motores, bombas, válvulas, tramos de tubería, sensores, transmisores, básculas, tolvas, intercambiadores de calor, entre otros.

B. Explicación De Los Datos De Proceso

Los datos usados en este paper corresponden a 14876 lotes producidos por 8 trenes de producción (a,b,c,d,e,f,g,h), los cuales pueden producir 4 familias de producto y de los cuales existen 51 variantes de producto, para un total de 236530 registros que corresponden a 516 días de producción.

Columns	Details	Total
Datetime	Datetime of production start	516 days
ID	Register ID	14876
Prod	Identification of manufactured products	51
Type	Product Family group	4
Train	Manufacturing Batch Reactor and devices instances group	8
Unit	Unit of process used	3
Phase_ID	Components and Phases	78
EU	Engineering units	7
Value	Value as a percentage with respect to the adjusted value of the real value (additions, wait times, speeds, temperatures, pressures, etc.)	
Check	Regular pH verification (yes or no?)	
Total	All transactions logged in CSV	236530

Tabla 1: Descripción de DataSet

Hay 3 tipos de equipos diferentes:

- **tmx**: Tanque mezclador ó reactor, este es el tanque principal, de gran capacidad donde se realizan los procesos químicos de fabricación.
- **pmx**: Tanque que hace premezclas entre varios productos antes de descargar al mezclador principal
- **pwt**: Tanque que se usa para pesar producto en cantidades pequeñas antes de descargarlo al tanque mezclador.

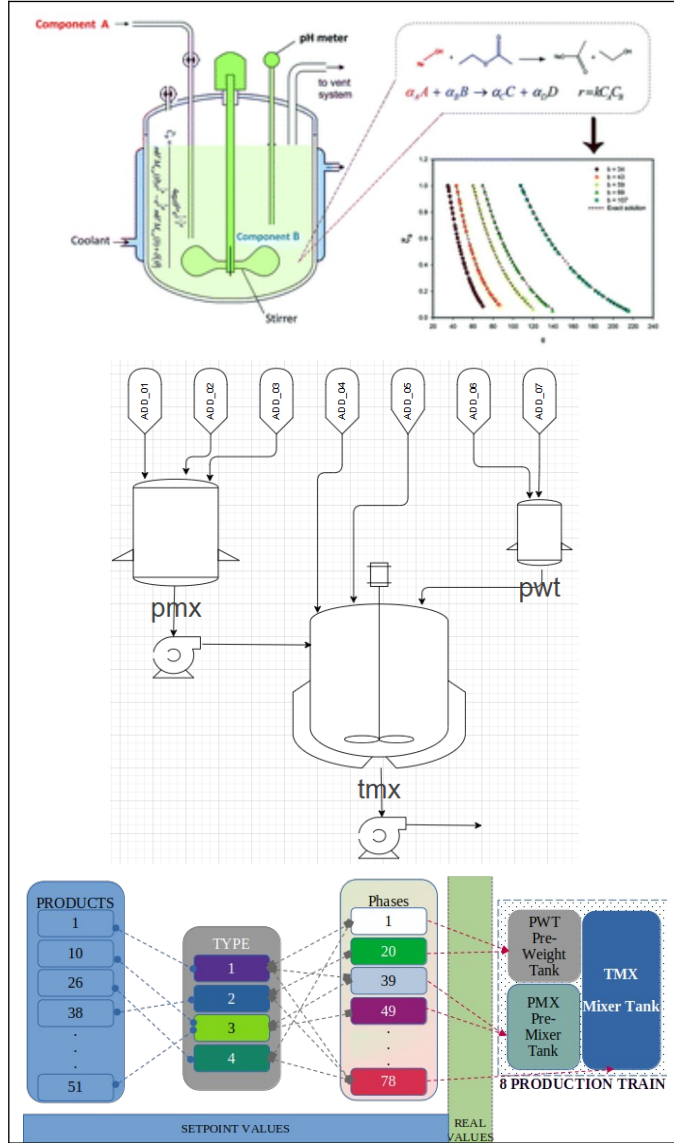


Tabla 2: Tren de fabricación

Ahora no todos los 8 trenes son iguales, puede tener 3 unidades (pmx, pwt, pmx) ó sólo 2 unidades (pwt, pmx). Esta diferencia causa que la capacidad de producir variantes también difiera entre los 2 tipos de trenes.

Resumen de métricas por Tren de producción:

	Train	Unit	Tipos	Cantidad_Productos	Total_lotes
0	a	3	[x, f, s]	34	1916
1	b	3	[x, f, s]	34	1883
2	c	3	[f, x, s]	38	1900
3	d	3	[f, x, s]	37	1871
4	e	3	[f, b, s]	35	1845
5	f	3	[f, b, s]	35	1910
6	g	2	[s]	15	1782
7	h	2	[f, s]	13	1750

Tabla 3: Productos por tren, unidades y tipo.

Las fases (phases) son los eventos en los cuales se hacen algún tipo de acción sobre el proceso, adiciones, cambios de velocidad de motores y/o bombas, esperas por estabilización, por cambios de temperatura o por una medición

EU_ID	EU_name	Description	type
0	seg	EU_0	waittime phase
1	ph	EU_1	acidity control
2	kg/l	EU_2	density control
3	kg	EU_3	weight phase
4	cp	EU_4	viscosity control
5	c	EU_5	temperature phase
6		EU_6	no assigned phase

Tabla 4: Tipos de fases y unidades de medición.

Los valores medidos durante las fases de adición, como la temperatura, las adiciones por peso, los tiempos para emulsificación y otros parámetros no asignados, son los predictores del proceso. Estas variables, que incluyen segundos (seg), kilogramos (kg), grados Celsius (°C) y valores no asignados (NaN) a una unidad específica, se registran durante las fases de adición, agitación, tiempo o velocidades del proceso. Los valores medidos se comparan con los valores objetivo y se expresan como porcentajes de cumplimiento.:

Equation 1: Predictores

$$X = \text{Predictor} = \% \text{Cumplimiento} = \frac{\text{Valor Real}}{\text{Valor Objetivo}}$$

Los valores de salida del sistema corresponden a las variables dependientes y corresponden a los controles de calidad realizados al producto, los cuales son factores determinantes para liberar o retener el producto. Estos valores incluyen el pH, la viscosidad y la acidez, medidos en unidades como pH, kilogramos por litro (kg/l) y centipoises (Cp).

C. Análisis Exploratorio De Los Datos.

Conocer los datos es fundamental para comprender el comportamiento del modelo, ya que estos datos reflejan el comportamiento del proceso de producción. Esto nos permite enfocar la atención de manera adecuada para generar un impacto más positivo utilizando una cantidad de recursos más

razonable.

En el contexto de Extracción, Transformación y Carga (ETL), es crucial entender que la calidad y preparación de los datos son esenciales para el éxito de cualquier proyecto de machine learning. Aunque es tentador desarrollar un módulo de machine learning que satisfaga todas las necesidades, esto puede ser un reto innecesario. El impacto en la producción podría ser mínimo, la cantidad de datos podría ser insuficiente para obtener un buen rendimiento, y los recursos necesarios para la programación y ejecución podrían ser demasiado elevados en comparación con los posibles resultados obtenidos.

El proceso de ETL permite extraer datos de diversas fuentes, transformarlos para asegurar su calidad y coherencia, y cargarlos en un sistema de almacenamiento adecuado para su análisis. Lo más importante es conocer los datos e identificar el foco. Cuando los datos se extraen de la base de datos, no siempre es evidente dónde debe centrarse la atención. Además, el deseo de los usuarios puede superar las capacidades de cualquier sistema, ya que tendemos a sobreestimar lo que la inteligencia artificial puede lograr, imaginando que puede realizar tareas que ni siquiera un ser humano con conocimientos extraordinarios podría llevar a cabo.

En resumen, un enfoque efectivo en ETL garantiza que los datos estén bien preparados y optimizados, lo que maximiza el potencial de los modelos de machine learning y minimiza el uso innecesario de recursos.

D. Observaciones De Todos Los Datos:

Crear Gráficos e histogramas es fundamental para entender los datos:

Imagen 2: Cantidad de Lotes producidos por cada tren por tipo de producto

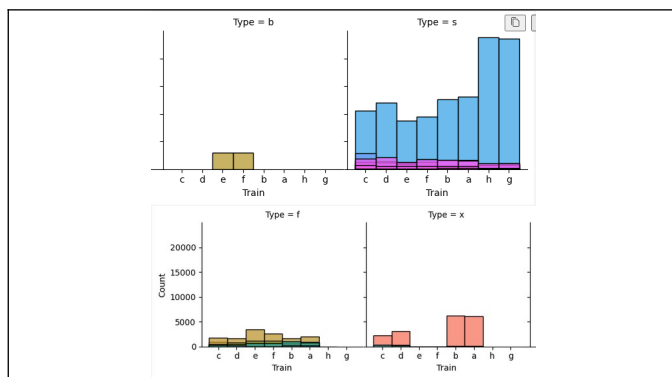


Imagen 3: Cantidad de Lotes producidos por producto y Tipo

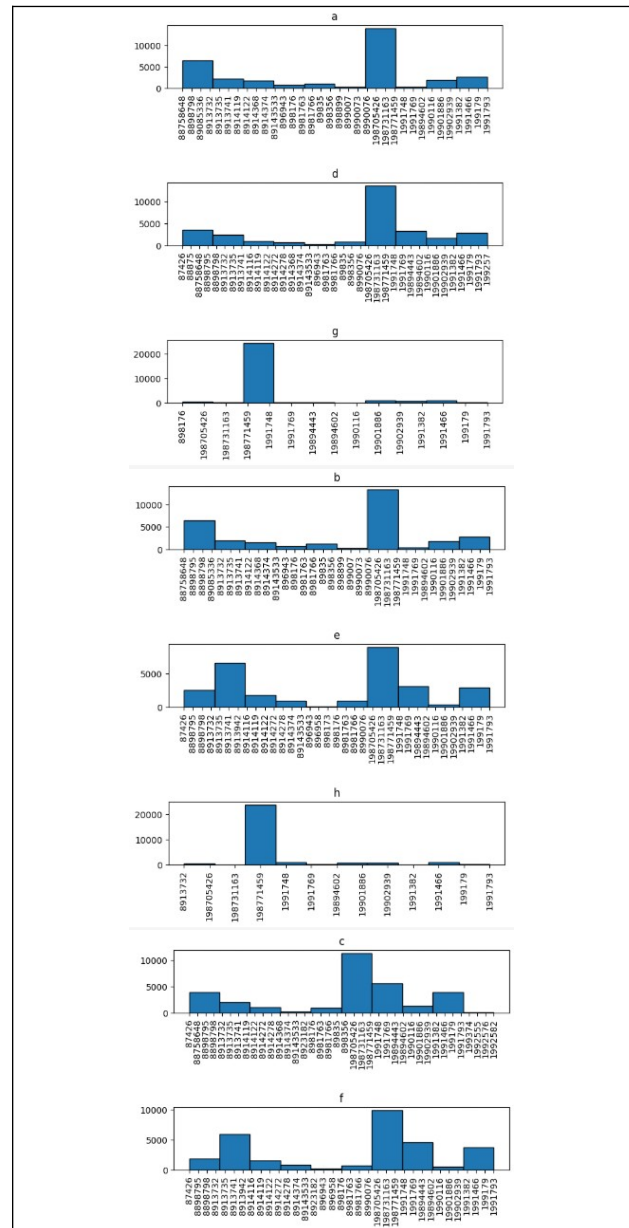
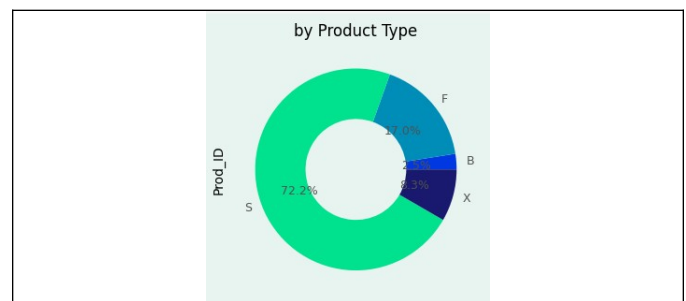


Imagen 4: Pastel de Familias de Productos



Aunque las gráficas dicen mucho, crear reportes de texto también ayuda a focalizarse de forma correcta para evitar usar recursos buscando resultados unicornios:

Resumen de métricas por Tipo de producto:						Unit					
Type	Total_lotes	Trenes				pmx	20	pwt	18	tmx	47
0	b	338	[e, f]								
1	f	2948	[c, d, e, f, b, a, h]								
2	s	10509	[g, a, b, c, d, e, f, h]								
3	x	1062	[d, a, b, c]								
EU						Group Products by Train					
50						Train					
20						E 28059					
23						H 28061					
50						G 28550					
13						F 29790					
27						D 29933					
50						C 30188					
Name: Prod_ID, dtype: int64						B 30822					
Group Products by Type						A 31126					
Type						Name: Prod_ID, dtype: int64					
B 5888											
X 19708											
F 40273											
S 179660											
Name: Prod_ID, dtype: int64											
Predictores						Dependientes					
	B	F	S	X	Grand Total		B	F	S	X	Grand Total
%	1291	9855	31348	2967	45461	cP			10445	816	11261
C	335		10455		10790	kg/l		379		797	1176
kg	3306	18616	105669	12157	139748	pH		286	2901		4003
Seg	670	8522	12743	2155	24090						
Grand Total	5602	36993	160215	17279	220089	Grand Total	286	3280	10445	2429	16440
Predictores						Dependientes					
	B	F	S	X	Grand Total		B	F	S	X	Grand Total
%	0.59%	4.48%	14.24%	1.35%	20.66%	cP	0.00%	0.00%	63.53%	4.96%	68.50%
C	0.15%	0.00%	4.75%	0.00%	4.90%	kg/l	0.00%	2.31%	0.00%	4.85%	7.15%
kg	1.50%	8.46%	48.01%	5.52%	63.50%	pH	1.74%	17.65%	0.00%	4.96%	24.35%
Seg	0.30%	3.87%	5.79%	0.98%	10.95%						
Grand Total	2.55%	16.81%	72.80%	7.85%	100.00%	Grand Total	1.74%	19.95%	63.53%	14.77%	100.00%

Tabla 5: Análisis numérico de producto por cantidades

Observando las gráficas y las tablas presentadas podemos evidenciar que:

1. La familia de productos tipo 'S' es la más producida con una proporción de 70.7% de los registros. (10509 lotes)
2. La segunda familia de productos más producidos es el tipo 'F' con el 19.8% (2948 lotes).
3. Para el dataset, hay pocos registros (1062 Lotes) de la familia de productos 'X' 7.1%.
4. La familia de productos b es una variante muy poco producida con solo 338 Lotes 0.23%
5. La prueba de calidad de viscosidad sólo es realizada para productos del grupo tipo 'S' y 'X'.
6. La prueba de calidad 'Densidad' sólo es realizada para productos del grupo tipo 'F' y 'X'.
7. La prueba de calidad 'PH' sólo es realizada para productos del grupo tipo 'B', 'F' y 'X'.
8. Basados en las observaciones anteriormente detalladas podemos hacer una primera conclusión:

La familia de productos críticos es el 'S' ya que es el grupo de más alta producción, se fabrica en los 8 trenes y nos ofrece una cantidad de datos mayor y la oportunidad de realizar in vivo un mayor número de pruebas, ajustes en el menor tiempo posible.

