

Problem Set 3

Ejecución del docker compose, ingreso y explicaciones sobre Jupyter Notebook - Spark

Para correr este proyecto se debe tener una carpeta y dentro de ella tener dos componentes (luego se le añade un tercero que es el archivo .env y el .env.example): una carpeta llamada work y el docker compose

Una vez se tiene esto se aplican los siguientes comandos:

```
docker compose up -d
docker compose exec pyspark-notebook pyspark
```

Se utiliza esto para ver el token

```
docker compose logs pyspark-notebook | Select-String -Pattern "token="
```

Sin embargo, sale esto:

```
pyspark-notebook | [I 2025-10-16 04:01:13.215 ServerApp] http://2fc795017f2
1:8888/lab?token=...
pyspark-notebook | [I 2025-10-16 04:01:13.215 ServerApp]
http://127.0.0.1:8888/lab?token=...
```

Aquí vemos que no hay un token como tal, pero en el docker compose se definió que el token iba a ser 'datamining1234'. Para abrir el Jupyter Notebook con Spark una vez que se corre el contenedor se utilizala siguiente URL:

```
http://127.0.0.1:8888/?token=datamining1234
```

Esta URL se puede poner en Microsoft Edge por ejemplo y ahí aparecerá el notebook. Ingresar directamente con el token asegura que no tengan que ponerse

credenciales.

Al docker compose en "environment" se añadieron variables que son las credenciales para ingresar a Snowflake. No están explícitamente, sino que se traen directamente desde el archivo .env (el cual será puesto en gitignore) para que las credenciales no estén expuestas directamente. Se creó un archivo .env.example que muestra como se ven las variables de entorno:

```
SF_ACCOUNT=your_account_name
SF_USER=your_username
SF_PASSWORD=your_password
SF_WAREHOUSE=COMPUTE_WH
SF_DATABASE=NYC_TAXI
SF_ROLE=ACCOUNTADMIN
```

En Jupyter hay dos carpetas creadas, una llamada "work" y la otra llamada "jars". En la carpeta "jars" están los archivos .jar necesarios para conectar Spark con Snowflake.

En el Jupyter incluiré dentro de la carpeta "work" un notebook adicional llamado "pruebas_conexion", en el cual se encuentra como cree la carpeta de jars y como instalé los archivos .jar para conectar con Snowflake.

Creación de la base de datos y primer esquema

Se crea la base de datos en Snowflake llamada NYC_TAXI. Dentro de esta base de datos se creó el esquema primer RAW.

Dentro del esquema RAW se crearon tres tablas. Dos correspondientes a los datos en sí de taxis amarillo (NYC_YELLOW_TAXIS) y taxis verdes (NYC_GREEN_TAXIS), es decir, una tabla para cada uno. La otra tabla consolida los metadatos de ambos servicios amarillos y verdes (INGEST_AUDIT).

Esto se hizo para crear las tablas del esquema RAW, directamente en Snowflake:

```
--creacion de la tabla de taxis amarillos con los datos en sí
CREATE OR REPLACE TABLE RAW.NYC_YELLOW_TAXIS (
  -- Datos originales del Parquet (Yellow Taxi)
  VendorID          INTEGER,
  tpep_pickup_datetime  TIMESTAMP_NTZ,
```

```

tpep_dropoff_datetime  TIMESTAMP_NTZ,
passenger_count        FLOAT,
trip_distance          FLOAT,
RatecodeID             FLOAT,
store_and_fwd_flag     STRING,
PULocationID           INTEGER,
DOLocationID           INTEGER,
payment_type           INTEGER,
fare_amount            FLOAT,
extra                  FLOAT,
mta_tax                FLOAT,
tip_amount             FLOAT,
tolls_amount           FLOAT,
improvement_surcharge  FLOAT,
total_amount           FLOAT,
congestion_surcharge   FLOAT,
Airport_fee            FLOAT,
cbd_congestion_fee     FLOAT,

-- Metadatos obligatorios P3
run_id                 STRING,      -- ID único de carga (ej: 2025_10_15_yellow_
2015_01)
service_type           STRING,      -- 'yellow'
source_year            INT,         -- Año del lote
source_month           INT,         -- Mes del lote
ingested_at_utc        TIMESTAMP_NTZ, -- Fecha y hora de ingesta
source_path            STRING       -- Ruta o URL origen del Parquet
);

```

```

--creacion de la tabla de taxis verdes con los datos en sí
CREATE OR REPLACE TABLE RAW.NYC_GREEN_TAXIS (
-- Datos originales del Parquet
VendorID              INTEGER,
lpep_pickup_datetime  TIMESTAMP_NTZ,
lpep_dropoff_datetime TIMESTAMP_NTZ,

```

```

store_and_fwd_flag    STRING,
RatecodeID           FLOAT,
PULocationID         INTEGER,
DOLocationID         INTEGER,
passenger_count       FLOAT,
trip_distance         FLOAT,
fare_amount           FLOAT,
extra                 FLOAT,
mta_tax               FLOAT,
tip_amount            FLOAT,
tolls_amount          FLOAT,
ehail_fee              FLOAT,
improvement_surcharge FLOAT,
total_amount          FLOAT,
payment_type          FLOAT,
trip_type             FLOAT,
congestion_surcharge  FLOAT,
cbd_congestion_fee    FLOAT,

-- Metadatos obligatorios
run_id                STRING,      -- ID único de corrida
service_type          STRING,      -- 'green'
source_year           INT,         -- Año del lote
source_month          INT,         -- Mes del lote
ingested_at_utc       TIMESTAMP_NTZ, -- Momento de ingesta
source_path           STRING       -- Ruta o URL del archivo Parquet
);

```

--crear tabla de metadatos consolidados de ambos servicios (amarillos y verdes)

```

CREATE OR REPLACE TABLE RAW.INGEST_AUDIT (
  RUN_ID          STRING,      -- Identificador único de ingesta
  SERVICE_TYPE    STRING,      -- 'yellow' o 'green'
  SOURCE_YEAR     INT,         -- Año del lote
  SOURCE_MONTH    INT,         -- Mes del lote

```

```
INGESTED_AT_UTC    TIMESTAMP_NTZ,  -- Fecha/hora de carga
SOURCE_PATH        STRING,        -- Ruta o URL del Parquet
ROW_COUNT          INT             -- Conteo de filas del lote
);
```

Tablas del esquema RAW y sus columnas y filas:

INGEST_AUDIT:

(7 columnas y 988 filas). Estas son las columnas:

```
RUN_ID
SERVICE_TYPE
SOURCE_YEAR
SOURCE_MONTH
INGESTED_AT_UTC
SOURCE_PATH
ROW_COUNT
```

NYC_GREEN_TAXIS:

(27 columnas 68,045,597 filas). Estas son las columnas:

```
VENDORID
LPEP_PICKUP_DATETIME
LPEP_DROPOFF_DATETIME
STORE_AND_FWD_FLAG
RATECODEID
PULOCATIONID
DOLOCATIONID
PASSENGER_COUNT
TRIP_DISTANCE
FARE_AMOUNT
EXTRA
MTA_TAX
TIP_AMOUNT
```

TOLLS_AMOUNT
EHAIL_FEE
IMPROVEMENT_SURCHARGE
TOTAL_AMOUNT
PAYMENT_TYPE
TRIP_TYPE
CONGESTION_SURCHARGE
CBD_CONGESTION_FEE
RUN_ID
SERVICE_TYPE
SOURCE_YEAR
SOURCE_MONTH
INGESTED_AT_UTC
SOURCE_PATH

NYC_YELLOW_TAXIS:

(26 columnas y 702,073,996 filas). Estas son las columnas:

VENDORID
TPEP_PICKUP_DATETIME
TPEP_DROPOFF_DATETIME
PASSENGER_COUNT
TRIP_DISTANCE
RATECODEID
STORE_AND_FWD_FLAG
PULOCATIONID
DOLOCATIONID
PAYMENT_TYPE
FARE_AMOUNT
EXTRA
MTA_TAX
TIP_AMOUNT
TOLLS_AMOUNT
IMPROVEMENT_SURCHARGE
TOTAL_AMOUNT

CONGESTION_SURCHARGE
AIRPORT_FEE
CBD_CONGESTION_FEE
RUN_ID
SERVICE_TYPE
SOURCE_YEAR
SOURCE_MONTH
INGESTED_AT_UTC
SOURCE_PATH

Ingesta de datos, Notebook01_ingesta_parquet_raw.ipynb

Ocurrió un error para ingestar los datos directamente con los URLs de nyc taxis database, por lo que se tuvieron que descargar los archivos localmente primero antes de ingestar. Esto se lo hizo en un notebook adicional llamado 00_descarga_parquets_local.ipynb. Luego de estas descargas en la carpeta local /home/work se siguió a ingestar los datos a Snowflake.

