Proyecto Final: Implementación de un Proceso ETL para el Análisis de Datos Inmobiliarios con Azure Data Factory

Eduardo José Avendaño Caicedo
Sebastián Dow Valenzuela

Maestría en Ciencia de Datos
Curso: Infraestructura de TI
PhD. Angela Villota Gómez
Universidad Icesi

Junio de 2025

Resumen

Este informe detalla el diseño, implementación y validación de un proceso de Extracción, Transformación y Carga (ETL) utilizando la plataforma en la nube de Microsoft Azure. El objetivo principal fue consolidar datos heterogéneos sobre propiedades inmobiliarias de la ciudad de Ames, provenientes de archivos CSV, una base de datos relacional PostgreSQL y colecciones de una base de datos NoSQL MongoDB. Se empleó Azure Data Factory como servicio de orquestación central, Azure Blob Storage como área de almacenamiento intermedio (staging) y de destino, y la funcionalidad de Mapping Data Flow para la implementación visual de la lógica de transformación. El resultado es un único archivo CSV, limpio, estructurado con 81 columnas y listo para ser consumido por modelos de análisis avanzado. El proyecto demuestra un flujo de trabajo de ingeniería de datos de extremo a extremo, abordando desafíos como la integración de esquemas dispares, el manejo de granularidad de datos y la limpieza sistemática de valores nulos.

Índice

1 Introducción

1.1 Contexto del Problema

En el campo de la ciencia de datos, la capacidad de analizar grandes volúmenes de información es precedida por un desafío fundamental: la consolidación y preparación de los datos. Frecuentemente, los datos de interés se encuentran dispersos en sistemas heterogéneos, cada uno con su propio esquema, formato y nivel de calidad. El conjunto de datos inmobiliarios de Ames, Iowa, utilizado en este proyecto, es un ejemplo representativo de este escenario, con información distribuida en una base de datos relacional (PostgreSQL), una base de datos NoSQL (MongoDB Atlas) y archivos de texto plano (CSV). La integración manual de estas fuentes es un proceso propenso a errores, ineficiente y no escalable.

1.2 Objetivo del Proyecto

El objetivo central de este proyecto es diseñar e implementar un pipeline de ETL automatizado y robusto en la nube de Azure. Dicho pipeline debe ser capaz de extraer datos de las tres fuentes mencionadas, aplicar un conjunto de transformaciones para limpiar, unificar, enriquecer y estructurar la información, y finalmente cargar el resultado en un único archivo CSV de salida, conforme a un esquema predefinido de 81 columnas.

1.3 Alcance y Tecnologías Utilizadas

El proyecto abarca un flujo de trabajo completo de ingeniería de datos. Las tecnologías y servicios clave empleados fueron:

- **Azure Data Factory (ADF):** Servicio de orquestación central para el diseño, ejecución y monitoreo del pipeline ETL.
- **Azure Blob Storage:** Utilizado como área de almacenamiento para el archivo fuente `Property.csv`, como zona de *staging* para los datos de MongoDB, y como destino final para el archivo procesado `salida.csv`.
- **Neon (PostgreSQL):** Base de datos relacional como servicio (DBaaS) que aloja los datos estructurales de las propiedades.
- **MongoDB Atlas:** Base de datos NoSQL como servicio que contiene colecciones con datos semi-estructurados y opcionales.

2 Diseño de la Arquitectura en la Nube

La base del proyecto fue el despliegue de una arquitectura lógica y organizada en Azure, contenida dentro de un único Grupo de Recursos para facilitar la gestión y el control de costos.

2.1 Configuración de Recursos

Se aprovisionaron los siguientes recursos, todos en la región `East US` para minimizar la latencia y optimizar los costos de transferencia de datos:

- **Grupo de Recursos:** `rg-etl-ames` como contenedor lógico de todos los componentes.
- **Cuenta de Almacenamiento:** `stetlamessd` de tipo `StorageV2 (general purpose v2)` con redundancia `LRS`. Dentro de esta, se crearon dos contenedores: `entrada` y `salida`, para separar lógicamente los datos crudos de los procesados.
- **Data Factory:** `adf-etl-ames-sd`, la instancia de ADF v2 que actúa como el cerebro del proceso.