Si el módulo de ML desarrollado tiene resultados óptimos el impacto y beneficios serán maximizados si está enfocado al producto tipo 'S', esto reduce las dimensiones de los predictores y simplifica el modelo de ML.

Por tanto la salida del modelo propuesta sería de viscosidad o Centipoises.

Hito 1: Selección de grupo de producto

III. DATASET

El dataset es un archivo separado por comas, csv el cual ha sido pre-procesado para ocultar información sensible y consta de las siguientes columnas:

Datetime	Fecha del registro
ID	ID de Lote de producción único
Prod_ID	ID del producto fabricado
Type	Tipo de producto
Train	Tren de producción usado para la fabricación
Unit	Tipo de equipos usados para la fase
Phase ID	ID del proceso realizado en la fabricación
EU	Unidades de la muestra
Value	Porcentaje de cumplimiento (Real/Objetivo)
Verify	Esta columna dice si el lote es o no verificado

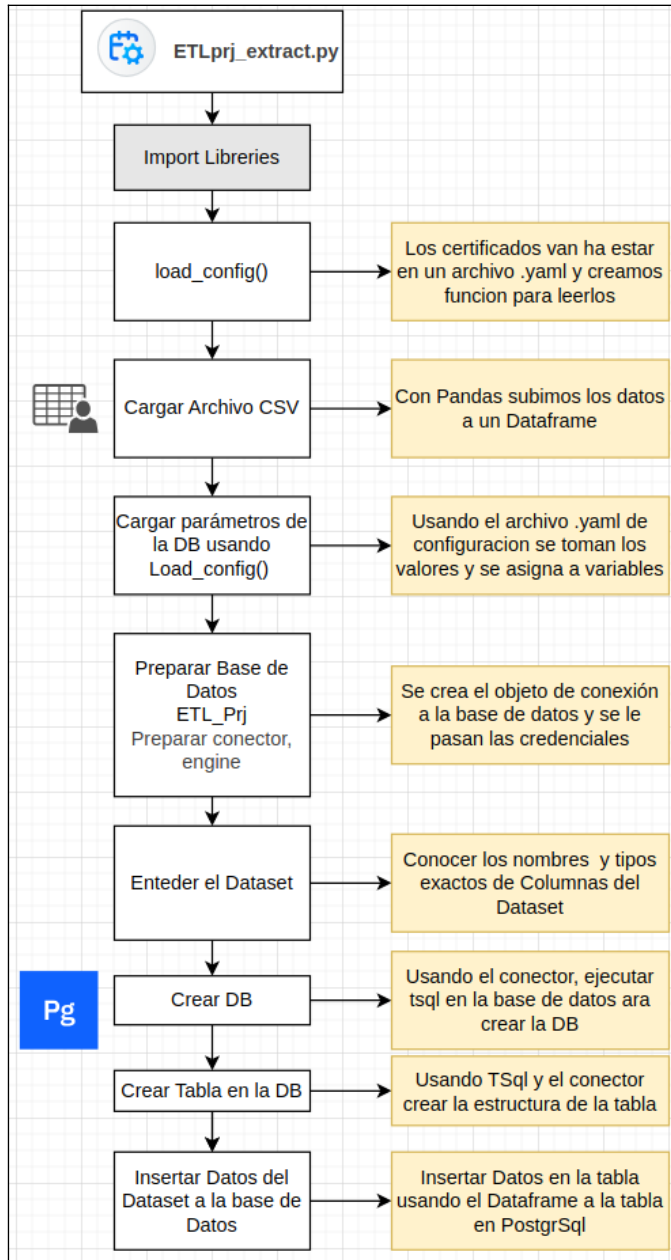
	A	B	C	D	E	F	G	H	I	J
1	DateTime	ID	Prod_ID	Type	Train	Unit	Phase_ID	EU	Value	Verify
2	2/27/2025 7:34:21	BYBAGDDIM	87426 F	C	PMX	MD1M66	kg		1	0
3	2/5/2025 6:57:51	BXWAGDD6N	87426 F	C	PMX	MD1M66	kg		1.003703704	0
4	2/5/2025 2:07:48	9NAGDD6M	87426 F	C	PMX	MD1M66	kg		0.996296296	0
5	2/4/2025 23:04:32	XT7AGDD67	87426 F	C	PMX	MD1M66	kg		1.014814815	0
6	2/4/2025 17:45:15	V9EAGDD66	87426 F	C	PMX	MD1M66	kg		0.97037037	0
7	2/4/2025 15:15:41	GZ7AGDD65	87426 F	C	PMX	MD1M66	kg		0.988888889	0
8	2/4/2025 11:36:37	RQVAGDD65	87426 F	C	PMX	MD1M66	kg		1	0
9	2/3/2025 18:31:57	FLBAGDD5M	87426 F	C	PMX	MD1M66	kg		1.003703704	0
10	2/3/2025 8:31:47	EOPAGDD5K	87426 F	C	PMX	MD1M66	kg		0.996296296	0
11	2/3/2025 18:17:04	AL3AGCCY0	87426 F	C	PMX	MD1M66	kg		1.003703704	0
12	1/20/2025 6:51:28	DKBAGCCXX	87426 F	C	PMX	MD1M66	kg		1.003703704	0
13	1/19/2025 23:07:44	TKIAGCCXH	87426 F	C	PMX	MD1M66	kg		1	0
14	1/19/2025 19:18:16	YDAGCCXH	87426 F	C	PMX	MD1M66	kg		1	0
15	1/19/2025 4:44:40	37QAGCCXE	87426 F	C	PMX	MD1M66	kg		1	0
16	1/19/2025 2:50:53	JZEAGCCXD	87426 F	C	PMX	MD1M66	kg		1	0
17	1/18/2025 17:45:30	XIAGCCWXX	87426 F	C	PMX	MD1M66	kg		1	0
18	10/3/2024 2:01:23	GATAG11DR	87426 F	C	PMX	MD1M66	kg		1	0
19	10/2/2024 20:36:35	KA3AG11DA	87426 F	C	PMX	MD1M66	kg		1	0
20	10/2/2024 17:09:49	9TQAG11DA	87426 F	C	PMX	MD1M66	kg		1	0
21	10/2/2024 5:01:43	15MAG11D7	87426 F	C	PMX	MD1M66	kg		1	0
22	10/1/2024 21:37:25	J96AG11CS	87426 F	C	PMX	MD1M66	kg		1.003703704	0
23	9/1/2024 20:29:01	AIHAG00WF	87426 F	C	PMX	MD1M66	kg		0.988888889	0
24	8/31/2024 16:41:42	0A5AG00VV	87426 F	C	PMX	MD1M66	kg		0.996296296	0
25	8/31/2024 1:54:07	Z0SAG00VT	87426 F	C	PMX	MD1M66	kg		1.007407407	0
26	8/30/2024 1:50:42	SKUAG00V8	87426 F	C	PMX	MD1M66	kg		1	0
27	5/6/2024 18:31:04	NFKAFZZ28	87426 F	C	PMX	MD1M66	kg		1	0
28	5/6/2024 13:40:22	BH1AFZZ27	87426 F	C	PMX	MD1M66	kg		0.992592593	0
29	4/28/2024 7:18:46	6RIAFYYXU	87426 F	C	PMX	MD1M66	kg		1.007407407	0
30	4/27/2024 22:44:11	EIQAFYYXD	87426 F	C	PMX	MD1M66	kg		1.007407407	0
31	4/27/2024 14:45:58	QI6AFYYXB	87426 F	C	PMX	MD1M66	kg		0.988888889	0
32	4/27/2024 2:33:26	U13AFYYX9	87426 F	C	PMX	MD1M66	kg		1.003703704	0
33	4/26/2024 18:14:11	PZNAFYOWT	87426 F	C	PMX	MD1M66	kg		0.996296296	0
34	11/2/2023 23:31:56	PW7AFLLAM	87426 F	C	PMX	MD1M66	kg		1.003703704	0
35	10/12/2023 13:51:13	7E1AFKKZ6	87426 F	C	PMX	MD1M66	kg		1	0
36	10/12/2023 9:39:44	HWBAFKKZ5	87426 F	C	PMX	MD1M66	kg		1.003703704	0
37	9/27/2023 6:58:57	JUTAFKKQZ	87426 F	C	PMX	MD1M66	kg		1	0
38	9/26/2023 22:08:54	68GAFKKQI	87426 F	C	PMX	MD1M66	kg		1	0
39	9/26/2023 17:43:33	HD5AFKKQH	87426 F	C	PMX	MD1M66	kg		1	0
40	9/26/2023 7:31:56	9EDAFKKQF	87426 F	C	PMX	MD1M66	kg		1.018518519	0
41	9/26/2023 0:49:31	1CRAFKKQE	87426 F	C	PMX	MD1M66	kg		1	0

IV. PREPARACION DEL AMBIENTE

Se va a trabajar con:

- IDE: VsCode
- Entornos de python: pyenv
- manejador de versiones de paquetes: Poetry
- Base de Datos: Posgresql
- Conector Base de Datos: pycpg2

V. EXTRACCIÓN



- Definir funcion Load_Config() con la cual vamos a leer el archivo donde estan las credenciales (.yaml)
- Cargar Archivo CSV en un Dataframe usando Pandas
- Cargar los parámetros del .yaml usando Load_config
- Creación de Objeto de comunicacion y la tarea de conexion con la base de datos (engine)
- Realizar un estudio del Dataset para conocer la estructura y el tipo de datos
- usando el engine ejecutar Tsql para crear base de datos para la extraccion
- Usando la información del Dataset y con el engine ejecutar Tsql el query de creacion de la DB
- Insertar los datos del dataframe al sql

Vamos a extraer los datos de la fuente, en este caso un CSV y los vamos a llevar a una base de datos PostgreSQL do python y la el conector para sql .

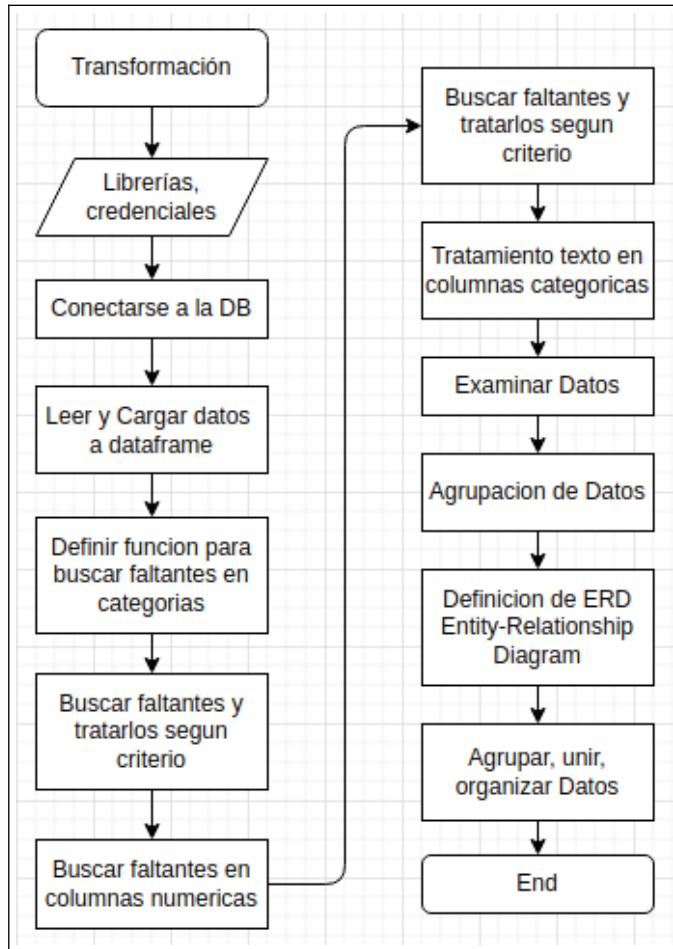
- Crear y habilitar entorno virtual.
- Iniciar Poetry
- Crear archivo .yaml para colocar las credenciales fuera del código fuente con la siguiente información:

```

database:
  user:      "user de la db"
  password:  "*****"
  host:      "ip ó nombre del host de la DB"
  port:      "puerto de com de la DB"
  name:      "nombre asignado"
  
```

- Crear archivo python .py
- Importar las librerías

VI. TRANSFORMACION



1. **Carga de librerías** necesarias para el analisis de los datos
2. **Cargar parámetros de la DB:** Usando el archivo de config.yam, tomamos los parametros de configuracion, los asigna a etiquetas y los carga en variables internas
3. **Asignando las variables:** se crea el objeto de conexion con la base de datos, Posgresql

para acceder a la base de datos debe ingresar como super user, el ingresará con el usuario del sistema y el password es el mismo del sistema:

\$ sudo su – postgres

4. **Crear conexión** a la base de datos
5. **Leer y Cargar Datos:** Leemos los datos desde la base de datos y los mostramos.
6. **Cargamos los datos a un Dataframe** haciendo una consulta simple

7. **Análisis de errores:** Buscamos errores en los datos, como la fuente de los datos puede tener errores, vamos a buscar errores y tomar las decisiones necesarias para subsanarlos.
8. **columnas categóricas con errores:** Una de las situaciones mas complejas es tener columnas categóricas con fallas ya que estas generan datos huérfanos, las columnas con valores pueden procesarse para generar un valor coherente, pero las categóricas es más complejo ya que no se puede generar una categoría aleatoria, se deben usar las columnas hermanas para generar la categoría correcta, sin embargo si se considera que la cantidad de datos es suficiente y la cantidad de errores es muy pequeña podemos descartar las filas con fallas categóricas y ahorrar trabajo evitando problemas de mala recategorización posterior, en especial que una vez se recategorizan en un dataset grande se pierden de vista las filas editadas y si están mal categorizadas pueden causar sesgos o fallas de la fuente cuando estas se usan en el entrenamiento de IA.
9. **Tratamiento de Errores Categóricos:** Como hemos observado, en nuestro Dataset tenemos 14876 lotes de los cuales 19 tienen errores en la categoría de ID de producto, eso significa que no conocemos que producto fue el fabricado, esto corresponde al 0.13% del dataset lo cual es una cantidad que podemos considerar bastante pequeña.