A continuación se muestran capturas de la ingesta en el notebook 01 con taxis amarillos, para varios años:

```
=====
[Icono de carpeta] Procesando 2015-06 -> /home/work/yellow_2015_06.parquet
[Icono de documento] Leyendo Parquet local: /home/work/yellow_2015_06.parquet
[Icono de documento] Filas totales: 12324936 -> Chunks de 1000000: 13
[Icono de flecha verde] Chunk 1/13 -> filas [1, 1000000] -> RUN_ID=YEL_2015_06_0001
[Icono de flecha verde] Chunk 2/13 -> filas [1000001, 2000000] -> RUN_ID=YEL_2015_06_0002
[Icono de flecha verde] Chunk 3/13 -> filas [2000001, 3000000] -> RUN_ID=YEL_2015_06_0003
[Icono de flecha verde] Chunk 4/13 -> filas [3000001, 4000000] -> RUN_ID=YEL_2015_06_0004
[Icono de flecha verde] Chunk 5/13 -> filas [4000001, 5000000] -> RUN_ID=YEL_2015_06_0005
[Icono de flecha verde] Chunk 6/13 -> filas [5000001, 6000000] -> RUN_ID=YEL_2015_06_0006
[Icono de flecha verde] Chunk 7/13 -> filas [6000001, 7000000] -> RUN_ID=YEL_2015_06_0007
[Icono de flecha verde] Chunk 8/13 -> filas [7000001, 8000000] -> RUN_ID=YEL_2015_06_0008
[Icono de flecha verde] Chunk 9/13 -> filas [8000001, 9000000] -> RUN_ID=YEL_2015_06_0009
[Icono de flecha verde] Chunk 10/13 -> filas [9000001, 10000000] -> RUN_ID=YEL_2015_06_0010
[Icono de flecha verde] Chunk 11/13 -> filas [10000001, 11000000] -> RUN_ID=YEL_2015_06_0011
[Icono de flecha verde] Chunk 12/13 -> filas [11000001, 12000000] -> RUN_ID=YEL_2015_06_0012
[Icono de flecha verde] Chunk 13/13 -> filas [12000001, 12324936] -> RUN_ID=YEL_2015_06_0013
[Icono de checkmark verde] Ingesta mensual chunked COMPLETADA con auditoría por chunk.
```

```

=====
[📁] Procesando 2017-12 → /home/work/yellow_2017_12.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2017_12.parquet
[📊] Filas totales: 9508501 → Chunks de 1000000: 10
[🌱] Chunk 1/10 → filas [1, 1000000] → RUN_ID=YEL_2017_12_0001
[🌱] Chunk 2/10 → filas [1000001, 2000000] → RUN_ID=YEL_2017_12_0002
[🌱] Chunk 3/10 → filas [2000001, 3000000] → RUN_ID=YEL_2017_12_0003
[🌱] Chunk 4/10 → filas [3000001, 4000000] → RUN_ID=YEL_2017_12_0004
[🌱] Chunk 5/10 → filas [4000001, 5000000] → RUN_ID=YEL_2017_12_0005
[🌱] Chunk 6/10 → filas [5000001, 6000000] → RUN_ID=YEL_2017_12_0006
[🌱] Chunk 7/10 → filas [6000001, 7000000] → RUN_ID=YEL_2017_12_0007
[🌱] Chunk 8/10 → filas [7000001, 8000000] → RUN_ID=YEL_2017_12_0008
[🌱] Chunk 9/10 → filas [8000001, 9000000] → RUN_ID=YEL_2017_12_0009
[🌱] Chunk 10/10 → filas [9000001, 9508501] → RUN_ID=YEL_2017_12_0010
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2018-01 → /home/work/yellow_2018_01.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2018_01.parquet
[📊] Filas totales: 8760687 → Chunks de 1000000: 9
[🌱] Chunk 1/9 → filas [1, 1000000] → RUN_ID=YEL_2018_01_0001
[🌱] Chunk 2/9 → filas [1000001, 2000000] → RUN_ID=YEL_2018_01_0002
[🌱] Chunk 3/9 → filas [2000001, 3000000] → RUN_ID=YEL_2018_01_0003
[🌱] Chunk 4/9 → filas [3000001, 4000000] → RUN_ID=YEL_2018_01_0004
[🌱] Chunk 5/9 → filas [4000001, 5000000] → RUN_ID=YEL_2018_01_0005
[🌱] Chunk 6/9 → filas [5000001, 6000000] → RUN_ID=YEL_2018_01_0006
[🌱] Chunk 7/9 → filas [6000001, 7000000] → RUN_ID=YEL_2018_01_0007
[🌱] Chunk 8/9 → filas [7000001, 8000000] → RUN_ID=YEL_2018_01_0008
[🌱] Chunk 9/9 → filas [8000001, 8760687] → RUN_ID=YEL_2018_01_0009
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2020-08 → /home/work/yellow_2020_08.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2020_08.parquet
[📊] Filas totales: 1007286 → Chunks de 1000000: 2
[🌱] Chunk 1/2 → filas [1, 1000000] → RUN_ID=YEL_2020_08_0001
[🌱] Chunk 2/2 → filas [1000001, 1007286] → RUN_ID=YEL_2020_08_0002
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2020-09 → /home/work/yellow_2020_09.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2020_09.parquet
[📊] Filas totales: 1341017 → Chunks de 1000000: 2
[🌱] Chunk 1/2 → filas [1, 1000000] → RUN_ID=YEL_2020_09_0001
[🌱] Chunk 2/2 → filas [1000001, 1341017] → RUN_ID=YEL_2020_09_0002
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2021-07 → /home/work/yellow_2021_07.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2021_07.parquet
[📊] Filas totales: 2821746 → Chunks de 1000000: 3
[🌱] Chunk 1/3 → filas [1, 1000000] → RUN_ID=YEL_2021_07_0001
[🌱] Chunk 2/3 → filas [1000001, 2000000] → RUN_ID=YEL_2021_07_0002
[🌱] Chunk 3/3 → filas [2000001, 2821746] → RUN_ID=YEL_2021_07_0003
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2021-08 → /home/work/yellow_2021_08.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2021_08.parquet
[📊] Filas totales: 2788757 → Chunks de 1000000: 3
[🌱] Chunk 1/3 → filas [1, 1000000] → RUN_ID=YEL_2021_08_0001
[🌱] Chunk 2/3 → filas [1000001, 2000000] → RUN_ID=YEL_2021_08_0002
[🌱] Chunk 3/3 → filas [2000001, 2788757] → RUN_ID=YEL_2021_08_0003
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2021-09 → /home/work/yellow_2021_09.parquet

```



```

=====
[📁] Procesando 2022-04 → /home/work/yellow_2022_04.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2022_04.parquet
[📊] Filas totales: 3599920 → Chunks de 1000000: 4
[🌱] Chunk 1/4 → filas [1, 1000000] → RUN_ID=YEL_2022_04_0001
[🌱] Chunk 2/4 → filas [1000001, 2000000] → RUN_ID=YEL_2022_04_0002
[🌱] Chunk 3/4 → filas [2000001, 3000000] → RUN_ID=YEL_2022_04_0003
[🌱] Chunk 4/4 → filas [3000001, 3599920] → RUN_ID=YEL_2022_04_0004
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2022-05 → /home/work/yellow_2022_05.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2022_05.parquet
[📊] Filas totales: 3588295 → Chunks de 1000000: 4
[🌱] Chunk 1/4 → filas [1, 1000000] → RUN_ID=YEL_2022_05_0001
[🌱] Chunk 2/4 → filas [1000001, 2000000] → RUN_ID=YEL_2022_05_0002
[🌱] Chunk 3/4 → filas [2000001, 3000000] → RUN_ID=YEL_2022_05_0003
[🌱] Chunk 4/4 → filas [3000001, 3588295] → RUN_ID=YEL_2022_05_0004
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2023-01 → /home/work/yellow_2023_01.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2023_01.parquet
[📊] Filas totales: 3066766 → Chunks de 1000000: 4
[🌱] Chunk 1/4 → filas [1, 1000000] → RUN_ID=YEL_2023_01_0001
[🌱] Chunk 2/4 → filas [1000001, 2000000] → RUN_ID=YEL_2023_01_0002
[🌱] Chunk 3/4 → filas [2000001, 3000000] → RUN_ID=YEL_2023_01_0003
[🌱] Chunk 4/4 → filas [3000001, 3066766] → RUN_ID=YEL_2023_01_0004
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2023-02 → /home/work/yellow_2023_02.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2023_02.parquet
[📊] Filas totales: 2913955 → Chunks de 1000000: 3
[🌱] Chunk 1/3 → filas [1, 1000000] → RUN_ID=YEL_2023_02_0001
[🌱] Chunk 2/3 → filas [1000001, 2000000] → RUN_ID=YEL_2023_02_0002
[🌱] Chunk 3/3 → filas [2000001, 2913955] → RUN_ID=YEL_2023_02_0003
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
📋 Resumen:
[✅] Meses OK : 16
[➡] Meses omitidos (sin archivo): 0
[❌] Meses con error : 0
[🏁] Terminado.

=====
[📁] Procesando 2023-07 → /home/work/yellow_2023_07.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2023_07.parquet
[📊] Filas totales: 2907108 → Chunks de 1000000: 3
[🌱] Chunk 1/3 → filas [1, 1000000] → RUN_ID=YEL_2023_07_0001
[🌱] Chunk 2/3 → filas [1000001, 2000000] → RUN_ID=YEL_2023_07_0002
[🌱] Chunk 3/3 → filas [2000001, 2907108] → RUN_ID=YEL_2023_07_0003
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[📁] Procesando 2023-08 → /home/work/yellow_2023_08.parquet
[📄] Leyendo Parquet local: /home/work/yellow_2023_08.parquet
[📊] Filas totales: 2824209 → Chunks de 1000000: 3
[🌱] Chunk 1/3 → filas [1, 1000000] → RUN_ID=YEL_2023_08_0001
[🌱] Chunk 2/3 → filas [1000001, 2000000] → RUN_ID=YEL_2023_08_0002
[🌱] Chunk 3/3 → filas [2000001, 2824209] → RUN_ID=YEL_2023_08_0003
[✅] Ingesta mensual chunked COMPLETADA con auditoría por chunk.
=====