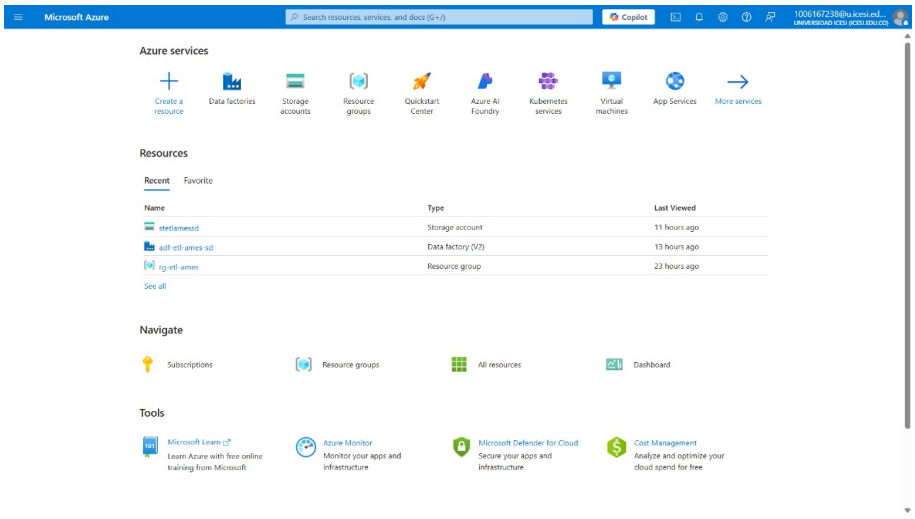


Figura 1: Vista de los recursos aprovisionados dentro del Grupo de Recursos `rg-etl-ames` en el portal de Azure.

Evidencia a añadir (Fig. 1): *Un pantallazo del portal de Azure mostrando los tres recursos (Storage Account, Data Factory, Resource Group) creados.*

3 Metodología y Proceso ETL

El proceso ETL fue implementado utilizando un pipeline principal en ADF llamado `PL_Ames_ETL`. Este pipeline orquesta las actividades de copia y transformación de datos, asegurando un flujo de trabajo cohesivo y automatizado.

3.1 Fase 1: Conexión a Fuentes de Datos (Linked Services)

El primer paso fue establecer la comunicación entre ADF y los orígenes de datos mediante la creación de `Linked Services`:

- **AzureBlobStorageConn:** Conectado a la cuenta de almacenamiento mediante `Account Key` para acceder a los contenedores `entrada` y `salida`.

- **NeonConn (PostgreSQL):** Configurado con el conector `Azure Database for PostgreSQL`. Se encontró que la **versión 1.0** del conector era necesaria para la compatibilidad con la actividad `Data Flow`, constituyendo el primer desafío técnico superado.
- **MongoConn (MongoDB Atlas):** Se utilizó el conector nativo `MongoDB Atlas` con la cadena de conexión provista para establecer el enlace con la base de datos NoSQL.

3.2 Fase 2: Ingesta y Preparación de Datos (Staging)

Durante el desarrollo, se identificó una limitación importante: las colecciones de MongoDB Atlas no podían ser utilizadas directamente como fuentes en la actividad `Mapping Data Flow` de manera confiable. Para superar este obstáculo, se adoptó una estrategia de *staging*.

1. **Actividad de Copia (Copy Data):** Dentro del pipeline `PL_Ames_ETL`, se implementó una actividad `Copy Data` que se ejecuta *antes* del Data Flow.
2. **Fuente (Source):** La fuente de esta actividad fue configurada para conectarse a MongoDB Atlas a través del `MongoConn` y leer cada una de las colecciones (`garage`, `pool`, `bsmt`, `misc`).
3. **Destino (Sink):** El destino fue configurado para escribir los datos de cada colección como archivos individuales en formato JSON en una carpeta `staging_mongo` dentro del contenedor `entrada` de nuestra cuenta de Blob Storage.