Ya que el dataset es suficientemente grande podemos eliminar estas filas sin necesidad de afectar la data y en cambio intentar categorizar estos productos puede causar pérdida de integridad del dataset, procedemos a eliminarlo

10. **Tratamiento de Errores no Categóricos:** Igual que las categorías las columnas de valores pueden tener valores nulos o faltantes, vamos a usar la función que creamos para buscar los IDs con valores faltantes
11. **Verificación de las faltantes:** Verificamos una última vez para observar si el dataset ya está limpio
12. **Tratamiento de texto en columnas categoricas:** Uno de los problemas más difíciles de identificar es cuando valores en las columnas categóricas tienen errores de tipografía, por ejemplo algún espacio antes o después de la categoría ó contengan mayúsculas, siempre es buena idea asegurarse que estas columnas estén normalizadas, así que vamos a asegurarnos que no tengan espacios adicionales, puntuación al final y estén en minúsculas.
13. **Exámen de los datos:** Para entender los datos tenemos que examinarlos, para ello los vamos a agrupar y graficar de forma lógica y Primero vamos a crear un resumen de que compone el dataset:

14. **Análisis de la cantidad** Distribución de Productos (Product_ID) agrupada por Tren de producción (Train).

15. Usando plots representamos la frecuencia de los Productos (por ID) en cada tren de productos por tipo y producción por cada tren:

Análisis:

- * Podemos observar que el tipo de producto más fabricado es \$\$ y se fabrica en todos los trenes.
- * El producto tipo *f* se produce en menor medida y no se fabrica en los trenes \$h,g\$
- * El producto tipo \$x\$ sólo se fabrica en los trenes \$ (c,d,b,a)\$, no se fabrica en los trenes \$e,f,h,g\$

16. **Distribución de Productos (Product_ID)** agrupada por Tren de producción (Train): Se presentan subplots que representarán la frecuencia de los Productos (por ID) en cada tren

VII. ERD

Se crea un modelo lógico para la ubicación final de los datos organizados, se verifican las necesidades de los datos para determinar como se van ha segementar los datos de tal manera que se realicen solo joins de nivel 1 .

Tabla de Eventos

en esta tabla vamos ha ubicar y expandir las marcas de tiempo para permitir obtener rápidamente la informacion necesaria para el análisis de los datos:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227205 entries, 0 to 227204
Data columns (total 15 columns):
#   Column              Non-Null Count  Dtype
---  -
0   DateTime            227205 non-null  datetime64[ns, UTC]
1   Year                227205 non-null  int32
2   Quarter             227205 non-null  int32
3   Month               227205 non-null  int32
4   MonthName           227205 non-null  object
5   Week                227205 non-null  UInt32
6   WeekDay             227205 non-null  int32
7   DayOfYear           227205 non-null  int32
8   Day                 227205 non-null  int32
9   DayName             227205 non-null  object
10  Hour                227205 non-null  int32
11  Minute              227205 non-null  int32
12  Second              227205 non-null  int32
13  Shift_8H            227205 non-null  int64
14  Formatted_Timestamp 227205 non-null  object
dtypes: UInt32(1), datetime64[ns, UTC](1), int32(9), int64(1), object(3)
memory usage: 17.6+ MB
None
```

Events	Event Log				
Datetime	timestamptz	N-N	default	UTC Tiezone-aware	
Year	smallint	N-N	default	year number	
Quarter	smallint	N-N	default	quarte number	
Month	smallint	N-N	default	month number	
MonthName	text	N-N	default	month name	
Week	smallint	N-N	default	week number	
WeekDay	smallint	N-N	default	day of a week	
DayOfYear	smallint	N-N	default	day of a year (0-365)	
Day	smallint	N-N	default	day of a month	
DayName	text	N-N	default	day name	
shift_8H	smallint	N-N	default	shifts of 8 hours	
Formatted_Timestamp	text	N-N	default	datetime serialized	
LogID	bigserial	N-N	default	auto inc id	
LogID	bigserial	N-N	default	comment	
LogID	bigserial	N-N	default	comment	
LogID	bigserial	N-N	default	comment	

Hay que tener en cuenta que en PostgreSQL el tipo timestamptz es UTC (universal time coordinates) el cual permite tener una referencia de tiempo global, cuando se graba en este registro un dato será formateado de forma automática a UTC y cuando se desea recuperar el valor de tiempo se debe definir en que zona horaria se desea el resultado.

El campo Formatted_timestamp serializa el valor de tiempo en el formato de texto (%Y%m%d%H%M%S')

añomesdiahoraminutosegundo

por ejemplo: si la fecha es 2025-02-28 17:51:04+00:00, el valor de la formatted_timestamp es: 20250228175104

Basados en la fecha del registro original se calculan las columnas adicionales.

En términos de producción se calculan 3 turnos de 8 horas cada uno, esto permite analizar los datos dependiendo del turno de produccion así:

- Turno 1:
 - Inicia a las 6:00:00 H
 - Termina a las 13:59:59 H
- Turno 2:
 - Inicia a las 14:00:00 H
 - Termina a las 21:59:59 H
- Turno 3:
 - Inicia a las 22:00:00 H
 - Termina a las 5:59:59 H del siguiente dia.

Tabla de Lotes

Utiliza la subtabla de Productos, en donde se almacenan los Identificadores de los producto fabricados, se le asigna un etiqueta a cada producto para facilitar el entendimiento en el formato "Product_x" donde x es el valor del id de la tabla.

Tambien utiliza lun diccionario de Trenes donde al igual que lo sproductos se le crea una etiqueta a los trenes en el formato

“Train_x”, en este caso se usa diccionario ya que la cantidad de trenes no es un valor variable al contrario del número de productos que puede cambiar rápidamente según las necesidades del negocio.

Se usa el ID y para mayor entendimiento se renombra a Lot_ID y se renombra train a Train_ID, se preseta el tipo de producto, se adicionan las etiquetas para los trenes y para los productos.

Por cada lote se toma la primera fecha y la última fecha de concurrencia en el Dataset.

La diferencia entre ellas se toma como el tiempo de duración del lote, usualmente los lotes terminan con una fase de poca duración con lo cual se tiene una precisión aceptable en el valor en tiempo de la duración de fabricación de un lote.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14857 entries, 0 to 14856
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   Lot_ID          14857 non-null object
1   Type            14857 non-null object
2   Train_ID        14857 non-null object
3   Train           14857 non-null object
4   Prod_ID         14857 non-null object
5   Product_name    14857 non-null object
6   First_Date      14857 non-null datetime64[ns, UTC]
7   Last_Date       14857 non-null datetime64[ns, UTC]
8   Duration        14857 non-null timedelta64[ns]
9   Formatted_Timestamp 14857 non-null object
dtypes: datetime64[ns, UTC](2), object(7), timedelta64[ns](1)
memory usage: 1.1+ MB
None
```

Lots		Produced Lots			
Lot_ID	VARCHAR(20)	NULL	default	comment	
Type	VARCHAR(2)	NULL	default	comment	
Train_ID	VARCHAR(2)	NULL	default	comment	
Train	VARCHAR(400)	NULL	default	comment	
Prod_ID	VARCHAR(20)	NULL	default	comment	
Prod_Name	VARCHAR(400)	NULL	default	comment	
First_Date	timestampz	NULL	default	comment	
Last_Date	timestampz	NULL	default	comment	
Duration	timetz	NULL	default	comment	
Formatted_Timestamp	text	NULL	default	comment	
LogID	bigserial	N-N	default	comment	
LogID	bigserial	N-N	default	comment	

Tabla de EU

Esta tabla contiene la información de las unidades de medida de las mediciones en el proceso de fabricación, estas permiten determinar si una fase pertenece a un predictor (phase) ó por el contrario es una variable dependiente (control).

Para ello se usan 2 diccionarios para asignar una descripción y un tipo cada una de las unidades, donde se definen las unidades que provienen de fases de fabricación (variables

predictoras) y las que provienen de controles de calidad (variables dependientes).

En este caso también renombramos el encabezado de EU a EU_ID esto igualmente es para facilitar la lectura posterior

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 4 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   EU_ID           7 non-null      object
1   EU_name         7 non-null      object
2   Description     7 non-null      object
3   Type            7 non-null      object
dtypes: object(4)
memory usage: 356.0+ bytes
None
```

EU	comment
EU_ID	varchar(20) NULL default comment
EU_name	varchar(400) NULL default comment
Description	varchar NULL default comment
Type	varchar(400) NULL default comment
LogID	bigserial N-N default comment
LogID	bigserial N-N default comment

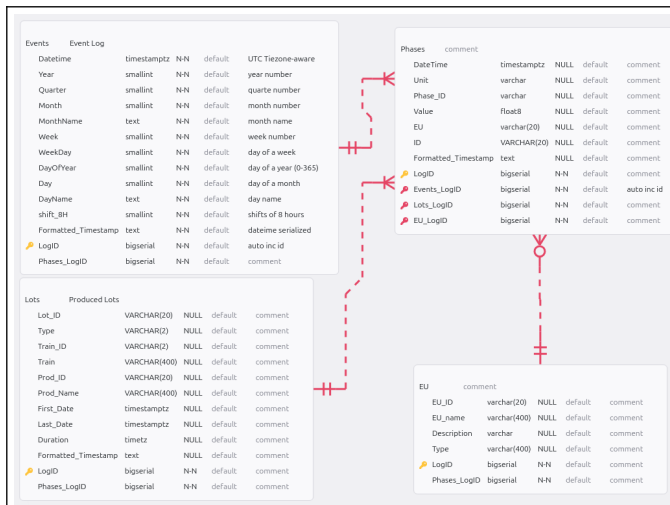
Tabla de Phases.

Esta tabla registra los valores tomados desde la fabricación, y es el que contiene los valores que van a determinar el comportamiento y el desempeño de los trenes de producción.

```
<class 'pandas.core.frame.DataFrame'>
Index: 236176 entries, 79486 to 215775
Data columns (total 7 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   DateTime        236176 non-null datetime64[ns, UTC]
1   Unit            236176 non-null object
2   Phase_ID        236176 non-null object
3   Value           236176 non-null float64
4   EU              236176 non-null object
5   ID              236176 non-null object
6   Formatted_Timestamp 236176 non-null object
dtypes: datetime64[ns, UTC](1), float64(1), object(5)
memory usage: 14.4+ MB
None
```

Phases	comment
DateTime	timestamptz NULL default comment
Unit	varchar NULL default comment
Phase_ID	varchar NULL default comment
Value	float8 NULL default comment
EU	varchar(20) NULL default comment
ID	VARCHAR(20) NULL default comment
Formatted_Timestamp	text NULL default comment
LogID	bigserial N-N default comment
LogID	bigserial N-N default auto inc id
LogID	bigserial N-N default comment
LogID	bigserial N-N default comment
LogID	bigserial N-N default comment

VIII. DIAGRAMA ERD



```

CREATE TABLE EU
(
  EU_ID          varchar(20) ,
  EU_name        varchar(400),
  Description     varchar   ,
  Type           varchar(400),
  LogID          bigserial  NOT NULL,
  Phases_LogID   bigserial  NOT NULL,
  PRIMARY KEY (LogID)
);

COMMENT ON COLUMN EU.EU_ID IS 'Engineering Units ID';
COMMENT ON COLUMN EU.EU_name IS 'Engineering Units label';
COMMENT ON COLUMN EU.Description IS 'Engineering Units desc';
COMMENT ON COLUMN EU.Type IS 'phase type';
COMMENT ON COLUMN EU.LogID IS 'table auto inc id';
COMMENT ON COLUMN EU.Phases_LogID IS 'foreign key phases table';
CREATE TABLE Events
(
  Datetime       timestamptz NOT NULL,
  Year           smallint   NOT NULL,
  Quarter        smallint   NOT NULL,
  Month          smallint   NOT NULL,
  MonthName      text       NOT NULL,
  Week           smallint   NOT NULL,
  WeekDay        smallint   NOT NULL,
  DayOfYear      smallint   NOT NULL,
  Day            smallint   NOT NULL,

```

```

DayName          text       NOT NULL,
shift_8H         smallint   NOT NULL,
Formatted_Timestamp text     NOT NULL,
LogID            bigserial  NOT NULL,
Phases_LogID     bigserial  NOT NULL,
PRIMARY KEY (LogID)
);