```

```

=====
[Icon] Procesando 2024-12 → /home/work/yellow_2024_12.parquet
[Icon] Leyendo Parquet local: /home/work/yellow_2024_12.parquet
[Icon] Filas totales: 3668371 → Chunks de 1000000: 4
[Icon] Chunk 1/4 → filas [1, 1000000] → RUN_ID=YEL_2024_12_0001
[Icon] Chunk 2/4 → filas [1000001, 2000000] → RUN_ID=YEL_2024_12_0002
[Icon] Chunk 3/4 → filas [2000001, 3000000] → RUN_ID=YEL_2024_12_0003
[Icon] Chunk 4/4 → filas [3000001, 3668371] → RUN_ID=YEL_2024_12_0004
[Icon] Ingesta mensual chunked COMPLETADA con auditoría por chunk.





=====
[Icon] Procesando 2025-01 → /home/work/yellow_2025_01.parquet
[Icon] Leyendo Parquet local: /home/work/yellow_2025_01.parquet
[Icon] Filas totales: 3475226 → Chunks de 1000000: 4
[Icon] Chunk 1/4 → filas [1, 1000000] → RUN_ID=YEL_2025_01_0001
[Icon] Chunk 2/4 → filas [1000001, 2000000] → RUN_ID=YEL_2025_01_0002
[Icon] Chunk 3/4 → filas [2000001, 3000000] → RUN_ID=YEL_2025_01_0003
[Icon] Chunk 4/4 → filas [3000001, 3475226] → RUN_ID=YEL_2025_01_0004
[Icon] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[Icon] Procesando 2025-02 → /home/work/yellow_2025_02.parquet
[Icon] Leyendo Parquet local: /home/work/yellow_2025_02.parquet
[Icon] Filas totales: 3577543 → Chunks de 1000000: 4
[Icon] Chunk 1/4 → filas [1, 1000000] → RUN_ID=YEL_2025_02_0001
[Icon] Chunk 2/4 → filas [1000001, 2000000] → RUN_ID=YEL_2025_02_0002
[Icon] Chunk 3/4 → filas [2000001, 3000000] → RUN_ID=YEL_2025_02_0003
[Icon] Chunk 4/4 → filas [3000001, 3577543] → RUN_ID=YEL_2025_02_0004
[Icon] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

```

Aquí se observa que se hizo la ingesta por mes y por chunks de hasta millón de datos para los taxis amarillos.

No se insertaron datos de los meses de 3,4,5,6,7,8,9 de 2017. De los meses 4, 5, 6, y 7 se vio que los archivos parquet que se habían descargado localmente correspondientes a esos meses estaban vacíos, como se muestra a continuación:

 yellow_2017_04.parquet	16/10/2025 12:14	Archivo PARQUET	0 KB
 yellow_2017_05.parquet	16/10/2025 12:14	Archivo PARQUET	0 KB
 yellow_2017_06.parquet	16/10/2025 12:14	Archivo PARQUET	0 KB
 yellow_2017_07.parquet	16/10/2025 12:14	Archivo PARQUET	0 KB

En cambio, al intentar ingestar los meses 3, 8, 9 a Snowflake se obtuvo el siguiente error:

```

-----
[INFO] Procesando 2017-08 -> /home/work/yellow_2017_08.parquet
[INFO] Leyendo Parquet local: /home/work/yellow_2017_08.parquet
[ERROR] Error en 2017-08: An error occurred while calling o39.parquet.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 1.0 failed 1 times, most recent failure:
Lost task 0.0 in stage 1.0 (TID 1) (5f892c4bd431 executor driver): org.apache.spark.SparkException: Exception thrown in awaitR
esult:
    at org.apache.spark.util.SparkThreadUtils$.awaitResult(SparkThreadUtils.scala:56)
    at org.apache.spark.util.ThreadUtils$.awaitResult(ThreadUtils.scala:310)
    at org.apache.spark.util.ThreadUtils$.parmap(ThreadUtils.scala:383)
    at org.apache.spark.sql.execution.datasources.parquet.ParquetFileFormat$.readParquetFootersInParallel(ParquetFileForma
t.scala:443)
    at org.apache.spark.sql.execution.datasources.parquet.ParquetFileFormat$.anonfun$mergeSchemasInParallel$1(ParquetFile
Format.scala:493)
    at org.apache.spark.sql.execution.datasources.parquet.ParquetFileFormat$.anonfun$mergeSchemasInParallel$1$adapted(Par
quetFileFormat.scala:485)
    at org.apache.spark.sql.execution.datasources.SchemaMergeUtils$.anonfun$mergeSchemasInParallel$2(SchemaMergeUtils.sca
la:79)
    at org.apache.spark.rdd.RDD$.anonfun$mapPartitions$2(RDD.scala:855)
    at org.apache.spark.rdd.RDD$.anonfun$mapPartitions$2$adapted(RDD.scala:855)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:364)

```

Para los taxis verdes se hizo la ingesta también por mes, sin embargo, por chunks de hasta 500,000 datos porque estos datos eran considerablemente menos que los de amarillos.

Ahora se muestran capturas de la ingesta de taxis verdes a Snowflake, para varios años:

```

=====
[GREEN] [INFO] Procesando 2015-01 -> /home/work/green_2015_01.parquet
[GREEN] [INFO] Leyendo Parquet local: /home/work/green_2015_01.parquet
[GREEN] [INFO] Filas totales: 1508493 -> Chunks de 500000: 4
[GREEN] [INFO] Chunk 1/4 -> filas [1, 500000] -> RUN_ID=GRE_2015_01_0001
[GREEN] [INFO] Chunk 2/4 -> filas [500001, 1000000] -> RUN_ID=GRE_2015_01_0002
[GREEN] [INFO] Chunk 3/4 -> filas [1000001, 1500000] -> RUN_ID=GRE_2015_01_0003
[GREEN] [INFO] Chunk 4/4 -> filas [1500001, 1508493] -> RUN_ID=GRE_2015_01_0004
[GREEN] [INFO] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] [INFO] Procesando 2015-02 -> /home/work/green_2015_02.parquet
[GREEN] [INFO] Leyendo Parquet local: /home/work/green_2015_02.parquet
[GREEN] [INFO] Filas totales: 1574830 -> Chunks de 500000: 4
[GREEN] [INFO] Chunk 1/4 -> filas [1, 500000] -> RUN_ID=GRE_2015_02_0001
[GREEN] [INFO] Chunk 2/4 -> filas [500001, 1000000] -> RUN_ID=GRE_2015_02_0002
[GREEN] [INFO] Chunk 3/4 -> filas [1000001, 1500000] -> RUN_ID=GRE_2015_02_0003
[GREEN] [INFO] Chunk 4/4 -> filas [1500001, 1574830] -> RUN_ID=GRE_2015_02_0004
[GREEN] [INFO] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] [INFO] Procesando 2015-03 -> /home/work/green_2015_03.parquet
[GREEN] [INFO] Leyendo Parquet local: /home/work/green_2015_03.parquet
[GREEN] [INFO] Filas totales: 1722574 -> Chunks de 500000: 4
[GREEN] [INFO] Chunk 1/4 -> filas [1, 500000] -> RUN_ID=GRE_2015_03_0001
[GREEN] [INFO] Chunk 2/4 -> filas [500001, 1000000] -> RUN_ID=GRE_2015_03_0002
[GREEN] [INFO] Chunk 3/4 -> filas [1000001, 1500000] -> RUN_ID=GRE_2015_03_0003
[GREEN] [INFO] Chunk 4/4 -> filas [1500001, 1722574] -> RUN_ID=GRE_2015_03_0004
[GREEN] [INFO] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

```

```

=====
[GREEN] Procesando 2016-01 → /home/work/green_2016_01.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2016_01.parquet
[GREEN] Filas totales: 1445292 → Chunks de 500000: 3
[GREEN] Chunk 1/3 → filas [1, 500000] → RUN_ID=GRE_2016_01_0001
[GREEN] Chunk 2/3 → filas [500001, 1000000] → RUN_ID=GRE_2016_01_0002
[GREEN] Chunk 3/3 → filas [1000001, 1445292] → RUN_ID=GRE_2016_01_0003
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Resumen:
  ✓ Meses OK : 13
  ➡ Meses omitidos (sin archivo): 0
  ✗ Meses con error : 0
[GREEN] Terminado.