Este paso desacopla la extracción de MongoDB de la transformación, convirtiendo los datos semi-estructurados en archivos JSON que el `Data Flow` puede consumir de manera estable y eficiente.

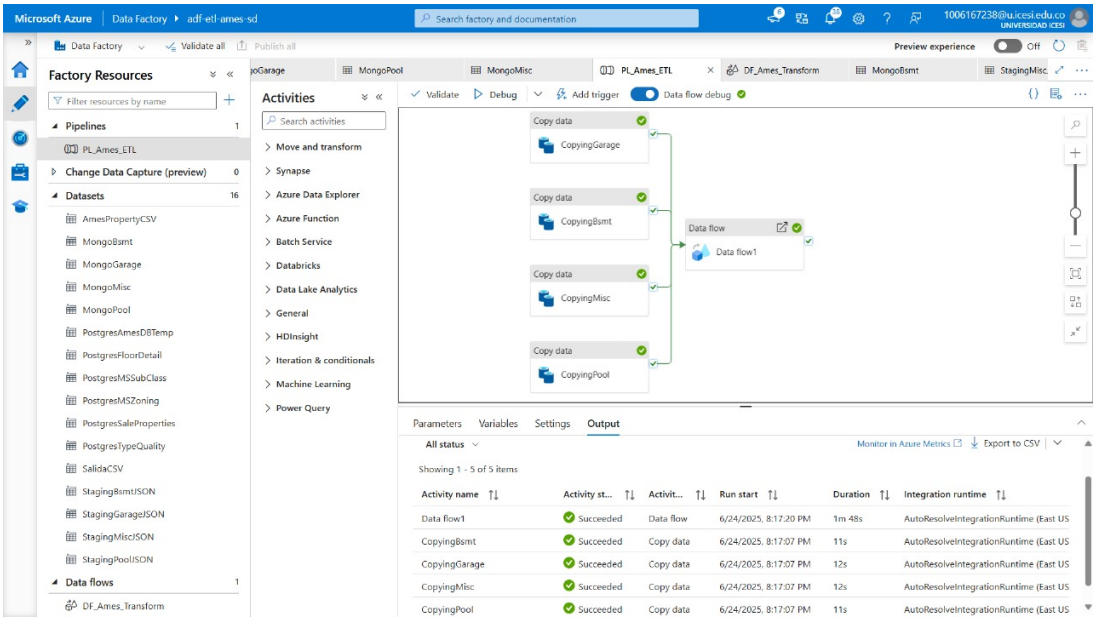
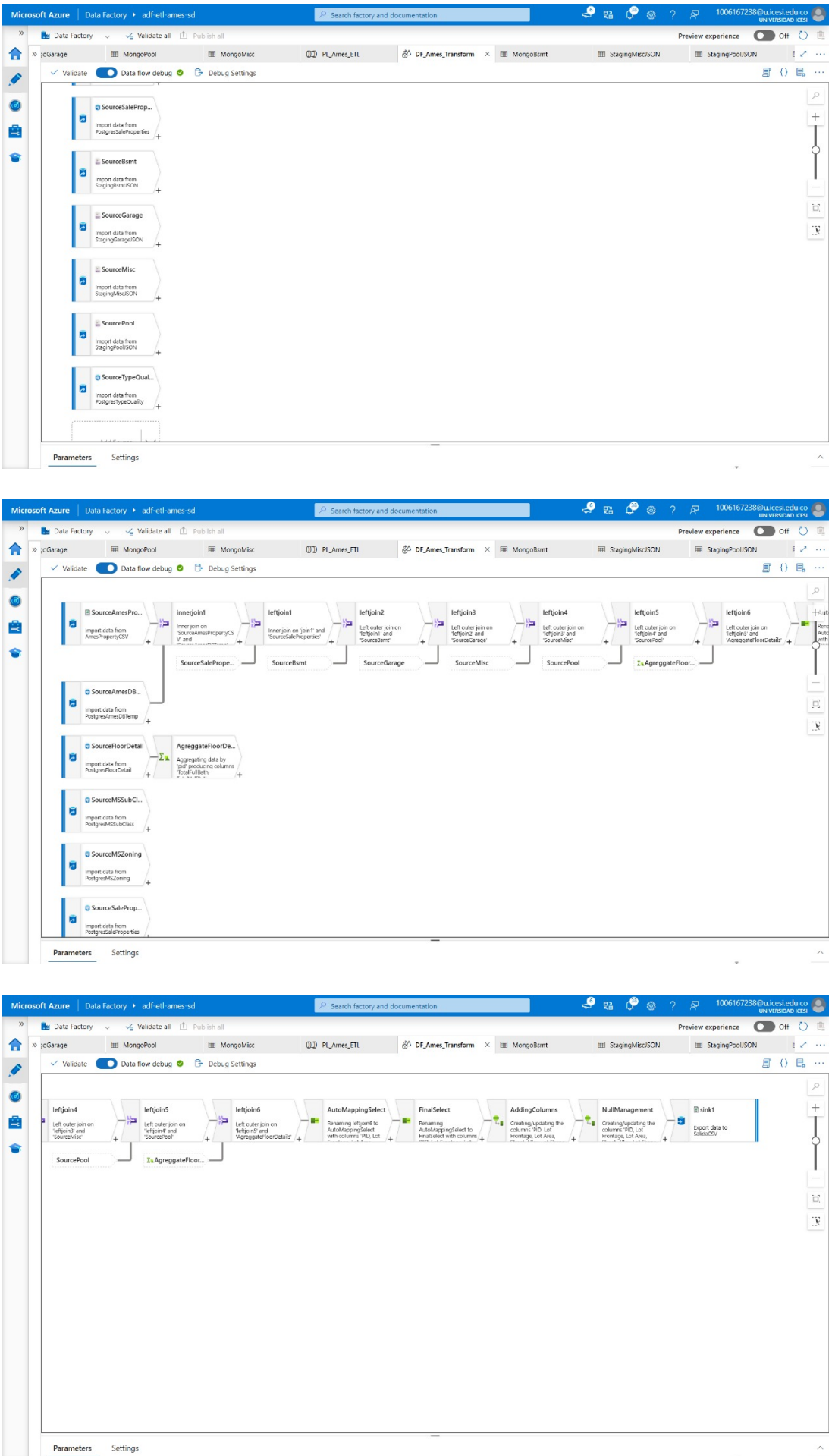