COMMENT ON TABLE Events IS 'Event Log';
COMMENT ON COLUMN Events.Datetime IS 'UTC Tiezone-aware';
COMMENT ON COLUMN Events.Year IS 'year number';
COMMENT ON COLUMN Events.Quarter IS 'quarte number';
COMMENT ON COLUMN Events.Month IS 'month number';
COMMENT ON COLUMN Events.MonthName IS 'month name';
COMMENT ON COLUMN Events.Week IS 'week number';
COMMENT ON COLUMN Events.WeekDay IS 'day of a week';
COMMENT ON COLUMN Events.DayOfYear IS 'day of a year (0-365)';
COMMENT ON COLUMN Events.Day IS 'day of a month';
COMMENT ON COLUMN Events.DayName IS 'day name';
COMMENT ON COLUMN Events.shift_8H IS 'shifts of 8 hours';
COMMENT ON COLUMN Events.Formatted_Timestamp IS 'timestamp
serialized';
COMMENT ON COLUMN Events.LogID IS 'table auto inc id';
COMMENT ON COLUMN Events.Phases_LogID IS 'foreign key phases table';

```

```

CREATE TABLE Lots
(
  Lot_ID          VARCHAR(20) ,
  Type            VARCHAR(2)   ,
  Train_ID        VARCHAR(2)   ,
  Train           VARCHAR(400) ,
  Prod_ID         VARCHAR(20)  ,
  Prod_Name       VARCHAR(400) ,
  First_Date      timestamptz ,
  Last_Date       timestamptz ,
  Duration        timetz      ,
  Formatted_Timestamp text     ,
  LogID           bigserial    NOT NULL,
  Phases_LogID    bigserial    NOT NULL,
  PRIMARY KEY (LogID)
);

COMMENT ON TABLE Lots IS 'Produced Lots';
COMMENT ON COLUMN Lots.Lot_ID IS 'Lot unique id';
COMMENT ON COLUMN Lots.Type IS 'type of product';
COMMENT ON COLUMN Lots.Train_ID IS 'train id';
COMMENT ON COLUMN Lots.Train IS 'train label';
COMMENT ON COLUMN Lots.Prod_ID IS 'product id';
COMMENT ON COLUMN Lots.Prod_Name IS 'product label';
COMMENT ON COLUMN Lots.First_Date IS 'first register datetime';
COMMENT ON COLUMN Lots.Last_Date IS 'last register datetime';
COMMENT ON COLUMN Lots.Duration IS 'difference between first and
last datetime';
COMMENT ON COLUMN Lots.Formatted_Timestamp IS 'timestamp
serialized';
COMMENT ON COLUMN Lots.LogID IS 'table auto inc id';
COMMENT ON COLUMN Lots.Phases_LogID IS 'foreign key phases table';

```

```

CREATE TABLE Phases
(
  DateTime       timestamptz,
  Unit           varchar   ,
  Phase_ID       varchar   ,
  Value          float8    ,
  EU             varchar(20),
  ID             VARCHAR(20),
  Formatted_Timestamp text ,
  LogID          bigserial  NOT NULL,
  Events_LogID   bigserial  NOT NULL,
  Lots_LogID     bigserial  NOT NULL,
  EU_LogID       bigserial  NOT NULL,
  PRIMARY KEY (LogID)
);

COMMENT ON COLUMN Phases.Formatted_Timestamp IS 'timestamp
serialized';
COMMENT ON COLUMN Phases.LogID IS 'table auto inc id';
COMMENT ON COLUMN Phases.Events_LogID IS 'foreign key Events table';
COMMENT ON COLUMN Phases.Lots_LogID IS 'foreign key Lots table';
COMMENT ON COLUMN Phases.EU_LogID IS 'foreign key EU table';

```

```

ALTER TABLE Phases
  ADD CONSTRAINT FK_Events_TO_Phases
    FOREIGN KEY (Events_LogID)
    REFERENCES Events (LogID);

ALTER TABLE Phases
  ADD CONSTRAINT FK_Lots_TO_Phases
    FOREIGN KEY (Lots_LogID)
    REFERENCES Lots (LogID);

```

```

ALTER TABLE Phases
  ADD CONSTRAINT FK_EU_TO_Phases
    FOREIGN KEY (EU_LogID)
    REFERENCES EU (LogID);

```


IX. PIPELINE

Se desarrolla usando airflow en modo standalone para crear los pipelines usando DAGs en python.

Para esto se crea un nuevo folder y se establece como folder de trabajo en VSCode y se usa pyenv para ajustar la version de python a 3.9.6 para el directorio local esto debido a que necesitamos mantener la estabilidad entre airflow, alchemysql, pandas y yml

Se crean 3 dags para realizar las tareas de ETL.

1. dag_01_getdata.py: hace la ingesta de datos y mantiene la integridad del sistema de archivos borrando archivos mas antiguos que 7 dias, trae el archivo bulkraw.csv desde github a disco duro y genera un data profile para permitir analizar los datos.
2. dag_02_extract.py: toma los datos desde el archivo en disco hace una limpieza de los datos huerfanos (nulos o NaN), usando un archivo de configuracion yaml crea la base de datos para staging, crea la tabla para los datos crudos y hace el vaciado de los mismos a la tabla.
3. dag_03_transform.py: toma la tabla de datos crudos, les hace un proceso de limpieza, aplica las transformaciones necesarias como normalizar los nombres de las columnas, busca datos huerfanos para imputarlos, luego usando un archivo con scripts de sql para crear las tablas necesarias para luego vacear los datos limpios a las tablas pre-diseñadas.

Diagrama 01: dag_01_getdata.py

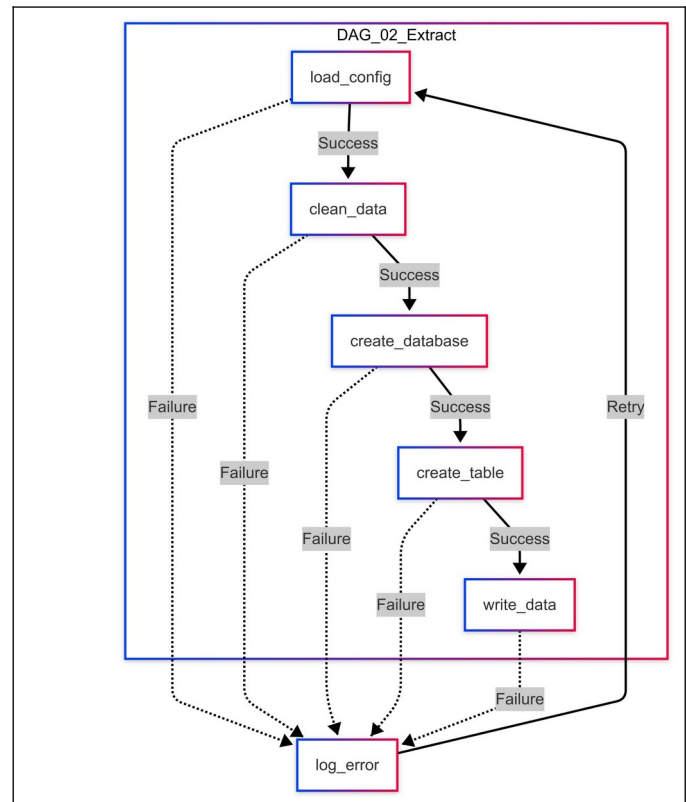
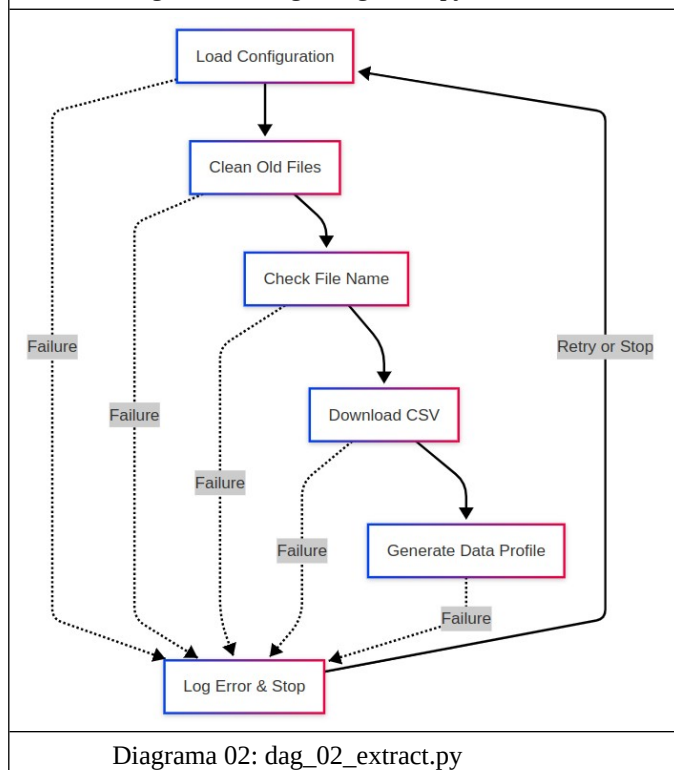
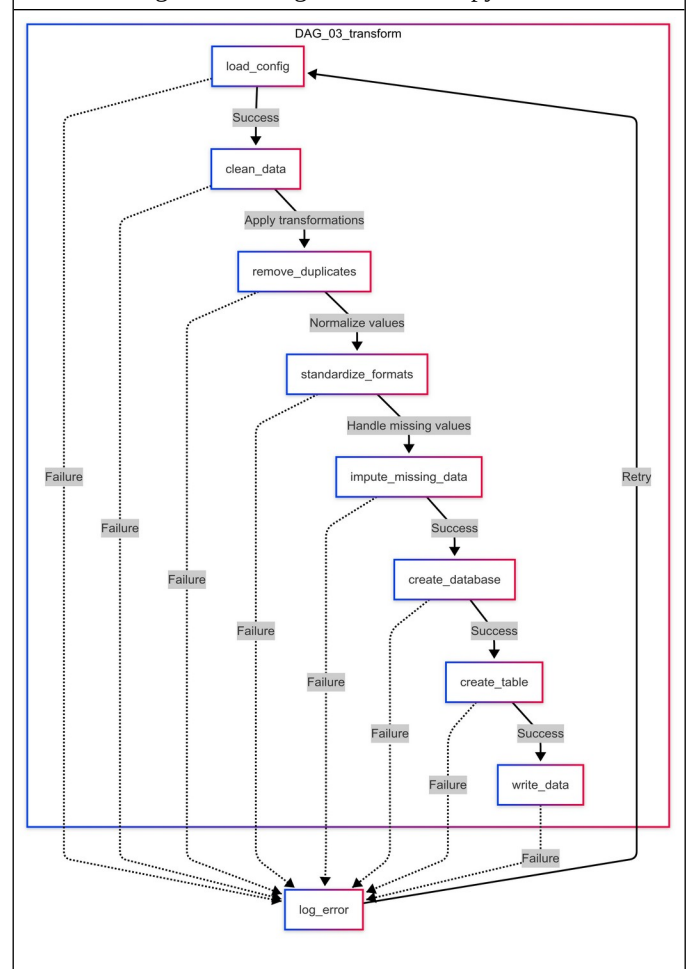


Diagrama 03: dag_03_transform.py



X. DAG_01_GETDATA:

La obtención y Perfilado de Datos, este DAG garantiza que los datos estén actualizados y listos para su análisis y sigue el siguiente flujo:

- A) Carga de configuración (job_00_load_config)
 - Carga un archivo YAML con configuraciones esenciales, como la ruta de los datos, el nombre del archivo, la URL de descarga y otros parámetros.
 - Verifica que las claves necesarias estén presentes.
 - Crea la carpeta de datos si no existe.
- B) Limpieza de archivos antiguos (job_01_clean_old_files)
 - Elimina archivos antiguos en la carpeta de datos según la política de retención definida en la configuración.
 - Filtra por extensión de archivo y elimina aquellos más antiguos que report_retention_days.
- C) Verificación del nombre del archivo (job_02_check_file_name)
 - Extrae el nombre del archivo a procesar desde la configuración y lo almacena en XCom para uso en tareas posteriores.
- D) Descarga del CSV (job_03_get_csv)
 - Obtiene el CSV desde la URL especificada en la configuración.
 - Guarda el archivo en la ruta de datos definida.
 - Si hay un problema en la descarga, genera un error y detiene la ejecución.
- E) Generación de informe de calidad de datos (job_04_profiling)
 - Carga el CSV en un DataFrame de Pandas.
 - Genera un informe de perfilado de datos con ydata_profiling, incluyendo estadísticas sobre los valores, tipos de datos, valores nulos, duplicados, etc.
 - Guarda el informe en formato HTML en la carpeta de datos.
- F) Manejo de Errores y Retrys

Cada tarea tiene:

 - Un callback task_success_callback para registrar ejecuciones exitosas.
 - Un callback task_failure_callback para registrar errores.
 - Configuración de retries=2 con un retry_delay de 5 minutos en caso de fallos.
- G) ¿Cómo lo hacemos?
 - Airflow DAG: Define tareas (PythonOperator) y su orden de ejecución con dependencias en cadena.
 - Manejo de configuración: Usa variables de Airflow y un archivo YAML (configuration.yaml).

- Gestión de archivos: Borra archivos antiguos según la política de retención.
- Descarga de datos: Obtiene un archivo CSV desde una URL y lo almacena localmente.
- Perfilado de datos: Usa pandas y ydata_profiling para generar un informe de calidad en HTML.

XI. DAG_02_EXTRACT

Este es un DAG de Airflow que sigue una serie de pasos para limpiar datos, crear una base de datos en PostgreSQL, crear una tabla y escribir los datos procesados en ella.

- A) Limpia los datos:
 - Lee un archivo CSV (Dataset.csv) desde una ruta definida en DATA_PATH.
 - Reemplaza valores no válidos (#VALUE!) con None.
 - Convierte columnas numéricas a formato correcto (Value y Verify).
 - Estiliza el texto en columnas de tipo string (elimina espacios y pone en minúsculas).
 - Guarda el CSV limpio (clean_Dataset.csv).
- B) Crea la base de datos si no existe:
 - Se conecta a PostgreSQL como usuario administrador.
 - Verifica si la base de datos ya existe.
 - Si no existe, la crea.
- C) Crea la tabla si no existe:
 - Define una estructura de tabla (RAW_TABLE_NAME).
 - Usa CREATE TABLE IF NOT EXISTS para crearla en la base de datos.
 - Escribe los datos en la tabla:
 - Lee el archivo clean_Dataset.csv.
 - Verifica que la tabla exista en PostgreSQL.
 - Inserta los datos en la tabla usando INSERT.
- D) ¿Cómo lo hacemos?
 - Airflow DAG: Define tareas (PythonOperator) y su orden de ejecución.
 - Manejo de configuración: Usa variables de Airflow y un archivo YAML (credentials.yaml).
 - Base de datos: Se conecta a PostgreSQL con psycopg2 y SQLAlchemy.
 - Limpieza y transformación: Utiliza pandas para modificar y guardar datos.

XII. DAG_03_TRANSFORM

- E) Carga de Configuración y Credenciales:
 - Se leen archivos YAML con las rutas de datos y credenciales de la base de datos.
 - Se verifica que las claves esenciales estén presentes.
- F) Creación de Tablas:
 - Se ejecuta un archivo SQL (create_tables.sql) para crear las tablas necesarias.
 - Se agregan restricciones (FOREIGN KEY) si no existen.

- G) Extracción de Datos:
- Se extraen datos desde la tabla `raw_table_name` en PostgreSQL.
 - Se convierten a un `DataFrame` de Pandas y se serializan en JSON.
- H) Transformaciones:
- General (`transform_data`):
 - Limpia valores inválidos, convierte tipos de datos y normaliza cadenas.
 - Entidades (`transform_eu_data`):
 - Extrae identificadores únicos (`EU_ID`), asigna descripciones y clasifica tipos.
 - Eventos (`transform_events_data`):
 - Analiza fechas, extrae métricas temporales (mes, semana, hora, turno, etc.).
 - Lotes (`transform_lots_data`):
 - Normaliza identificadores de producción y trenes.
 - Calcula duración de lotes en segundos y minutos.
 - Fases (`transform_phases_data`):
 - Filtra y formatea datos relacionados con fases de producción.
- I) Carga de Datos en PostgreSQL:
- Cada `DataFrame` transformado se convierte en JSON y se inserta en su tabla correspondiente (`eu`, `events`, etc.).
 - Se usa `INSERT ON CONFLICT DO NOTHING` para evitar duplicados.
- J) ¿Cómo lo hacemos?
- Airflow DAG: Define tareas (`PythonOperator`) y su orden de ejecución.
 - Manejo de configuración: Usa archivos `YAML` (`configuration.yaml`, `credentials.yaml`) para obtener parámetros y credenciales.
 - Base de datos: Se conecta a PostgreSQL usando `psycopg2` y `SQLAlchemy`.
 - Extracción de datos: Lee datos desde PostgreSQL con `pandas.read_sql_query()`.
 - Limpieza y transformación: Usa `pandas` para normalizar, convertir tipos de datos y calcular nuevas columnas.
 - Carga de datos: Inserta datos transformados en PostgreSQL con `to_sql()` y `ON CONFLICT DO NOTHING`.
 - Manejo de errores y logs: Implementa `try-except` en cada tarea y registra logs en Airflow.
 - Notificaciones: Usa `task_success_callback` y `task_failure_callback` para registrar el estado de cada tarea.

XIII. APACHE AIRFLOW:

Apache Airflow es una plataforma de orquestación de flujos de trabajo que permite programar, monitorear y gestionar tareas automatizadas (ETL, análisis de datos, machine learning, etc.).

Se usa para definir flujos de trabajo mediante DAGs (Directed Acyclic Graphs) y coordinar tareas y su ejecución en el orden correcto, además nos ofrece la capacidad de monitorear procesos con una interfaz web, registros de ejecución y alertas.

Además automatiza ETL (Extracción, Transformación y Carga de datos) integrándose con bases de datos, APIs, sistemas de almacenamiento, etc.

Las ventajas de airflow son:

Escalabilidad: Se adapta a grandes volúmenes de datos y tareas complejas.

Flexibilidad: Permite definir flujos en Python, integrando múltiples herramientas.

Observabilidad: Proporciona trazabilidad, logs y reintentos automáticos.

Ahora desde la carpeta de nuestro proyecto y en el ambiente activo del proyecto iniciamos airflow en modo standalone, este va a leer los dags y nos va a permitir la ejecución manual de nuestros pipelines.

Los retos de usar airflow es sin duda lograr mantener la compatibilidad entre airflow, python y sus dependencias, si esto no se lleva de forma correcta va a impactar la estabilidad de la ejecución de los pipelines y de la interfaz gráfica en sí misma.

Adicionalmente a esto, lograr que `AlchemySQL` y `pandas` se acoplen en el código es un reto en sí ya que el cambio de versiones y el tipo de uso han cambiado la forma de funcionamiento según el escenario de uso.

“`df.to_sql()` de Pandas” es una forma rápida y sencilla de escribir un `DataFrame` en una base de datos SQL, pero tiene limitaciones cuando trabajamos con Apache Airflow y PostgreSQL en un entorno productivo.

Algunas razones clave para que sea necesario evitar `df.to_sql()` son:

1. Falta de control sobre transacciones

`df.to_sql()` usa `INSERT` para cada fila sin optimizar el proceso.

No permite usar `BEGIN`, `COMMIT` o `ROLLBACK`, lo que es esencial en flujos ETL.

2. Baja eficiencia en grandes volúmenes de datos

`df.to_sql()` no está optimizado para cargas masivas en PostgreSQL.

Por ello es necesario usar `INSERT` en lugar de `COPY` y esto es mucho más rápido.

3. Dificultad para manejar errores

`df.to_sql()` no proporciona una gestión detallada de errores SQL.

Si falla en medio de una inserción, la tabla podría quedar en un estado inconsistente.

4. No permite ejecutar consultas SQL personalizadas

`df.to_sql()` no facilita la creación de tablas personalizadas con restricciones, índices o tipos de datos específicos.

Con `SQLAlchemy`, se puede definir la estructura de la tabla de manera más flexible.

5. ¿Cómo abordar correctamente la inserción en PostgreSQL con Airflow?

La mejor manera de manejar esto en Airflow es con `SQLAlchemy` y `psycopg2`, asegurando control de transacciones, manejo de errores y optimización de carga.

Este es el enfoque correcto:

Crear la tabla (si no existe)

Se usa `create_engine()` y `engine.begin()` para abrir una conexión segura y ejecutar `CREATE TABLE`:

```
'''Phyton
from sqlalchemy import create_engine, text
from sqlalchemy.exc import SQLAlchemyError
import logging

try:
    # Crear conexión con la base de datos
    engine = create_engine(f"postgresql://{db_user}:
{db_password}@{db_host}:{db_port}/{db_name}")

    # Crear la tabla usando una transacción
    with engine.begin() as conn:
        conn.execute(text(fixed_query))
        logging.info(f"Tabla {RAW_TABLE_NAME} creada
exitosamente.")
    except SQLAlchemyError as e:
        logging.error(f"Error en create_table: {e}")
        raise
'''
```

Ventajas:

- Usa transacciones (`begin()`) para evitar dejar la base de datos en un estado inconsistente.
- `text()` permite escribir consultas SQL personalizadas. `logging` permite depuración y monitoreo en Airflow.

Este método también permite insertar datos con eficiencia (usando `COPY` en lugar de `INSERT`), para insertar datos rápidamente, se recomienda usar `COPY`, que es más eficiente que `INSERT`:

```
'''Phyton
import pandas as pd
import io

def insert_data(df, engine, table_name):
    try:
        # Convertir DataFrame a CSV en memoria
        buffer = io.StringIO()
        df.to_csv(buffer, index=False, header=False, sep="\t") #
Usar tabulaciones para evitar problemas con comas
        buffer.seek(0)