```

Primero se cargaron los primeros 13 meses de los taxis verdes, luego se cargó el resto:

```

=====
[GREEN] Procesando 2017-11 → /home/work/green_2017_11.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2017_11.parquet
[GREEN] Filas totales: 874173 → Chunks de 500000: 2
[GREEN] Chunk 1/2 → filas [1, 500000] → RUN_ID=GRE_2017_11_0001
[GREEN] Chunk 2/2 → filas [500001, 874173] → RUN_ID=GRE_2017_11_0002
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2017-12 → /home/work/green_2017_12.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2017_12.parquet
[GREEN] Filas totales: 906016 → Chunks de 500000: 2
[GREEN] Chunk 1/2 → filas [1, 500000] → RUN_ID=GRE_2017_12_0001
[GREEN] Chunk 2/2 → filas [500001, 906016] → RUN_ID=GRE_2017_12_0002
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2018-01 → /home/work/green_2018_01.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2018_01.parquet
[GREEN] Filas totales: 792744 → Chunks de 500000: 2
[GREEN] Chunk 1/2 → filas [1, 500000] → RUN_ID=GRE_2018_01_0001
[GREEN] Chunk 2/2 → filas [500001, 792744] → RUN_ID=GRE_2018_01_0002
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2019-12 → /home/work/green_2019_12.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2019_12.parquet
[GREEN] Filas totales: 455294 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 455294] → RUN_ID=GRE_2019_12_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2020-01 → /home/work/green_2020_01.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2020_01.parquet
[GREEN] Filas totales: 447770 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 447770] → RUN_ID=GRE_2020_01_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2020-02 → /home/work/green_2020_02.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2020_02.parquet
[GREEN] Filas totales: 398632 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 398632] → RUN_ID=GRE_2020_02_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

```

```

=====
[GREEN] Procesando 2021-12 → /home/work/green_2021_12.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2021_12.parquet
[GREEN] Filas totales: 99961 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 99961] → RUN_ID=GRE_2021_12_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2022-01 → /home/work/green_2022_01.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2022_01.parquet
[GREEN] Filas totales: 62495 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 62495] → RUN_ID=GRE_2022_01_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2022-12 → /home/work/green_2022_12.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2022_12.parquet
[GREEN] Filas totales: 72439 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 72439] → RUN_ID=GRE_2022_12_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2023-01 → /home/work/green_2023_01.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2023_01.parquet
[GREEN] Filas totales: 68211 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 68211] → RUN_ID=GRE_2023_01_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2024-12 → /home/work/green_2024_12.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2024_12.parquet
[GREEN] Filas totales: 53994 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 53994] → RUN_ID=GRE_2024_12_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2025-01 → /home/work/green_2025_01.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2025_01.parquet
[GREEN] Filas totales: 48326 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 48326] → RUN_ID=GRE_2025_01_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Procesando 2025-08 → /home/work/green_2025_08.parquet
[GREEN] Leyendo Parquet local: /home/work/green_2025_08.parquet
[GREEN] Filas totales: 46306 → Chunks de 500000: 1
[GREEN] Chunk 1/1 → filas [1, 46306] → RUN_ID=GRE_2025_08_0001
[GREEN] Ingesta mensual chunked COMPLETADA con auditoría por chunk.

=====
[GREEN] Resumen:
  [GREEN] Meses OK      : 115
  [GREEN] Meses omitidos (sin archivo): 0
  [GREEN] Meses con error      : 0
[GREEN] Terminado.

```

Es importante notar que en el notebook de ingesta va a salir como que si se ingestaron los datos de manera de corrida para ambos tipos (verdes y amarillos).

Sin embargo, el proceso fue por partes y sobre todo para taxis amarillos se hizo una ingesta de año por año (solo cambiándose esta parte:

Rango solicitado

START_YEAR, START_MONTH = 2015, 1

END_YEAR, END_MONTH = 2025, 8

Matriz de cobertura

Matriz de Cobertura Yellow											
	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025
1	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
3	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK
4	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK
5	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK
6	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK
7	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK
8	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	OK
9	OK	OK	FALTANTE	OK	OK	OK	OK	OK	OK	OK	FALTANTE
10	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE
11	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE
12	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE

Matriz de Cobertura Green											
	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025
1	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
2	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
3	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
4	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
5	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
6	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
7	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
8	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK
9	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE
10	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE
11	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE
12	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	FALTANTE

Esquema Silver, 02_enriquecimiento_y_unificacion.ipynb

Para el segundo notebook (02_enriquecimiento_y_unificacion.ipynb), se creó un esquema SILVER en la base de datos NYC_TAXI. Este permitía unificar datos de taxis verdes y amarillos, así como descargar localmente el csv de taxi zones e insertar los datos en Snowflake.

El esquema SILVER se lo hizo directamente en Snowflake, así:

```
CREATE SCHEMA SILVER;
```

Sin embargo, todas las tablas se crearon directamente desde el Notebook 02_enriquecimiento_y_unificacion. Se creó la tabla NYC_ALL_TAXITRIPS que junta los datos de los taxis amarillos y verdes de las tablas NYC_GREEN_TAXIS y NYC_YELLOW_TAXIS del esquema RAW. Además, en el esquema SILVER se añadió la tabla de TAXI_ZONES. Por último se creó la tabla NYC_ALL_TAXIS_ENRICHED que es tiene las mismas filas que NYC_ALL_TAXITRIPS pero se añaden las columnas de la tabla de TAXI_ZONES (pickup_borough, pickup_zone, dropoff_borough, dropoff_zone). A continuación se muestran varias capturas de ejecución de este notebook:

```
✓ YELLOW 2016-02 - filas insertadas: 11183
Insertando GREEN 2016-02 ...
✓ GREEN 2016-02 - filas insertadas: 1510722
Insertando YELLOW 2016-03 ...
✓ YELLOW 2016-03 - filas insertadas: 12203824
Insertando GREEN 2016-03 ...
✓ GREEN 2016-03 - filas insertadas: 1576393
Insertando YELLOW 2016-04 ...
✓ YELLOW 2016-04 - filas insertadas: 11927996
Insertando GREEN 2016-04 ...
✓ GREEN 2016-04 - filas insertadas: 1543926
Insertando YELLOW 2016-05 ...
✓ YELLOW 2016-05 - filas insertadas: 11832049
Insertando GREEN 2016-05 ...
✓ GREEN 2016-05 - filas insertadas: 1536979
Insertando YELLOW 2016-06 ...
✓ YELLOW 2016-06 - filas insertadas: 11131645
Insertando GREEN 2016-06 ...
✓ GREEN 2016-06 - filas insertadas: 1404727
Insertando YELLOW 2016-07 ...
✓ YELLOW 2016-07 - filas insertadas: 10294080
Insertando GREEN 2016-07 ...
✓ GREEN 2016-07 - filas insertadas: 1332510
```

Este es un ejemplo de que no se insertan las filas que no se tienen (ciertos meses de 2017):


```

-----
🔥 Insertando YELLOW 2017-03 ...
✅ YELLOW 2017-03 - filas insertadas: 0
🔥 Insertando GREEN 2017-03 ...
✅ GREEN 2017-03 - filas insertadas: 1157827
🔥 Insertando YELLOW 2017-04 ...
✅ YELLOW 2017-04 - filas insertadas: 0
🔥 Insertando GREEN 2017-04 ...
✅ GREEN 2017-04 - filas insertadas: 1080844
🔥 Insertando YELLOW 2017-05 ...
✅ YELLOW 2017-05 - filas insertadas: 0
🔥 Insertando GREEN 2017-05 ...
✅ GREEN 2017-05 - filas insertadas: 1059463
🔥 Insertando YELLOW 2017-06 ...
✅ YELLOW 2017-06 - filas insertadas: 0
🔥 Insertando GREEN 2017-06 ...
✅ GREEN 2017-06 - filas insertadas: 976467
🔥 Insertando YELLOW 2017-07 ...
✅ YELLOW 2017-07 - filas insertadas: 0
🔥 Insertando GREEN 2017-07 ...
✅ GREEN 2017-07 - filas insertadas: 914783
🔥 Insertando YELLOW 2017-08 ...
✅ YELLOW 2017-08 - filas insertadas: 0
🔥 Insertando GREEN 2017-08 ...
✅ GREEN 2017-08 - filas insertadas: 867407
🔥 Insertando YELLOW 2017-09 ...
✅ YELLOW 2017-09 - filas insertadas: 0

```

Así sale la creación de la tabla NYC_ALL_TAXITRIPS_ENRICHED

```

🌱 Construyendo SILVER.NYC_ALL_TAXIS_ENRICHED ...
✅ SILVER.NYC_ALL_TAXIS_ENRICHED creada.
📦 Total filas en SILVER.NYC_ALL_TAXITRIPS: 770119593
📦 Total filas en SILVER.NYC_ALL_TAXIS_ENRICHED: 770119593

```

Tablas del esquema SILVER y sus columnas y filas:

NYC_ALL_TAXITRIPS:

(32 columnas y 770,119,593 filas)

```

HASH_KEY
PICKUP_DATETIME
DROPOFF_DATETIME
VENDORID

```


VENDOR_DESC
RATECODEID
RATE_CODE_DESC
PASSENGER_COUNT
TRIP_DISTANCE
PULOCATIONID
DOLOCATIONID
STORE_AND_FWD_FLAG
PAYMENT_TYPE
PAYMENT_TYPE_DESC
FARE_AMOUNT
EXTRA
MTA_TAX
TIP_AMOUNT
TOLLS_AMOUNT
IMPROVEMENT_SURCHARGE
TOTAL_AMOUNT
CONGESTION_SURCHARGE
CBD_CONGESTION_FEE
AIRPORT_FEE
EHAIL_FEE
TRIP_TYPE
SERVICE_TYPE
SOURCE_YEAR
SOURCE_MONTH
RUN_ID
INGESTED_AT_UTC
SOURCE_PATH

TAXI_ZONES:

(4 columnas y 265 filas)

LOCATIONID
BOROUGH

ZONE
SERVICE_ZONE

NYC_ALL_TAXIS_ENRICHED:

(36 columnas y 770,119,593 filas)

HASH_KEY
PICKUP_DATETIME
DROPOFF_DATETIME
VENDORID
VENDOR_DESC
RATECODEID
RATE_CODE_DESC
PASSENGER_COUNT
TRIP_DISTANCE
PULOCATIONID
DOLOCATIONID
STORE_AND_FWD_FLAG
PAYMENT_TYPE
PAYMENT_TYPE_DESC
FARE_AMOUNT
EXTRA
MTA_TAX
TIP_AMOUNT
TOLLS_AMOUNT
IMPROVEMENT_SURCHARGE
TOTAL_AMOUNT
CONGESTION_SURCHARGE
CBD_CONGESTION_FEE
AIRPORT_FEE
EHAIL_FEE
TRIP_TYPE
SERVICE_TYPE
SOURCE_YEAR
SOURCE_MONTH

```
RUN_ID
INGESTED_AT_UTC
SOURCE_PATH
PICKUP_BOROUGH
PICKUP_ZONE
DROPOFF_BOROUGH
DROPOFF_ZONE
```

Normalización de payment_type, vendor, rate_code:

Para normalizar estos datos se mantuvieron las columnas sin normalizar de las tablas del esquema RAW en las tablas de NYC_ALL_TAXITRIPS y NYC_TAXIS_ENRICHED, pero se añadieron columnas de descripción para normalizar.