Figura 2: Vista del pipeline `PL_Ames_ETL`, mostrando la secuencia de la actividad `Copy Data` (para staging de MongoDB) seguida por la actividad `Data Flow`.

Evidencia a añadir (Fig. 2): *Un pantallazo del lienzo del pipeline principal, donde se vean las dos actividades conectadas.*

3.3 Fase 3: Transformación y Modelado (Mapping Data Flow)

El corazón del proyecto es un `Mapping Data Flow` llamado `DF_Ames_Transform`, que ejecuta la lógica de transformación en un clúster de Spark gestionado por ADF.



8

Figura 3: Diagrama completo del `Mapping Data Flow` en Azure Data Factory Studio.

Evidencia a añadir (Fig. 3): *Un pantallazo panorámico de todo el lienzo del Data Flow, desde las fuentes hasta el Sink.*

3.3.1. Integración de Fuentes y Estrategia de Unión

La consolidación de las fuentes se realizó mediante una cadena de transformaciones `Join`, usando el campo `PID` como clave.

- **Fuentes Utilizadas:** Se agregaron como fuentes el archivo `Property.csv`, las tablas de PostgreSQL, y los nuevos archivos JSON de la carpeta `staging_mongo`.
- **Inner Join:** Se utilizó para la unión inicial entre `Property.csv` y la tabla `amesdbtemp` de PostgreSQL para asegurar un conjunto de datos base consistente.
- **Left Outer Join:** Se utilizó para todas las uniones subsecuentes. Esta decisión fue fundamental para no perder registros de propiedades que carecían de datos opcionales (ej. piscina o garaje). Las columnas se rellenan con valores nulos, los cuales son tratados posteriormente.

3.3.2. Transformaciones Clave y Lógica de Negocio

Se aplicaron varias transformaciones para cumplir con los requisitos del esquema de salida:

- **Eliminación de Columnas Redundantes (Primer `Select`):** Inmediatamente después de la cadena de `Joins`, se añadió una transformación `Select`. Se activó la opción de `Auto mapping` para eliminar automáticamente las columnas `PID` duplicadas, evitando errores de ambigüedad.
- **Agregación por Patrón de Rama:** Para calcular `FullBath`, `HalfBath` y `Bedroom` a partir de la tabla `FloorDetail`, se implementó un patrón de agregación en una rama separada del flujo. El resultado se reunió con el flujo principal mediante un `Left Join`, evitando así la duplicación de datos.
- **Manejo Avanzado de Valores Nulos (`Derived Column`):** Se utilizó una única transformación `Derived Column` con dos `Column Patterns` para limpiar sistemáticamente los valores nulos:
  - **Cualitativas (string):**
    - *Condición:* `type=='string'`
    - *Expresión:* `iifNull(toString(byName($$)), 'NA')`
  - **Numéricas:**
    - *Condición:* `!(type=='string') !(type=='date') !(type=='binary')`
    - *Expresión:* `iifNull($$, 0)`
- **Limpieza Final y Estructuración (Segundo `Select`):** Cerca del final del flujo, se añadió una segunda transformación `Select`, configurada manualmente para:
  1. Eliminar la columna `_id` proveniente de MongoDB, la cual contenía una notación de objeto (`$oid`) que impedía la correcta exportación a CSV.

2. Renombrar columnas calculadas a sus nombres finales (ej. `TotalFullBath` a `FullBath`).
3. Reordenar las 81 columnas para que coincidieran exactamente con el esquema especificado.