        # Insertar datos con COPY para mayor eficiencia
        with engine.raw_connection() as conn:
            with conn.cursor() as cursor:
                cursor.copy_from(buffer, table_name, sep="\t",
null="")
            conn.commit()
            logging.info(f"Datos insertados correctamente en
{table_name}.")
        except Exception as e:
            logging.error(f"Error al insertar datos: {e}")
            raise
'''
```

Ventajas sobre `df.to_sql()`:

`COPY` es hasta 10 veces más rápido que `INSERT`.

Se usa `raw_connection()` para mayor eficiencia.

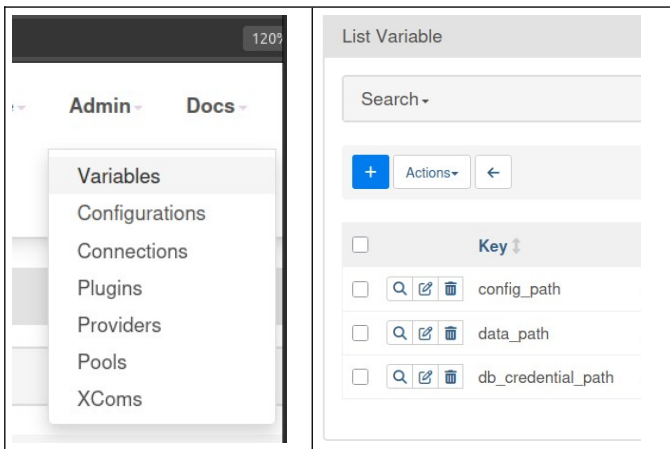
Se pueden evitar errores al manejar datos nulos correctamente (`null=""`).

Para ingresar a irflow es necesario colocar la dirección del loopback ó localhost y el puerto configurado en el archivo de configuración:

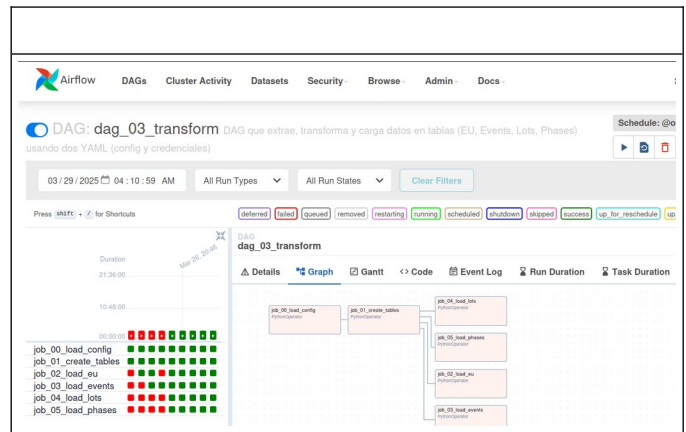
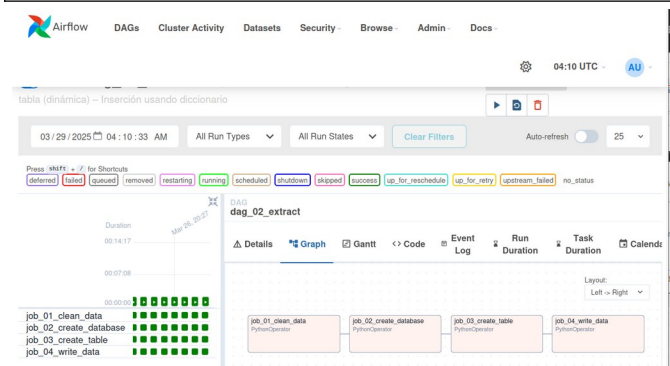
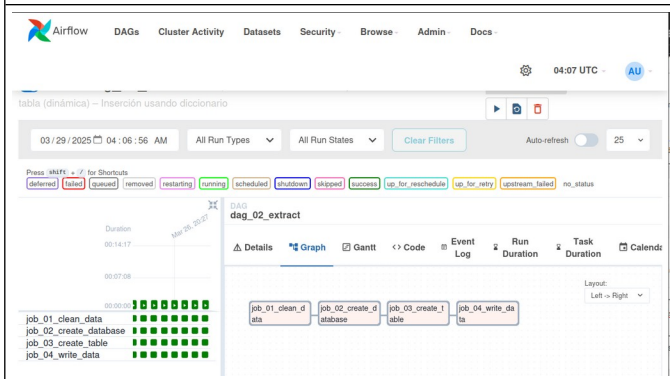
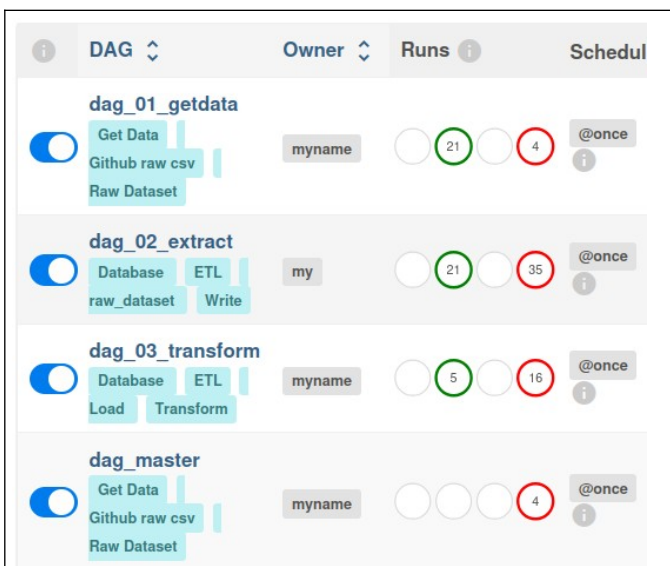
127.0.0.0:7070, el puerto por defecto es 8080.

para este proyecto debes de crear 3 variables que dene tener las ubicacione spara los archivos yaml:

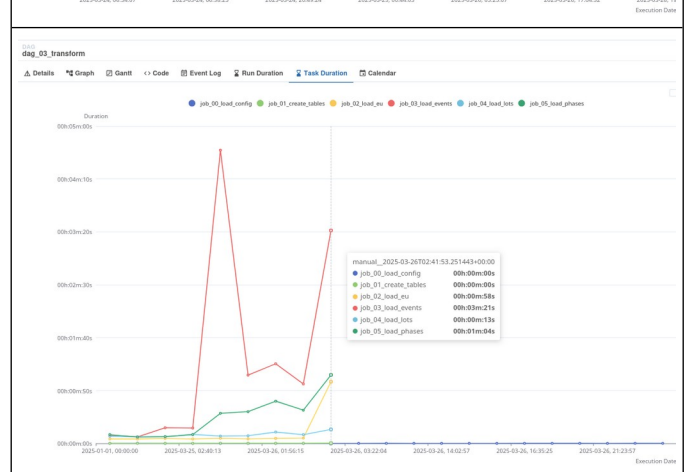
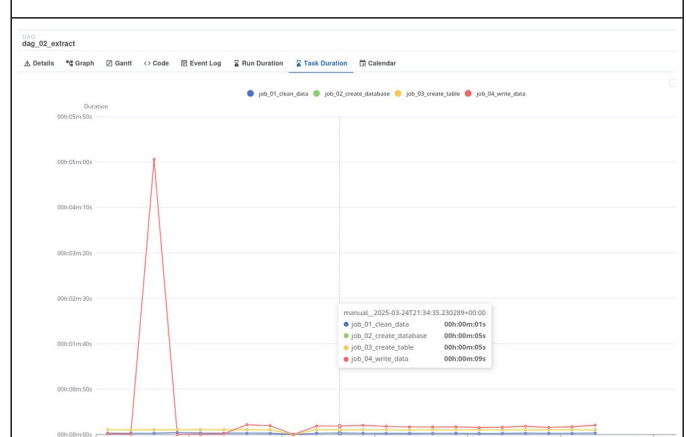
- `config_path`: configuration file yaml path
- `data_path`: Absolut path for datafiles and html reports
- `db_credential_path`: yaml file with database credentials path



En la interface web se realiza la actividad de administracion de los pipelines:



Duracion de ejecucion para los tres dags



XIV. POSTGRESQL

Las bases de datos se aborda la integración con la base de datos postgresql por 2 metodos diferentes en los dos DAGs 2 y 3.

En el Dag2 se crea la Base ded datos y la tabla para los datos crudos usando una funcion que toma una consulta en forma de script tsql y la envia al cursor del conector de la base de datos.

En el Dag3 se aborda algo diferente ya que los scripts estan escritos en un archivo .sql y se le envia a la funcion de ejecucion del engine, este método ya no necesita un cursor y le envia la orden directa de ejecutar un sql.

Uno de los retos fúe crear las restricciones (Constraints) entre las tablas, fue necesario enviarlas aparte colocandolas en un diccionario y luego pasandoselas al engine.

Uno de los problemas que me encontré fue popular las tablas con las restricciones, esto debido a que se hacen vaciados bulk a las tablas pero una a una y esto causa falla ya que las restricciones hace la validacion dato a dato tabla contra tabla y eso no puede hacerse en descargas tipo bulk.

Vista de las tablas:

Tabla de datos crudos creada en el pipeline 1:

Query Query History

SELECT id, "DateTime", "Lot_ID", "Prod_ID", "Type", "Train", "Unit", "Phase_ID", "EU", "Value", "Verify" FROM public.etl_dataset;

Data Output Messages Notifications

	id	DateTime	Lot_ID	Prod_ID	Type	Train	Unit	Phase_ID	EU	Value	double precision	Verify
	[PK]	timestamp with time zone	character varying	character varying	character v	character v	character v	character vary				
1	1	2025-02-27 07:34:21.05	b7ba9ddm	7426.0	f	c	pmx	mdf66	kg	1	0	0
2	2	2025-02-05 06:57:51.05	bwaagddm	7426.0	f	c	pmx	mdf66	kg	1.003703704	0	0
3	3	2025-02-05 02:47:48.05	9agagddm	7426.0	f	c	pmx	mdf66	kg	0.996296296	0	0
4	4	2025-02-04 23:04:32.05	v7agad67	7426.0	f	c	pmx	mdf66	kg	1.014814815	0	0
5	5	2025-02-04 17:45:15.05	v9agad66	7426.0	f	c	pmx	mdf66	kg	0.97637037	0	0
6	6	2025-02-04 15:15:41.05	g27agad65	7426.0	f	c	pmx	mdf66	kg	0.988888889	0	0
7	7	2025-02-04 11:36:37.05	raagad65	7426.0	f	c	pmx	mdf66	kg	1	0	0
8	8	2025-02-03 18:31:57.05	flagad6m	7426.0	f	c	pmx	mdf66	kg	1.003703704	0	0
9	9	2025-02-03 08:31:47.05	etwagad6k	7426.0	f	c	pmx	mdf66	kg	0.996296296	0	0
10	10	2025-01-18 17:04:46.05	afagay0y	7426.0	f	c	pmx	mdf66	kg	1.003703704	0	0
11	11	2025-01-20 06:18:26.05	dbagay0y	7426.0	f	c	pmx	mdf66	kg	1.003703704	0	0
12	12	2025-01-19 23:07:44.05	9t/agacch	7426.0	f	c	pmx	mdf66	kg	1	0	0
13	13	2025-01-19 18:16:15.05	y9agacch	7426.0	f	c	pmx	mdf66	kg	1	0	0
14	14	2025-01-19 04:44:05.05	37agacpe	7426.0	f	c	pmx	mdf66	kg	1	0	0
15	15	2025-01-19 02:50:53.05	jzaxcccd	7426.0	f	c	pmx	mdf66	kg	1	0	0
16	16	2025-01-18 17:45:30.05	xiaocwcc	7426.0	f	c	pmx	mdf66	kg	1	0	0
17	17	2025-01-02 02:21:23.05	ga7ag71ld	7426.0	f	c	pmx	mdf66	kg	1	0	0