Para **payment_type** se tuvo lo siguiente:

Código	Significado
1	Credit Card
2	Cash
3	No Charge
4	Dispute
5	Unknown
6	Voided Trip

Aquí se muestra un ejemplo al ejecutar el siguiente query en Snowflake:

```
select payment_type, payment_type_desc from silver.nyc_all_taxis_enriched
limit 100
```

	# PAYMENT_TYPE		A PAYMENT_TYPE_DESC
1		1	Credit card
2		1	Credit card
3		2	Cash
4		2	Cash
5		2	Cash
6		1	Credit card
7		2	Cash

Para **rate_code** se tuvo lo siguiente

Código	Significado
1	Standard rate
2	JFK
3	Newark
4	Nassau/Westchester
5	Negotiated fare
6	Group ride

Aquí se muestra un ejemplo al ejecutar el siguiente query en Snowflake:

```
select ratecodeid, rate_code_desc from silver.nyc_all_taxis_enriched
limit 100
```

	# RATECODEID	A RATE_CODE_DESC
1	1	Standard rate
2	1	Standard rate
3	1	Standard rate
4	1	Standard rate
5	1	Standard rate
6	1	Standard rate
7	1	Standard rate

Para **vendor** se tuvo lo siguiente:

Código	Vendor Original
1	Creative Mobile Technologies (CMT)
2	VeriFone Transportation Systems (VTS)
4	DDS
5	VIP
Otro	Unknown

Aquí se muestra un ejemplo al ejecutar el siguiente query en Snowflake:

```
select vendorid, vendor_desc from silver.nyc_all_taxis_enriched
limit 100
```

	# VENDORID	A VENDOR_DESC
1	1	Creative Mobile Technologies (CMT)
2	2	VeriFone Transportation Systems (VTS)
3	2	VeriFone Transportation Systems (VTS)
4	2	VeriFone Transportation Systems (VTS)
5	2	VeriFone Transportation Systems (VTS)
6	1	Creative Mobile Technologies (CMT)
7	2	VeriFone Transportation Systems (VTS)

Construcción de la tabla OBT y esquema Analytics, Notebook 03_construccion_obt.ipynb

El esquema ANALYTICS y la tabla OBT_TRIPS se crearon desde el script. Al terminar la ejecución, se obtiene este mensaje:

✅ OBT_TRIPS generada. Total filas: 770119593

La tabla OBT_TRIPS tiene 41 columnas, que son las siguientes:

PICKUP_DATETIME
DROPOFF_DATETIME
PICKUP_DATE
PICKUP_HOUR
DROPOFF_HOUR
DAY_OF_WEEK
MONTH
YEAR
PU_LOCATION_ID
PU_ZONE
PU_BOROUGH
DO_LOCATION_ID
DO_ZONE
DO_BOROUGH
SERVICE_TYPE
VENDOR_ID

```
VENDOR_NAME
RATE_CODE_ID
RATE_CODE_DESC
PAYMENT_TYPE
PAYMENT_TYPE_DESC
TRIP_TYPE
PASSENGER_COUNT
TRIP_DISTANCE
STORE_AND_FWD_FLAG
FARE_AMOUNT
EXTRA
MTA_TAX
TIP_AMOUNT
TOLLS_AMOUNT
IMPROVEMENT_SURCHARGE
CONGESTION_SURCHARGE
AIRPORT_FEE
TOTAL_AMOUNT
TRIP_DURATION_MIN
AVG_SPEED_MPH
TIP_PCT
RUN_ID
INGESTED_AT_UTC
SOURCE_YEAR
SOURCE_MONTH
```

Aparte de la creación del esquema y la tabla, se hace una prueba o verificación de idempotencia. Primero, genera un HASH_KEY único por cada viaje usando campos clave como pickup_datetime, dropoff_datetime, VendorID, PULocationID, DOLocationID y total_amount junto con el tipo de servicio (YELLOW o GREEN). Este hash representa la identidad natural del viaje, por lo cual dos registros exactamente iguales producen el mismo HASH_KEY. Después, el código utiliza un INSERT con condición WHERE NOT EXISTS, que compara ese HASH_KEY contra la tabla destino SILVER.NYC_ALL_TAXITRIPS:

```
WHERE NOT EXISTS (SELECT 1 FROM SILVER.NYC_ALL_TAXITRIPS T WHERE  
T.HASH_KEY = S.HASH_KEY)
```

Esto impide que un viaje ya insertado vuelva a ser cargado. En el notebook también hay una celda de test de idempotencia, donde se reingesta 2017 mes 10. Cuando se hace esto, la primera carga inserta filas, pero la segunda no encuentra viajes nuevos y no agrega nada.

Finalmente, en el test de idempotencia, contamos filas antes y después de dos inserciones consecutivas del mismo mes. Si el número de filas no cambia, significa que no hubo duplicados, verificando la idempotencia.

Así se ve la ejecución del test de idempotencia:

```
🔍 Probando idempotencia para 2017-10  
📊 Filas iniciales (antes de carga): 10694409  
🟡 Insertando YELLOW de prueba...  
✅ YELLOW insert OK  
🟢 Insertando GREEN de prueba...  
✅ GREEN insert OK  
📊 Filas después del primer insert: 10694409  
🔄 Reintentando carga del mismo mes (idempotencia test)...  
📊 Filas después del segundo insert: 10694409  
  
🟢 ✅ IDEMPOTENCIA COMPROBADA - No se duplicaron filas.
```

Validaciones de la tabla OBT_TRIPS, 04_validaciones_y_exploracion

Se hizo un control de calidad sobre la tabla ANALYTICS.OBT_TRIPS, con el objetivo de garantizar que los datos estén en condiciones aptas para análisis. Para esto, en el esquema ANALYTICS se creó otra tabla para registrar las validaciones, llamada DATA_QUALITY_LOG. El objetivo es detectar y registrar problemas comunes en los datos de viajes:

Tipo de Validación	Riesgo Detectado
Nulos críticos	Campos esenciales vacíos (sin pickup, tarifa, etc.)
Rangos lógicos	Distancias negativas, montos fuera de rango
Coherencia temporal	Viajes con fecha de salida antes de la llegada

Tipo de Validación	Riesgo Detectado
Distribución de viajes (INFO)	Asegurar que no faltan meses o servicios

Se tomó un enfoque “tolerante” para hacer la validaciones.

Categoría	Umbral	Ejemplo de fallo
NULL	≤ 100 permitidos	Si 101 viajes no tienen pickup_datetime → FAIL
RANGE	≤ 500 permitidos	Si 800 distancias > 200 km → FAIL
TIME	0 permitidos	Si 1 viaje tiene pickup >= dropoff → FAIL
COUNT	No aplica	Solo reporta distribución

Esto quiere decir que permite errores menores sin dar fallo. En la tabla de validaciones (DATA_QUALITY_LOG) una parte importante son las filas que tienen como valor de la columna Check_Status PASS o FAIL. Estas filas demuestran tipos de errores como nulos, errores de fuera de rango, tiempo y repetición. A continuación se muestran las filas:

Check_Name	Check_Type	Check_Value
null_pickup_datetime	NULL	0
null_dropoff_datetime	NULL	0
null_pu_location_id	NULL	0
null_do_location_id	NULL	0
null_passenger_count	NULL	18408586
null_trip_distance	NULL	0
null_fare_amount	NULL	0
null_total_amount	NULL	0
null_service_type	NULL	0
null_vendor_id	NULL	0
null_payment_type	NULL	1913192
null_rate_code_id	NULL	18408586
range_trip_distance	RANGE	43497
range_passenger_count	RANGE	9236
range_total_amount	RANGE	2553660

range_tip_amount_vs_fare	RANGE	4199007
time_pickup_before_dropoff	TIME	1037412
null_pickup_datetime	NULL	0
null_dropoff_datetime	NULL	0
null_pu_location_id	NULL	0
null_do_location_id	NULL	0
null_passenger_count	NULL	18408586
null_trip_distance	NULL	0
null_fare_amount	NULL	0
null_total_amount	NULL	0
null_service_type	NULL	0
null_vendor_id	NULL	0
null_payment_type	NULL	1913192
null_rate_code_id	NULL	18408586
range_trip_distance	RANGE	43497
range_passenger_count	RANGE	9236
range_total_amount	RANGE	2553660
range_tip_amount_vs_fare	RANGE	4199007
time_pickup_before_dropoff	TIME	1037412