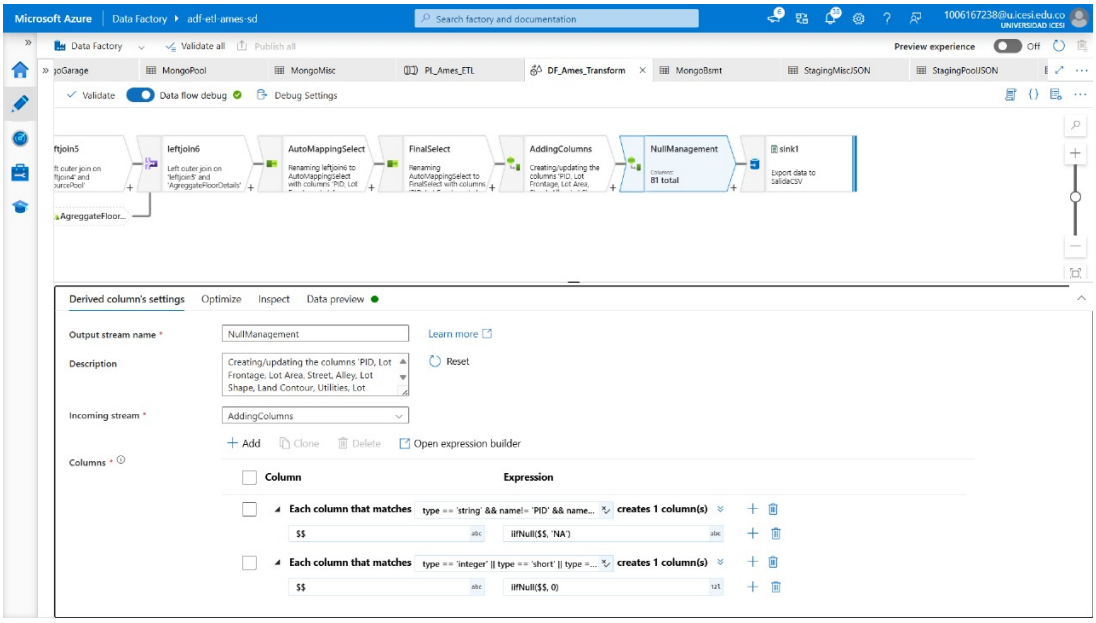


Figura 4: Configuración del `Column Pattern` para el manejo de nulos cualitativos.

Evidencia a añadir (Fig. 4): *Un pantallazo de la configuración de la transformación Derived Column mostrando una de las reglas de Column Pattern.*

3.4 Fase 4: Carga y Configuración del Destino (Sink)

El destino se configuró para escribir en un `Dataset` de tipo `DelimitedText` en el contenedor `salida`. Las configuraciones clave en la transformación `Sink` fueron:

- **File name option:** `Output to single file`, con el nombre `salida.csv`.
- **Optimize:** Se estableció el particionamiento a `Single partition` para asegurar que el resultado se escriba en un único archivo.

4 Resultados y Verificación

4.1 Ejecución y Monitoreo

La ejecución del pipeline se realizó en modo `Debug` durante el desarrollo y finalmente se publicó y ejecutó mediante un `Trigger now`. La ejecución completa tuvo una duración de **2 minutos y 34 segundos**, 46 segundos para el proceso de copiado de datos del staging, y 1m 48s para la ejecución del data flow.

Evidencia a añadir (Fig. 5): *Un pantallazo de la vista de monitoreo de ADF mostrando una corrida exitosa (Succeeded) con sus métricas principales.*

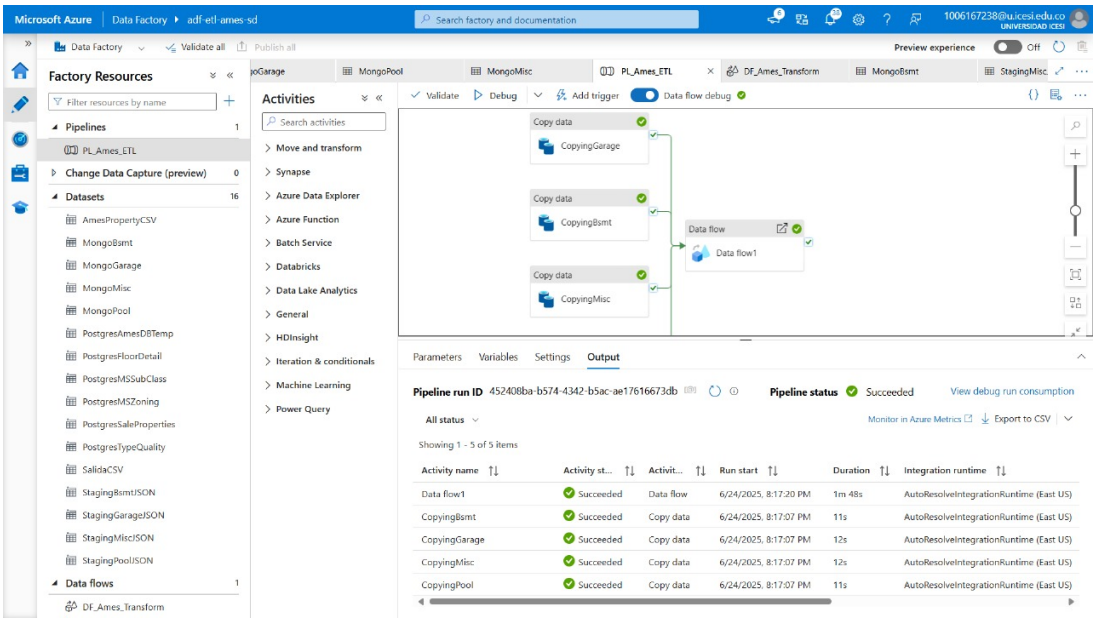


Figura 5: Detalles de una ejecución exitosa de la pipeline `PL_Ames_ETL` en la pestaña "Monitor"de ADF Studio.