```

POSTGRESQL EXPLORER: POSTGRESQL
+
  localhost
  etl_prj
    public
      Functions
      etl_dataset
        id : integer
        DateTime : timestamp with time zone
        Lot_ID : character varying(250)
        Prod_ID : character varying(100)
        Type : character varying(10)
        Train : character varying(10)
        Unit : character varying(10)
        Phase_ID : character varying(100)
        EU : character varying(10)
        Value : double precision
        Verify : smallint
      eu
      events
      lots
      phases
      metabase_db

```

Tabla de eventos

Events	Event Log				
Datetime	timestampz	N-N	default	UTC Tiezone-aware	
Year	smallint	N-N	default	year number	
Quarter	smallint	N-N	default	quarte number	
Month	smallint	N-N	default	month number	
MonthName	text	N-N	default	month name	
Week	smallint	N-N	default	week number	
WeekDay	smallint	N-N	default	day of a week	
DayOfYear	smallint	N-N	default	day of a year (0-365)	
Day	smallint	N-N	default	day of a month	
DayName	text	N-N	default	day name	
shift_8H	smallint	N-N	default	shifts of 8 hours	X
Formatted_Timestamp	text	N-N	default	timestamp serialized	
LogID	bigserial	N-N	default	table auto inc id	
Phases_LogID	bigserial	N-N	default	foreign key phases table	

Tabla de lotes


Lots		Produced Lots		
Lot_ID	VARCHAR(20)	NULL	default	Lot unique id
Type	VARCHAR(2)	NULL	default	type of product
Train_ID	VARCHAR(2)	NULL	default	train id
Train	VARCHAR(400)	NULL	default	train label
Prod_ID	VARCHAR(20)	NULL	default	product id
Prod_Name	VARCHAR(400)	NULL	default	product label
First_Date	timestamptz	NULL	default	first register datetime
Last_Date	timestamptz	NULL	default	last register datetime
Duration	timetz	NULL	default	difference between first and last datetime
Formatted_Timestamp	text	NULL	default	timestamp serialized
 LogID	bigserial	N-N	default	table auto inc id
Phases_LogID	bigserial	N-N	default	foreign key phases table

Tabla de unidades

EU	comment			
EU_ID	varchar(20)	NULL	default	Engineering Units ID
EU_name	varchar(400)	NULL	default	Engineering Units label
Description	varchar	NULL	default	Engineering Units desc
Type	varchar(400)	NULL	default	phase type
LogID	bigserial	N-N	default	table auto inc id
Phases_LogID	bigserial	N-N	default	foreign key phases table

Phases	comment			
DateTime	timestamptz	NULL	default	comment
Unit	varchar	NULL	default	comment
Phase_ID	varchar	NULL	default	comment
Value	float8	NULL	default	comment
EU	varchar(20)	NULL	default	comment
ID	VARCHAR(20)	NULL	default	comment
Formatted_Timestamp	text	NULL	default	timestamp serialized
LogID	bigserial	N-N	default	table auto inc id
Events_LogID	bigserial	N-N	default	foreign key Events table
Lots_LogID	bigserial	N-N	default	foreign key Lots table
EU_LogID	bigserial	N-N	default	foreign key EU table

Tabla de Fases

XV. DASHBOARD, METABASE

Metabase es una herramienta de Business Intelligence (BI) de código abierto que permite visualizar datos sin necesidad de escribir consultas SQL complejas. Con Metabase, los usuarios pueden:

- ◆ Conectar bases de datos como PostgreSQL, MySQL, BigQuery, entre otras.
- ◆ Crear paneles e informes con gráficos interactivos.
- ◆ Explorar datos de forma intuitiva sin conocimientos avanzados de SQL.
- ◆ Compartir visualizaciones con equipos o clientes fácilmente.

Es una excelente opción para empresas que necesitan análisis de datos sin depender completamente de equipos técnicos.

Enlace oficial: <https://www.metabase.com/>

Conexion con postgresql

Configuración Metabase

Bases de datos > ETL

Tipo base de datos: PostgreSQL

Nombre para mostrar: eti

Servidor: localhost

Puerto: 5432