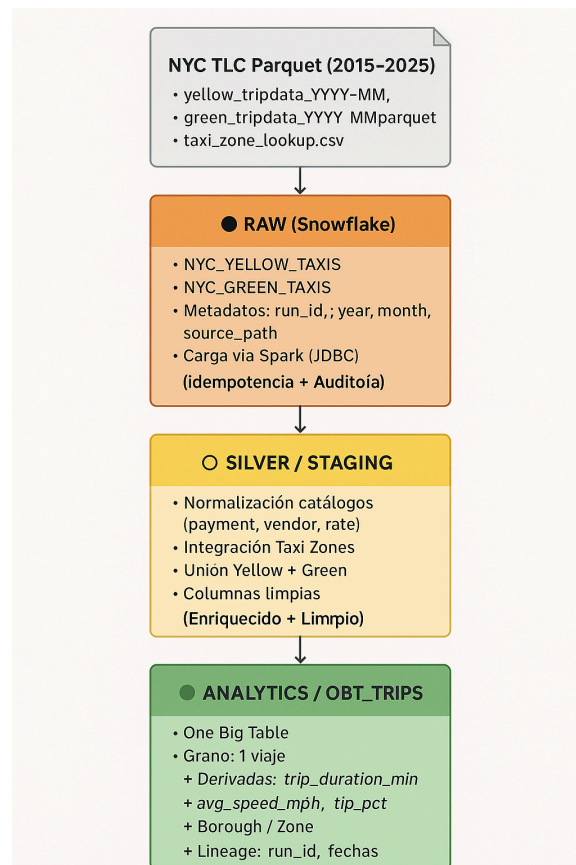
Estas vienen acompañadas de una columna llamada Check_Value, en la que se contabilizan los errores de cada tipo en toda la tabla de obt_trips. Haciendo una suma de todos estos tipos de errores con el query

```
select sum(check_value) from analytics.data_quality_log where check_status
= 'FAIL' or check_status = 'PASS'
```

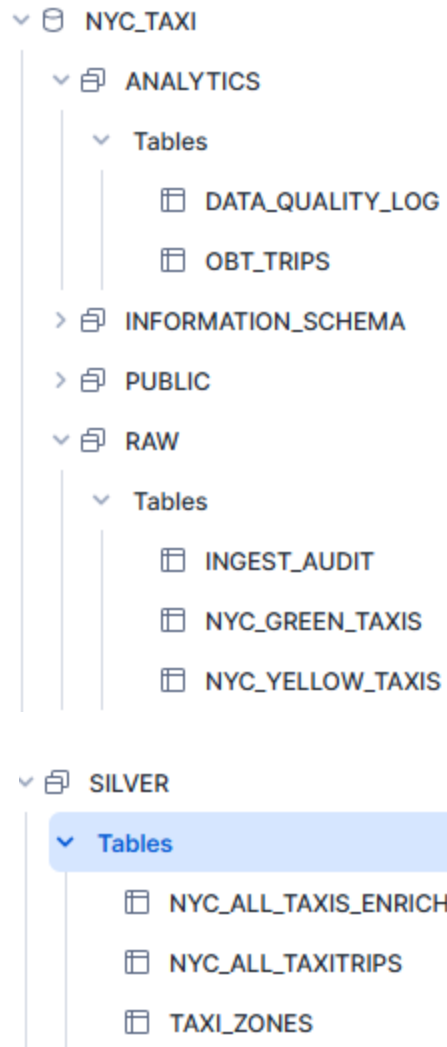
se obtuvo que hay 93,146,352 de errores en la tabla obt_trips. Por ejemplo, se encontró que 18,408,586 viajes tenían número nulo de pasajeros, algo que no es correcto. El resto de datos son recuentos de la cantidad de datos por mes, que es más algo informativo, es decir, no busca errores directamente sino más bien te dice cuántos viajes existen por año, mes y tipo de servicio. Esto fue con el

propósito de ver si había datos de otros años que no fueran 2015-2025, y se encontró que en efecto habían datos de otros años no seleccionados.

Esquema de lo que se realizó



Capturas de cómo quedó la base de datos en Snowflake:



Preguntas de negocio, notebook 05_data_analysis

a. Top 10 zonas de pickup por volumen mensual.

Se van a tomar en cuenta dos casos. El primero es uno en que las zonas con más pickups por volumen mensual se pueden repetir y el otro que no se pueden repetir. A continuación se toma el primer caso:

	PU_ZONE	PU_BOROUGH	YEAR	MONTH	TOTAL_TRIPS	GLOBAL_RANK
0	Upper East Side South	Manhattan	2015	4	497703	1
1	Upper East Side South	Manhattan	2015	5	484637	2
2	Upper East Side South	Manhattan	2015	3	473444	3
3	Upper East Side South	Manhattan	2015	10	468467	4
4	Upper East Side South	Manhattan	2016	5	465264	5
5	Midtown Center	Manhattan	2015	3	464162	6
6	Midtown Center	Manhattan	2015	4	460933	7
7	Upper East Side North	Manhattan	2015	4	458050	8
8	Midtown Center	Manhattan	2016	3	457857	9
9	Upper East Side South	Manhattan	2015	6	454707	10

Donde Upper East Side South, Upper East Side North y Midtown Center son las zonas con más pickups por volumen mensual.

A continuación se toma en segundo caso de no zonas repetidas:

	PU_ZONE	PU_BOROUGH	YEAR	MONTH	TOTAL_TRIPS	GLOBAL_RANK
0	Upper East Side South	Manhattan	2015	4	497703	1
1	Midtown Center	Manhattan	2015	3	464162	2
2	Upper East Side North	Manhattan	2015	4	458050	3
3	Times Sq/Theatre District	Manhattan	2015	3	442047	4
4	Midtown East	Manhattan	2015	3	441582	5
5	Murray Hill	Manhattan	2015	3	440112	6
6	East Village	Manhattan	2015	3	439620	7
7	Union Sq	Manhattan	2015	3	431348	8
8	Penn Station/Madison Sq West	Manhattan	2015	3	424726	9
9	Clinton East	Manhattan	2015	3	417905	10

En ambos casos los mayores números de pickups por zona se dieron en el 2015, mostrando que en ese año se realizaban más viajes en taxis.

b. Top 10 zonas de dropoff por volumen mensual.

Aquí se tomaron igualmente los dos casos de que se permiten zonas repetidas y no. A continuación se muestra el resultado del primer caso:

	DO_ZONE	DO_BOROUGH	YEAR	MONTH	TOTAL_TRIPS	GLOBAL_RANK
0	Midtown Center	Manhattan	2015	3	506987	1
1	Midtown Center	Manhattan	2015	4	502432	2
2	Midtown Center	Manhattan	2015	5	480660	3
3	Midtown Center	Manhattan	2015	6	475216	4
4	Upper East Side North	Manhattan	2015	4	468671	5
5	Midtown Center	Manhattan	2016	3	467864	6
6	Upper East Side North	Manhattan	2015	5	464786	7
7	Midtown Center	Manhattan	2015	7	464059	8
8	Midtown Center	Manhattan	2015	2	462245	9
9	Midtown Center	Manhattan	2015	10	458148	10

Donde Midtown Center y Upper East Side North son las zonas con más pickups por volumen mensual.

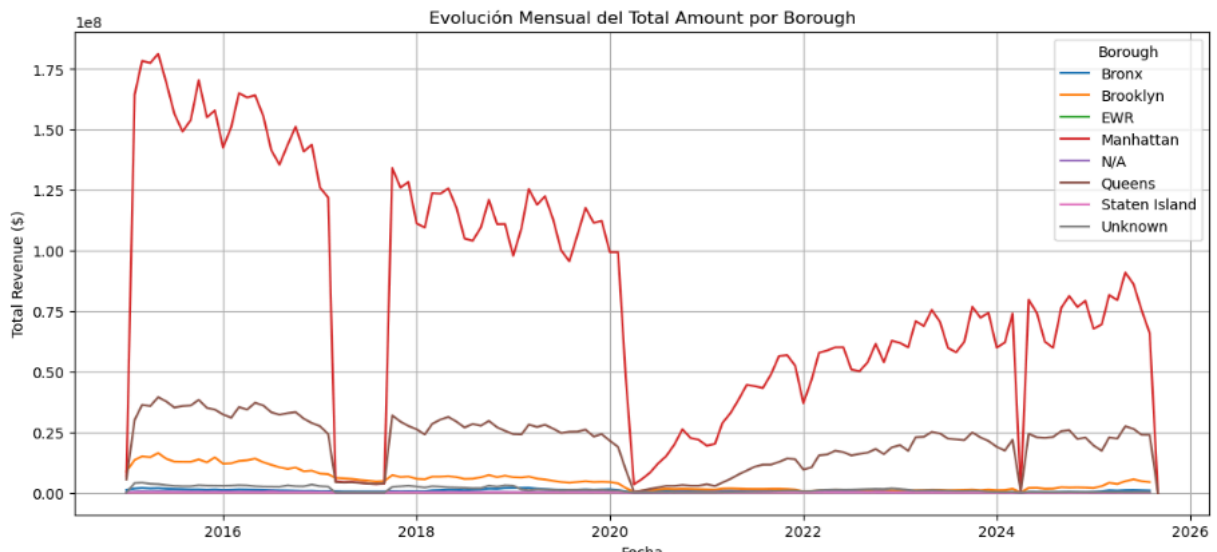
A continuación se muestra el resultado del segundo caso:

	DO_ZONE	DO_BOROUGH	YEAR	MONTH	TOTAL_TRIPS	GLOBAL_RANK
0	Midtown Center	Manhattan	2015	3	506987	1
1	Upper East Side North	Manhattan	2015	4	468671	2
2	Upper East Side South	Manhattan	2015	4	440255	3
3	Murray Hill	Manhattan	2015	3	431839	4
4	Times Sq/Theatre District	Manhattan	2015	3	423468	5
5	Midtown East	Manhattan	2015	3	406537	6
6	Union Sq	Manhattan	2015	3	383288	7
7	Penn Station/Madison Sq West	Manhattan	2015	3	364131	8
8	Clinton East	Manhattan	2015	3	358657	9
9	East Village	Manhattan	2015	3	358584	10

Se ve que las zonas con mayor pickup por volumen mensual se dieron en los meses de 2015, afirmando que en ese año se hacían muchos más viajes comparado con el resto de años.