4.2 Análisis del Archivo de Salida

Se verificó el archivo `salida.csv` en el contenedor `salida` del Blob Storage. El archivo resultante cumple con todas las especificaciones:

- Contiene un encabezado con los *81 nombres de columna* correctos, incluyendo el identificador principal `PID`.
- Tiene un total de *394 registros* de propiedades. Este número se mantuvo constante desde la primera transformación `Inner Join`, la cual actuó como un filtro sobre los 1000 registros iniciales del archivo `Property.csv`, seleccionando únicamente las propiedades con correspondencia en la base de datos relacional.
- Los datos reflejan las transformaciones aplicadas: no hay valores nulos visibles (reemplazados por `NA` o `0`), y las columnas calculadas están correctamente pobladas.

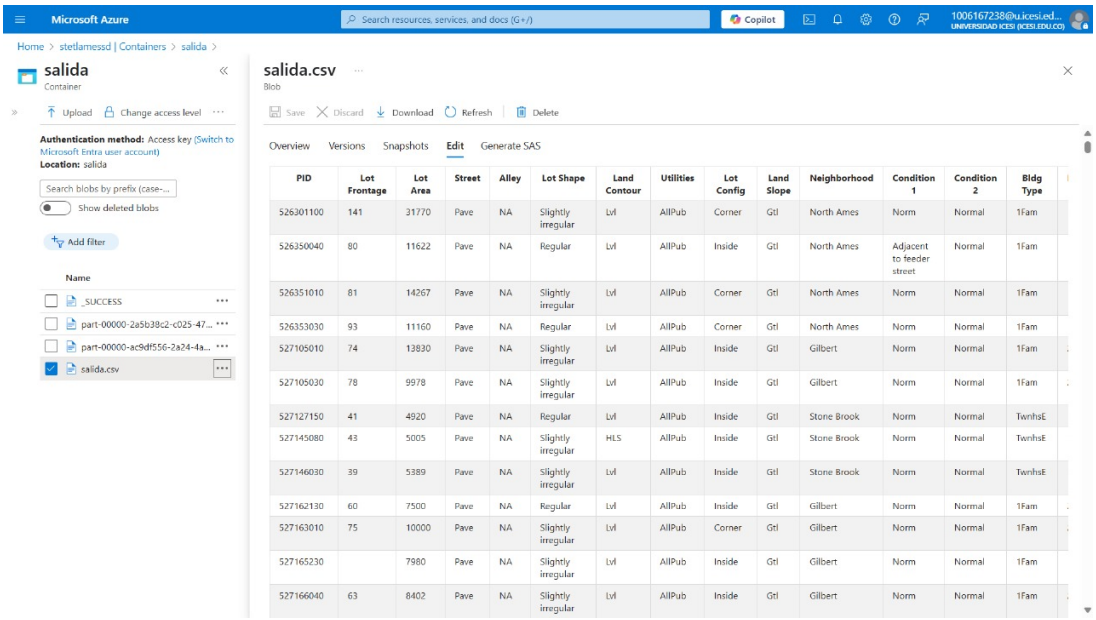
Evidencia a añadir (Fig. 6): *Un pantallazo del archivo salida.csv abierto en Excel o un editor de texto, mostrando las primeras filas y columnas para demostrar su estructura correcta.*

5 Conclusiones y Lecciones Aprendidas

Este proyecto sirvió como una aplicación práctica y comprensiva de los conceptos de ingeniería de datos en un entorno de nube moderno. La implementación exitosa del pipeline ETL valida la capacidad de Azure Data Factory como una herramienta potente y flexible para la integración de datos heterogéneos.