nombre de la base de datos: eti_prj

Nombre Usuario: postgres

Contraseña: [oculto]

Schemas: Todos

Utilizar una conexión segura (SSL): [activado]

Modo SSL: require

Tablas ocultas a los usuarios

Configuración Metabase

Datos Segmentos Métricas

eti

Encontrar una tabla

4 TABLAS CONSULTABLES

Eu

Events

Lots

Phases

1 TABLA ESCONDIDA

Eti Dataset

No hay una descripción de la tabla

VISIBILIDAD Consultable Oculto

Columnas Esquema original

CAMPO VISIBILIDAD

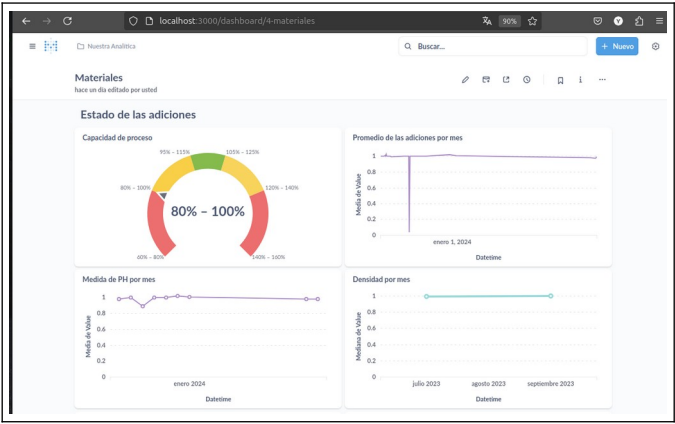
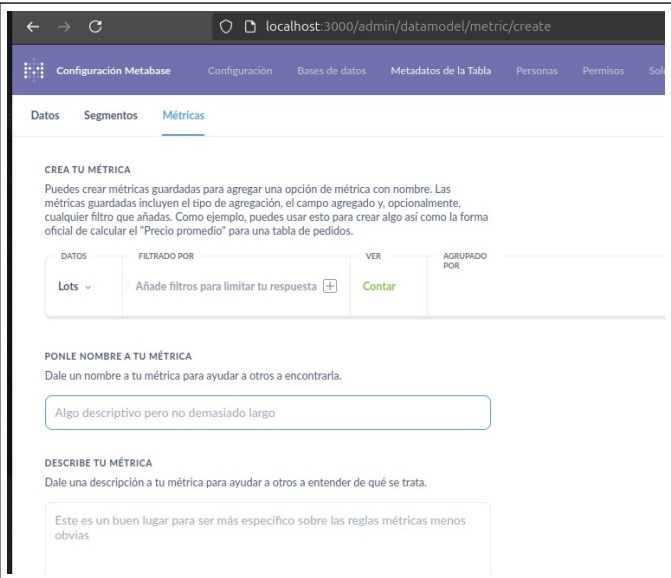
eu_id

Eu ID

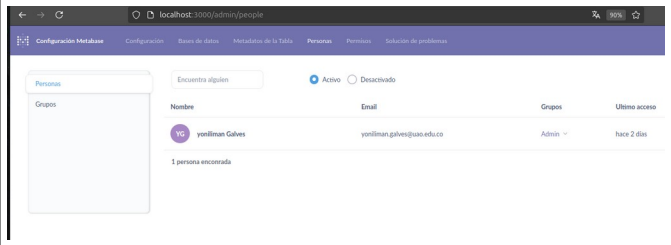
Engineering Units ID

eu_name

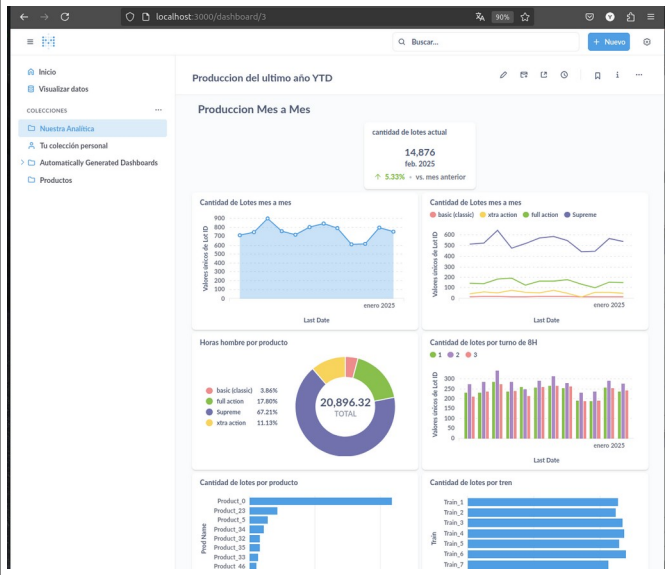
Creacion de métricas



Manejo de usuarios



Dashboards presentados



Dashboard de Materiales

XVI. BIBLIOGRAPHY

- [1] P. Mooring, "Predicción de valores con Deep Learning y Keras," 22 01 2022. [Online]. Available: <https://www.peterspython.com/es/blog/prediccion-de-valores-con-deep-learning-y-keras>. [Accessed 18 11 2023].
- [2] j. Brownlee, "How to Use the ColumnTransformer for Data Preparation," 31 12 2020. [Online]. Available: <https://machinelearningmastery.com/columntransformer-for-numerical-and-categorical-data/>. [Accessed 01 11 2023].
- [3] T. J. Fan, "VOTE SLEP018 - Pandas Output for Transformers," 17 07 2022. [Online]. Available: https://github.com/scikit-learn/enhancement_proposals/pull/72. [Accessed 01 11 2023].
- [4] T. J. Fan, A. Mueller and J. V. den Bossche, "ColumnTransformer & Pipeline Simplified," [Online]. Available: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/compose/_column_transformer.py. [Accessed 01 11 2023].
- [5] D. Davis, "Selección automática de características en Python: una guía esencial," 19 07 2021. [Online]. Available: <https://hackernoon.com/es/seleccion-automatica-de-caracteristicas-en-python-una-guia-esencial-uv3e37mk>. [Accessed 01 11 2023].
- [6] elmundodelosdatos.com, "Identificación e imputación de valores perdidos en Python," 05 06 2021. [Online]. Available: <https://elmundodelosdatos.com/identificacion-valores-perdidos-python/>. [Accessed 01 11 2023].
- [7] J. Brownlee, "Dropout Regularization in Deep Learning Models with Keras," 08 06 2022. [Online]. Available: <https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/>. [Accessed 01 11 2023].
- [8] R. Vaquerizo, "Modelos lineales dinámicos (DLM) con R," 07 09 2014. [Online]. Available: <https://analisisydecision.es/modelos-lineales-dinamicos-dlm-con-r/>. [Accessed 01 11 2013].

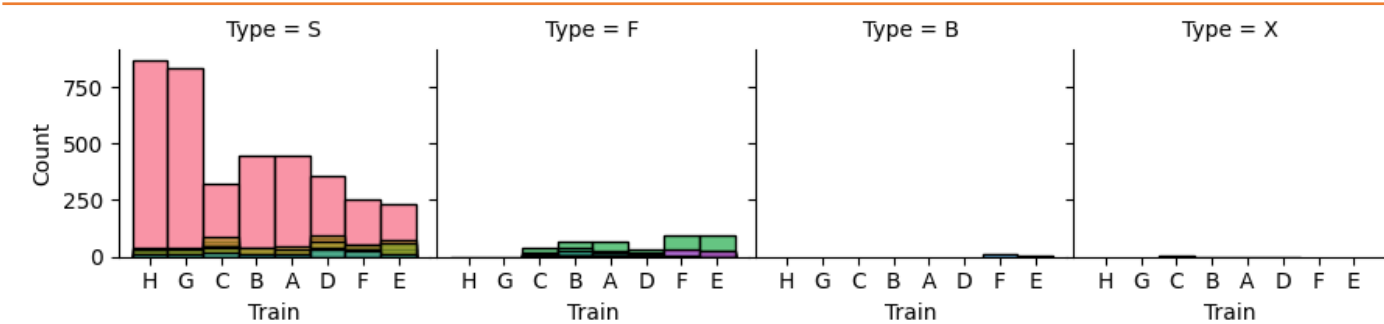


Imagen 2: Cantidad de Lotes producidos por cada tren por tipo de producto

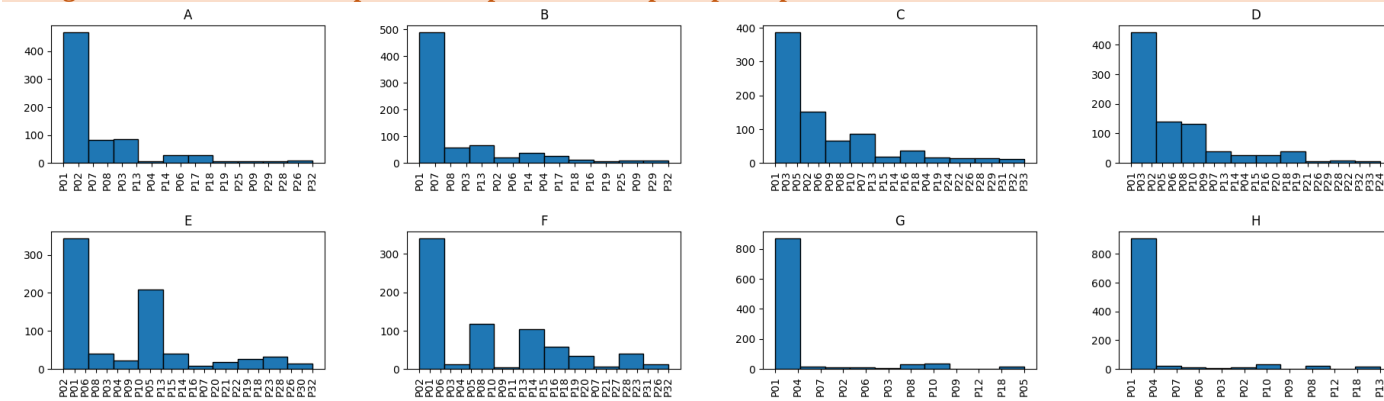


Imagen 3: Cantidad de Lotes producidos por producto y Tipo

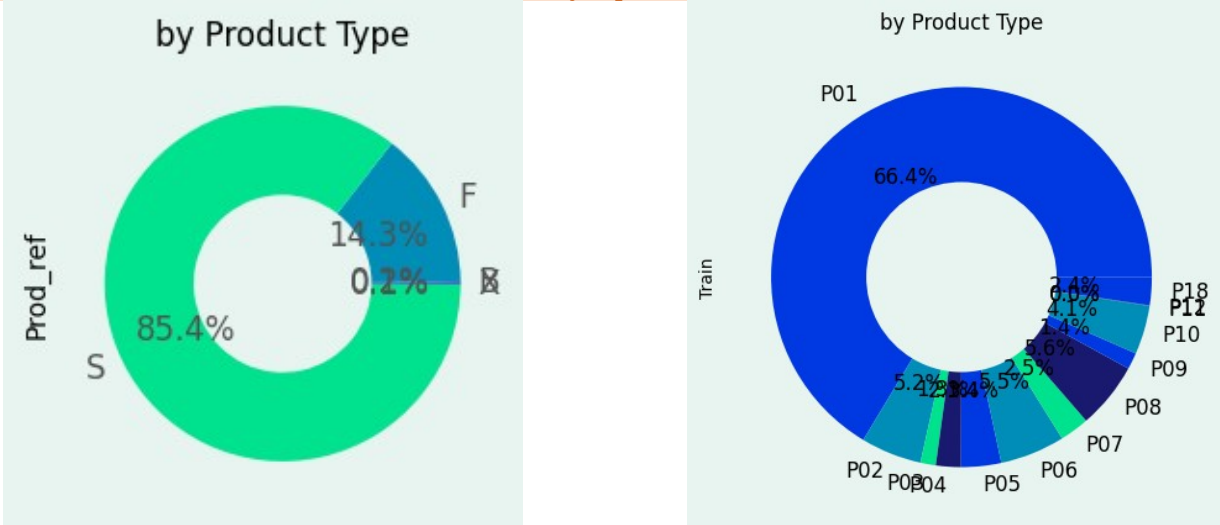
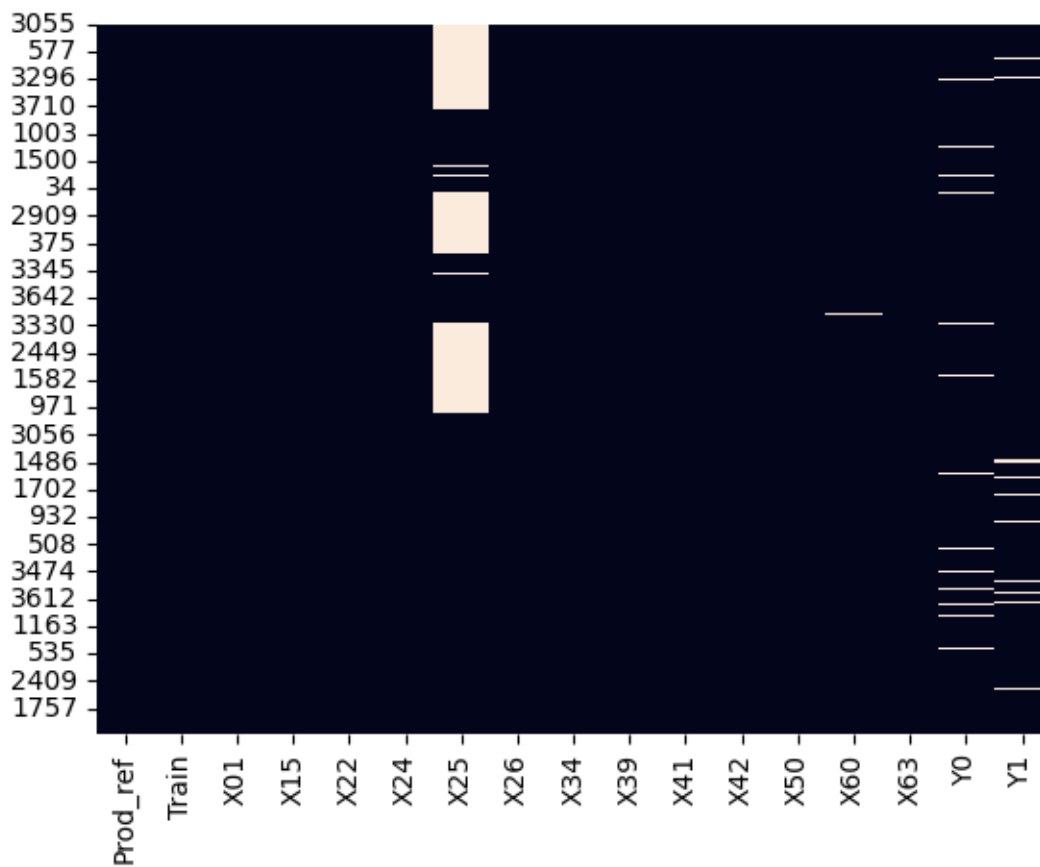
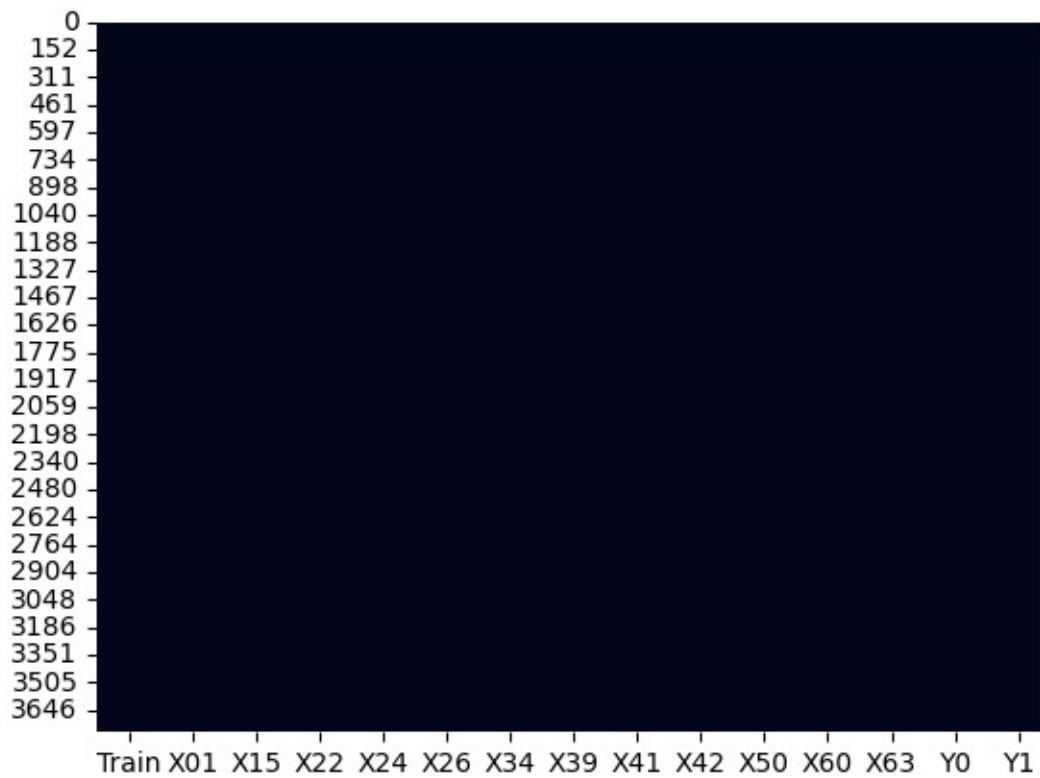


Imagen 4: Pastel de Familias de Productos

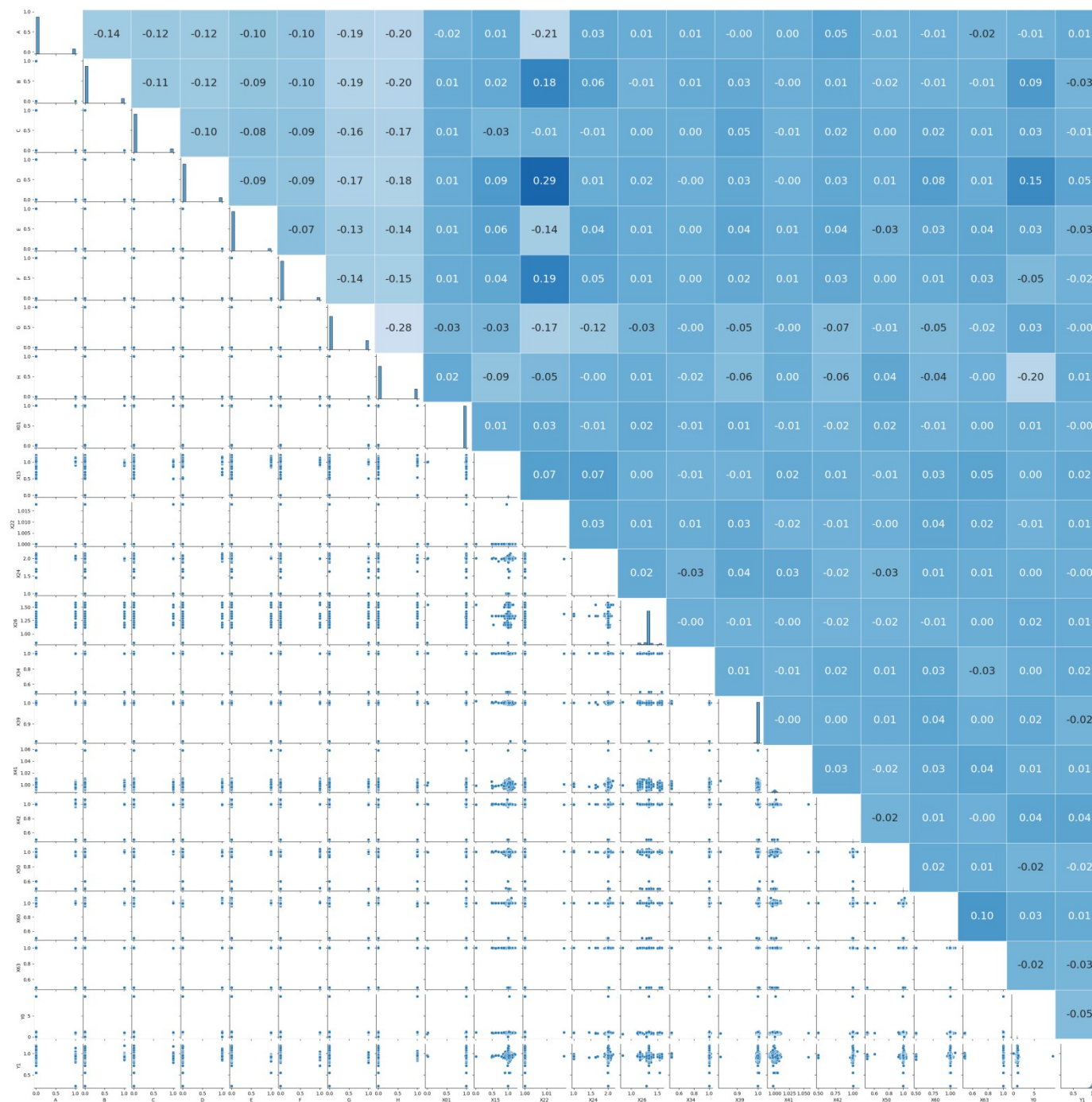
Error: Reference source not found



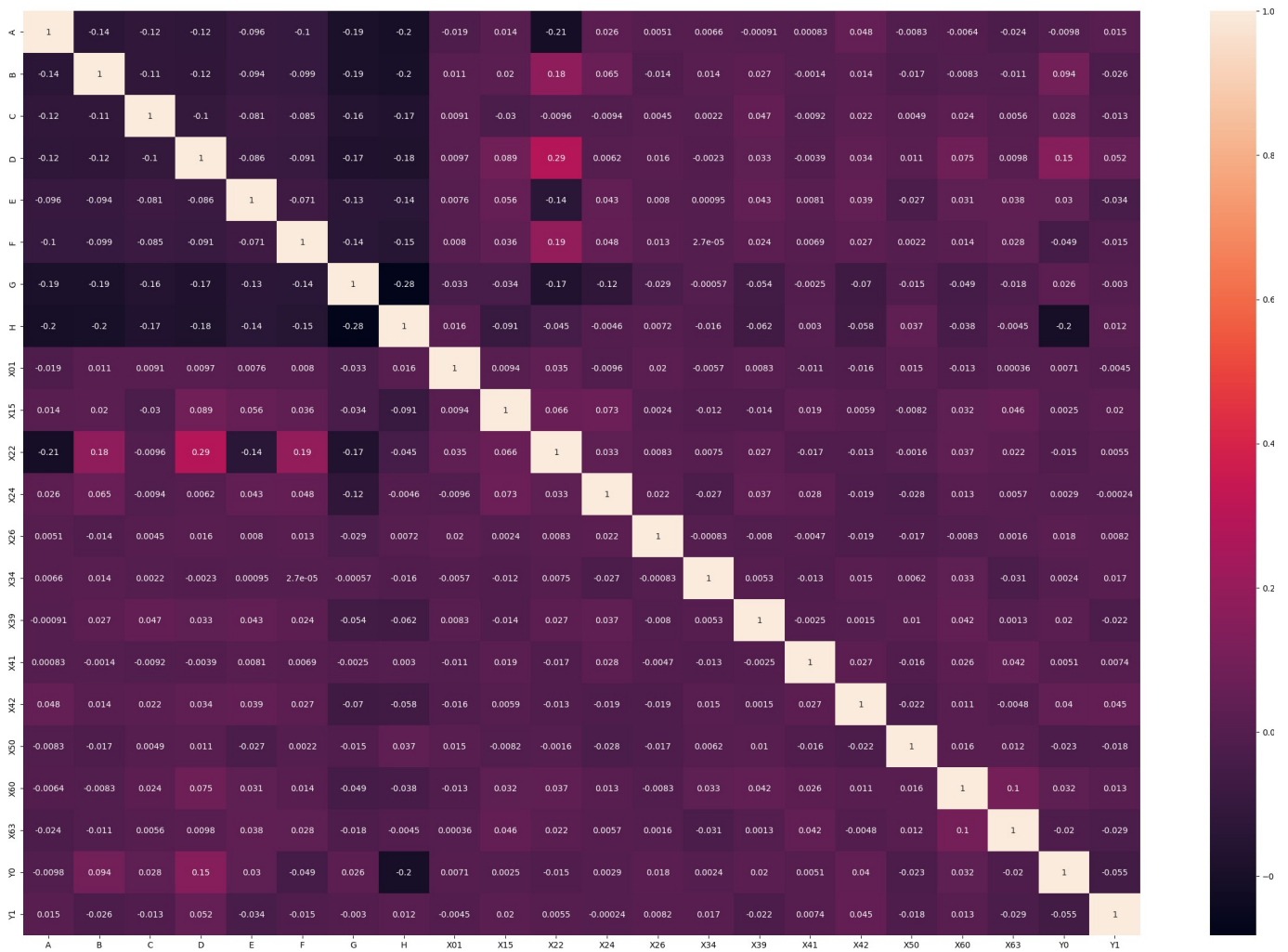
Error: Reference source not found



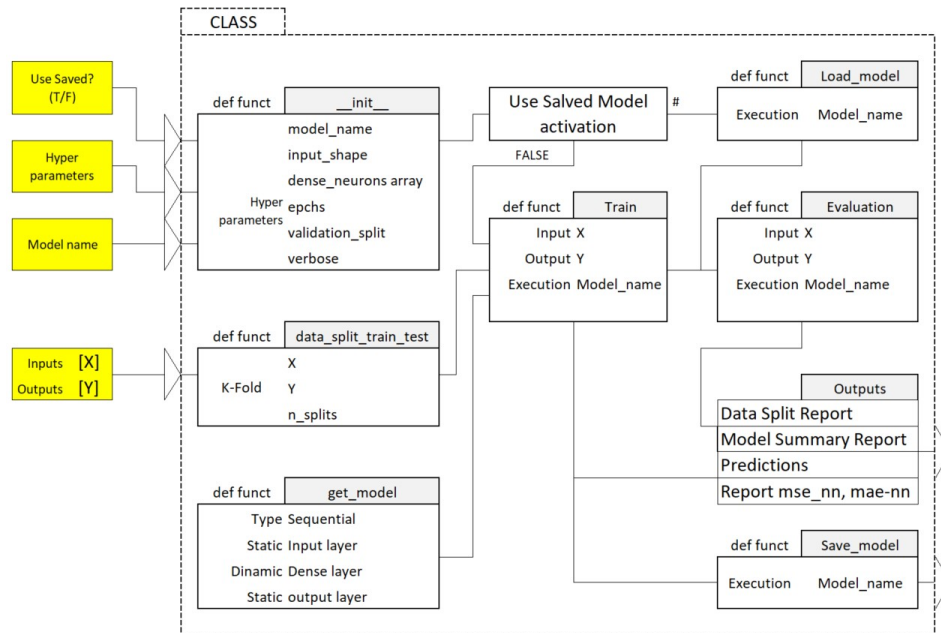
Error: Reference source not found



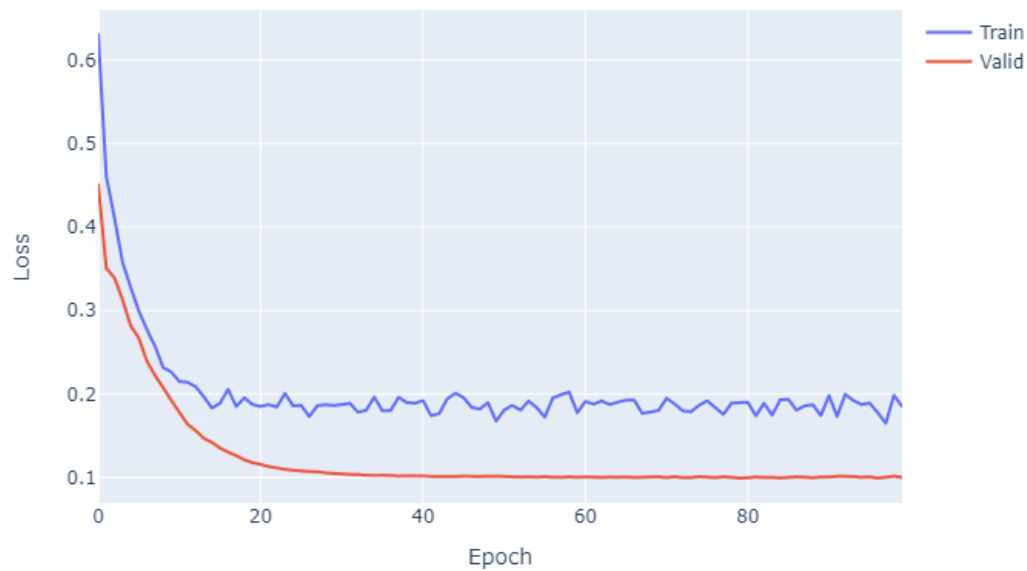
Error: Reference source not found



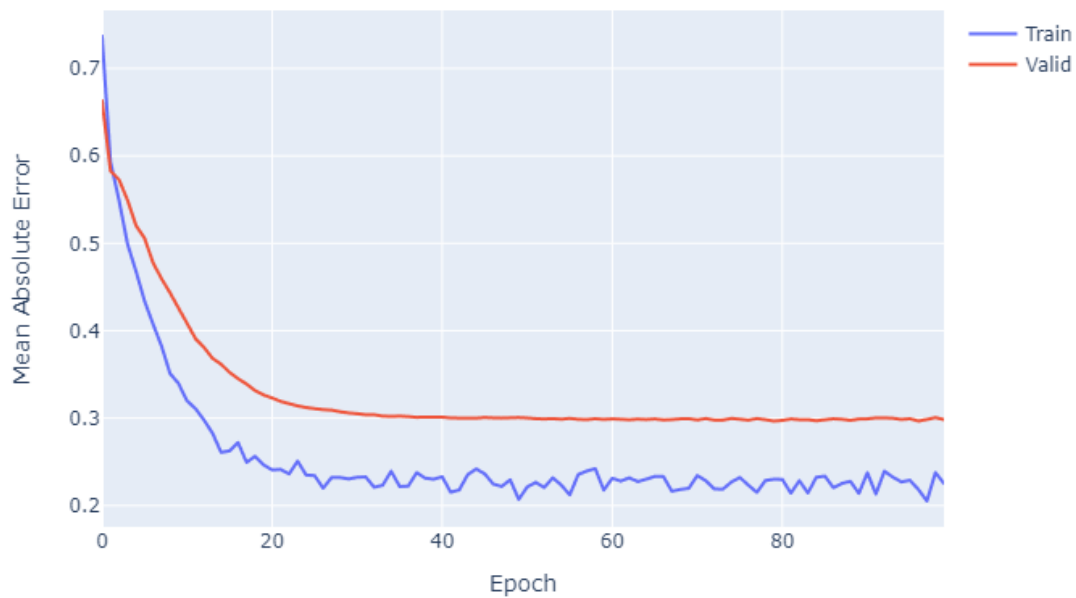
Error: Reference source not found



Error: Reference source not found



Error: Reference source not found



Error: Reference source not found

```
37/37 [=====] - 0s 2ms/step - loss: 0.0735 - mean_absolute_error: 0.1977  
Mean squared error on test data: 0.07350727170705795  
Mean absolute error on test data: 0.19767481088638306
```

Error: Reference source not found