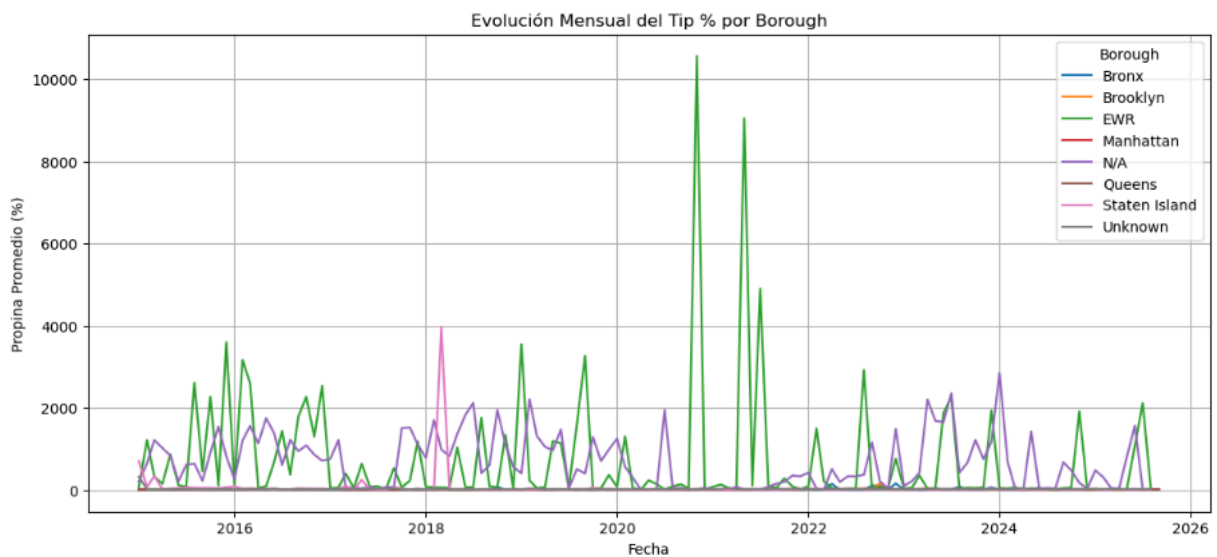
c. Evolución mensual de total_amount y tip_pct por borough.

Evolución mensual de total_amount



El gráfico muestra el revenue total (eje y) por meses (eje x) dividido en las distintas zonas (lo cual se muestra por colores). En general vemos una tendencia a disminuir el revenue con los meses hasta 2020, pero vemos que el mayor revenue se obtuvo en los meses de 2015-2017, pero en los primeros meses de 2017 se desplomó y a finales de 2017 se vio un gran incremento. La tendencia de decremento se mantiene entre los meses de 2018 hasta finales de 2019 cuando se desploma totalmente el revenue total probablemente debido a la pandemia. Después de esto se ve un incremento en el revenue hasta mediados 2025.

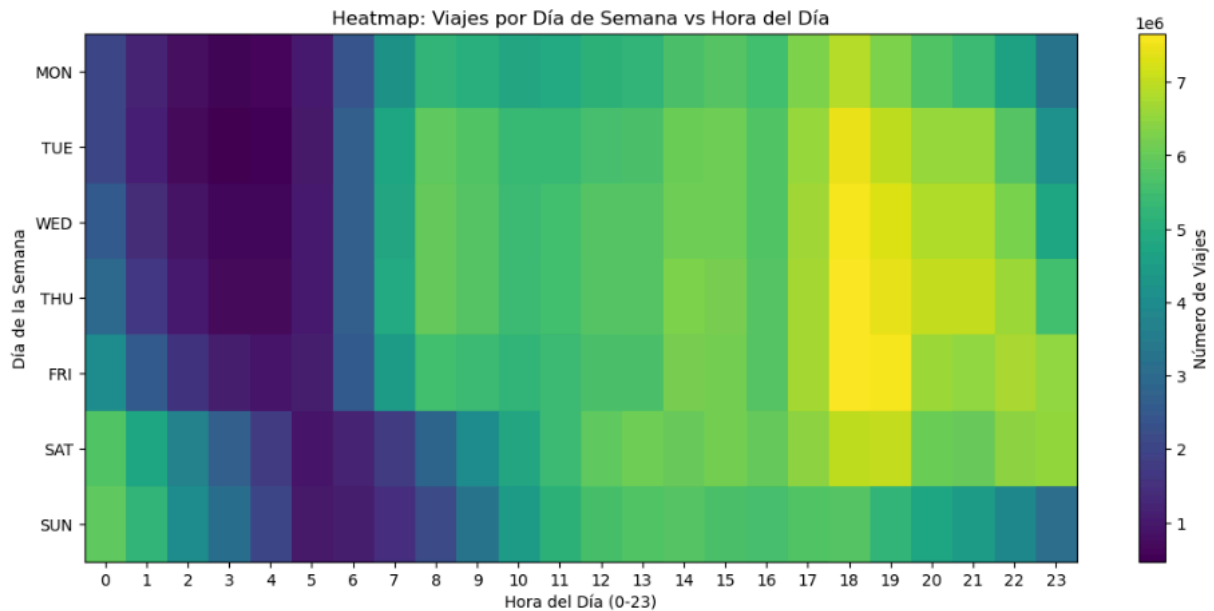
Evolución mensual de tip_pct



d. Ticket promedio (avg total_amount) por service_type y mes.

	YEAR	MONTH	SERVICE_TYPE	AVG_TICKET
0	2015	1	GREEN	14.780456
1	2015	2	GREEN	14.490028
2	2015	2	YELLOW	15.400438
3	2015	3	GREEN	14.576802
4	2015	3	YELLOW	15.849297
5	2015	4	GREEN	14.814617
6	2015	4	YELLOW	16.035905
7	2015	5	GREEN	15.250106
8	2015	5	YELLOW	16.426353
9	2015	6	GREEN	14.958572
10	2015	6	YELLOW	16.378267
11	2015	7	GREEN	14.908148
12	2015	7	YELLOW	16.143468
13	2015	8	GREEN	14.931218
14	2015	8	YELLOW	16.153374
15	2015	9	GREEN	15.032139

e. Viajes por hora del día y día de semana (picos).



Este gráfico muestra el número de viajes (la leyenda en la parte derecha muestra 7 colores distintos dependiendo de la cantidad) por hora del día (eje x) y por día de la semana (eje y). Mientras más amarillo está un cuadrado, significa que se contabilizaron más viajes realizados en esa hora y ese día específicos. De lunes a viernes vemos que la hora pico es las 18 horas, es decir, las 6 de la tarde. Para sábado vemos que las horas pico son las 18 y las 19 horas.

Si es que vemos por hora, tenemos lo siguiente:

	HOUR_OF_DAY	TOTAL_TRIPS
0	18	50077837
1	19	47864856
2	17	45190232
3	20	43519870
4	21	42857785
5	15	42259799
6	14	42186711
7	22	40417831
8	16	40108054
9	13	39915438

Esto nos muestra que las horas pico son las 17, 18, 19, y 20 horas.

f. p50/p90 de trip_duration_min por borough de pickup.

	BOROUGH	P50_DURATION	P90_DURATION
0	EWB	0.000000	2.0
1	N/A	1.000000	60.0
2	Unknown	10.975578	28.0
3	Manhattan	11.000000	25.0
4	Brooklyn	13.000000	33.0
5	Bronx	13.244216	38.0
6	Staten Island	23.993762	71.0
7	Queens	24.000000	54.0

En Manhattan hay mucha movilidad pero vemos que son viajes cortos porque incluso en tráfico o viajes largos no superan los 25 minutos (P90). En Brooklyn hay más variación que en Manhattan y Bronx es similar a Brooklyn con lo de

distancias urbanas medias. Queens y Staten Island tienen los viajes más largos, probablemente porque ahí hay aeropuertos o distancias más largas, igual en Staten Island hay conectividad más limitada por lo que puede preferirse hacer viajes inter Borough. EWR, N/A y Unknown son datos poco fiables porque probablemente son de viajes en zonas que no se registraron.

g. avg_speed_mph por franja horaria (6–9, 17–20) y borough.

	BOROUGH	TIME_WINDOW	AVG_SPEED_MPH
0	Bronx	Evening Peak (17-20)	13.046795
1	Bronx	Morning Peak (6-9)	14.429150
2	Brooklyn	Evening Peak (17-20)	11.350631
3	Brooklyn	Morning Peak (6-9)	13.258930
4	EWR	Evening Peak (17-20)	22.036847
5	EWR	Morning Peak (6-9)	23.714381
6	Manhattan	Evening Peak (17-20)	10.013497
7	Manhattan	Morning Peak (6-9)	11.445526
8	N/A	Evening Peak (17-20)	14.381043
9	N/A	Morning Peak (6-9)	14.281479
10	Queens	Evening Peak (17-20)	18.708507
11	Queens	Morning Peak (6-9)	18.423350
12	Staten Island	Evening Peak (17-20)	21.803862
13	Staten Island	Morning Peak (6-9)	20.976314
14	Unknown	Evening Peak (17-20)	10.938506
15	Unknown	Morning Peak (6-9)	11.991127

h. Participación por payment_type_desc y su relación con tip_pct.