Las lecciones aprendidas más significativas incluyen:

1. **La Importancia del Diseño de Uniones:** La elección entre `Inner` y `Left Outer Join` no es trivial y tiene un impacto directo en la integridad del conjunto de datos final. Una mala elección puede llevar a la pérdida silenciosa de datos.



PID	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	Utilities	Lot Config	Land Slope	Neighborhood	Condition 1	Condition 2	Bldg Type
526301100	141	31770	Pave	NA	Slightly irregular	Lvl	AllPub	Corner	Gtl	North Ames	Norm	Normal	1Fam
526350040	80	11622	Pave	NA	Regular	Lvl	AllPub	Inside	Gtl	North Ames	Adjacent to feeder street	Normal	1Fam
526351010	81	14267	Pave	NA	Slightly irregular	Lvl	AllPub	Corner	Gtl	North Ames	Norm	Normal	1Fam
526353030	93	11160	Pave	NA	Regular	Lvl	AllPub	Corner	Gtl	North Ames	Norm	Normal	1Fam
527105010	74	13830	Pave	NA	Slightly irregular	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Normal	1Fam
527105030	78	9978	Pave	NA	Slightly irregular	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Normal	1Fam
527127150	41	4920	Pave	NA	Regular	Lvl	AllPub	Inside	Gtl	Stone Brook	Norm	Normal	TwnhsE
527145080	43	5005	Pave	NA	Slightly irregular	HLS	AllPub	Inside	Gtl	Stone Brook	Norm	Normal	TwnhsE
527146030	39	5389	Pave	NA	Slightly irregular	Lvl	AllPub	Inside	Gtl	Stone Brook	Norm	Normal	TwnhsE
527162130	60	7500	Pave	NA	Regular	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Normal	1Fam
527163010	75	10000	Pave	NA	Slightly irregular	Lvl	AllPub	Corner	Gtl	Gilbert	Norm	Normal	1Fam
527165230		7980	Pave	NA	Slightly irregular	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Normal	1Fam
527166040	63	8402	Pave	NA	Slightly irregular	Lvl	AllPub	Inside	Gtl	Gilbert	Norm	Normal	1Fam

Figura 6: Vista previa de las primeras filas y columnas del archivo final `salida.csv`.

2. **Adaptabilidad y Estrategias de Staging:** La imposibilidad de usar MongoDB directamente en el `Data Flow` nos forzó a adoptar un patrón de *staging*, una técnica común en ingeniería de datos para pre-procesar o estabilizar datos antes de la transformación principal.

3. **Eficiencia en la Transformación:** El uso de `Column Patterns` para el manejo de nulos demostró ser una técnica mucho más escalable y mantenible que crear reglas individuales por columna, destacando la importancia de buscar soluciones genéricas a problemas repetitivos.

4. **Gestión de Incompatibilidades y Errores:** Superamos desafíos técnicos como la incompatibilidad de la versión del conector de PostgreSQL, los errores por columnas duplicadas después de los `Joins`, y el formato de la columna `_id` de MongoDB, lo que subraya la naturaleza iterativa y de resolución de problemas de la ingeniería de datos.

5. **El Poder de la Orquestación Visual:** ADF permite abstraer la complejidad de la programación de Spark subyacente, permitiendo a los ingenieros de datos centrarse en la lógica de negocio de la transformación en lugar de en la infraestructura.

En resumen, el proyecto no solo cumplió con sus objetivos técnicos, sino que también proporcionó una valiosa experiencia en la resolución de problemas y la toma de decisiones de diseño en un contexto de ingeniería de datos del mundo real.

Referencias

[1] Villota Gómez, A. (2025). *Enunciado Proyecto: Proceso de Extracción, Transformación y Carga*. Documento de curso, Infraestructura y Arquitectura de TI. Universidad Icesi.

[2] Villota Gómez, A. (2025). *Datos de Conexión 2025-1*. Documento de curso. Universidad Icesi.

[3] Rojas, M. & Villota Gómez, A. (2024). *Práctica ETL utilizando Azure Data Factory*. Guía de laboratorio. Universidad Icesi.