	PAYMENT_TYPE_DESC	TOTAL_TRIPS	PARTICIPATION_PCT	AVG_TIP_PCT
0	Credit card	515683083	66.96	25.37
1	Cash	228912187	29.72	0.00
2	Unknown	18411624	2.39	5.90
3	No charge	3742930	0.49	0.07
4	Dispute	3366300	0.44	0.08

La mayoría de viajes se pagan con tarjeta de crédito (según lo que se definió en la normalización de los tipos de pagos). 2 de cada 3 viajes se pagan con tarjeta de crédito y la propina que se deja es alta, pues es del 25%. Después está el efectivo y que el porcentaje promedio de tips sea 0 no significa que no hayan dejado propinas, si no que puede ser que no se registra.

i. ¿Qué rate_code_desc concentran mayor trip_distance y total_amount?

	RATE_CODE_DESC	TOTAL_TRIPS	AVG_TRIP_DISTANCE	AVG_TOTAL_AMOUNT
0	Unknown	19611263	40.11	27.36
1	Nassau/Westchester	660642	27.69	96.89
2	JFK	17787657	22.28	70.57
3	Newark	1573373	16.29	91.50
4	Negotiated fare	4944564	9.94	54.50
5	Standard rate	725532058	4.36	16.65
6	Group ride	6567	1.23	23.69

(Esta respuesta es en base a la descripción que se hizo para el rate_code)

Vemos que standard rate concentra el mayor volumen (más del 95%). Vemos aquí distancias cortas (4.36 millas) y también un total amount promedio bastante bajo (el más bajo de hecho).

Vemos que los rate_code de JFK y Newark generan un alto revenue.

Nassau/Westchester son tienen el total_amount más alto (con la segunda

avg_trip_distance más alta).

j. Mix yellow vs green por mes y borough.

Este es un ejemplo de mayo de 2025:

	YEAR	MONTH	BOROUGH	YELLOW_TRIPS	GREEN_TRIPS	PCT_YELLOW	PCT_GREEN
992	2025	5	Bronx	38607	1107	97.21	2.79
993	2025	5	Brooklyn	180940	8130	95.70	4.30
994	2025	5	EWB	532	2	99.63	0.37
995	2025	5	Manhattan	3907565	33495	99.15	0.85
996	2025	5	N/A	2091	38	98.22	1.78
997	2025	5	Queens	454009	12498	97.32	2.68
998	2025	5	Staten Island	504	8	98.44	1.56
999	2025	5	Unknown	7596	127	98.36	1.64

Aquí vemos el borough y cuantos viajes hubo por cada tipo (amarillos o verdes), así como el porcentaje del total de viajes.

k. Top 20 flujos PU→DO por volumen y su ticket promedio.

	ROUTE	TOTAL_TRIPS	AVG_TICKET
0	N/A → N/A	6707216	17.89
1	Upper East Side South → Upper East Side North	4051848	10.47
2	Upper East Side North → Upper East Side South	3467785	11.38
3	Upper East Side North → Upper East Side North	3203336	8.67
4	Upper East Side South → Upper East Side South	3073863	9.22
5	Upper West Side South → Upper West Side North	1794600	9.03
6	Upper West Side South → Lincoln Square East	1779951	9.57
7	Upper East Side South → Midtown Center	1731209	12.28
8	Upper East Side South → Midtown East	1716327	10.86
9	Lincoln Square East → Upper West Side South	1703205	10.03

	ROUTE	TOTAL_TRIPS	AVG_TICKET
10	Midtown Center → Upper East Side South	1655148	11.92
11	Upper West Side North → Upper West Side South	1549673	8.94
12	Lenox Hill West → Upper East Side North	1533148	10.26
13	Penn Station/Madison Sq West → Times Sq/Theatr...	1483077	12.63
14	Penn Station/Madison Sq West → Midtown Center	1474285	13.26
15	Upper East Side South → Lenox Hill West	1375482	9.40
16	Times Sq/Theatre District → Penn Station/Madis...	1357704	10.96
17	Clinton East → Clinton East	1351713	10.19
18	Gramercy → Murray Hill	1343206	10.45
19	Lenox Hill West → Upper East Side South	1324603	9.99

La ruta más solicitada es desde Upper East Side North hasta Upper East Side South, que es también la ruta con el avg_ticket más elevado.

I. Distribución de passenger_count y efecto en total_amount.

	PASSENGER_COUNT	TOTAL_TRIPS	AVG_TOTAL_AMOUNT
0	0.0	5794119	19.72
1	1.0	549203643	18.23
2	2.0	105632100	19.67
3	3.0	29160923	19.16
4	4.0	14125689	20.07
5	5.0	29640870	17.13
6	6.0	18144427	16.96
7	7.0	3629	46.37
8	8.0	3728	48.09
9	9.0	1872	60.51
10	32.0	1	60.35
11	48.0	1	40.30
12	96.0	2	12.59
13	112.0	1	14.76
14	192.0	2	9.15

Vemos que dominan los viajes para una persona. En viajes grupales (2-4) aumenta ligeramente el total_amount en relación a los de una persona.

Vemos que si hay bastantes errores o inconsistencias. No puede haber viajes con cero pasajeros (y son casi 6 millones). Podrían deberse a cancelaciones o errores como tal. Los viajes que tienen de 7-9 pasajes deben ser taxis más grandes o grupales, que son los que tienen las tarifas más altas. De ahí los siguientes valores (desde el décimo dato de la tabla en adelante) deben marcarse como registros "corruptos" porque es imposible llevar a más de 32 personas en un taxi.

m. Impacto de tolls_amount y congestion_surcharge por zona.

	PU_ZONE	PU_BOROUGH	AVG_TOLLS	AVG_CONGESTION	AVG_TOTAL_AMOUNT	TOTAL_TRIPS
0	Rikers Island	Bronx	29.51	1.19	57.42	289
1	Arden Heights	Staten Island	13.87	0.11	85.06	2203
2	Arrochar/Fort Wadsworth	Staten Island	9.37	0.08	31.97	4664
3	Bloomfield/Emerson Hill	Staten Island	7.31	0.04	68.36	7464
4	Charleston/Tottenville	Staten Island	6.96	0.01	88.84	3895
5	Eltingville/Annadale/Prince's Bay	Staten Island	5.64	0.08	69.99	528
6	South Beach/Dongan Hills	Staten Island	5.38	0.08	48.04	2038
7	Rossville/Woodrow	Staten Island	5.26	0.07	84.12	367
8	Mariners Harbor	Staten Island	5.07	0.07	50.60	4784
9	Grymes Hill/Clifton	Staten Island	4.92	0.07	44.82	5452
10	Oakwood	Staten Island	4.71	0.22	57.63	444
11	New Dorp/Midland Beach	Staten Island	4.53	0.17	45.90	1004
12	Westerleigh	Staten Island	4.44	0.19	48.65	1516
13	Great Kills	Staten Island	4.38	0.13	58.14	628
14	Freshkills Park	Staten Island	4.17	0.12	66.52	308

	PU_ZONE	PU_BOROUGH	AVG_TOLLS	AVG_CONGESTION	AVG_TOTAL_AMOUNT	TOTAL_TRIPS
15	Randalls Island	Manhattan	4.16	1.07	58.31	50943
16	LaGuardia Airport	Queens	3.85	1.51	47.16	18298828
17	Heartland Village/Todt Hill	Staten Island	3.85	0.24	53.54	2499
18	Port Richmond	Staten Island	3.79	0.20	41.70	1331
19	East Elmhurst	Queens	3.52	1.45	48.64	826521
20	Baisley Park	Queens	3.35	1.26	56.82	230217
21	Stapleton	Staten Island	3.35	0.24	37.08	4901
22	Newark Airport	EWB	3.15	0.02	92.72	68159
23	JFK Airport	Queens	2.91	1.12	61.21	20975981
24	Rockaway Park	Queens	2.84	0.05	48.50	10650
25	Breezy Point/Fort Tilden/Riis Beach	Queens	2.80	0.11	45.38	1838
26	Saint George/New Brighton	Staten Island	2.73	0.12	38.18	6290
27	Hammels/Arverne	Queens	2.69	0.02	55.05	39870
28	Springfield Gardens South	Queens	2.64	0.68	56.02	79570
29	South Jamaica	Queens	2.46	0.90	46.74	132644

Los peajes más altos no se encuentran en Manhattan, sino en Staten Island, Bronx y puntos aeroportuarios, donde los viajes deben cruzar puentes de alto peaje.

Aunque Manhattan es históricamente la más congestionada, los aeropuertos de Queens (LGA y JFK) muestran altos recargos de congestión. Los peajes elevados se concentran en viajes suburbanos (Staten Island, Bronx, aeropuertos), mientras que los recargos por congestión se localizan en los accesos a Queens y Manhattan.

n. Proporción de viajes cortos vs largos por borough y estacionalidad

Este es un ejemplo de Bronx de los primeros dos meses de 2015:

	PU_BOROUGH	YEAR	MONTH	TRIP_CATEGORY	TOTAL_TRIPS	PCT_CATEGORY
0	Bronx	2015	1	Long	3129	3.41
1	Bronx	2015	1	Medium	39394	42.95
2	Bronx	2015	1	Short	49189	53.63
3	Bronx	2015	2	Long	5873	4.38
4	Bronx	2015	2	Medium	63988	47.74
5	Bronx	2015	2	Short	64162	47.87

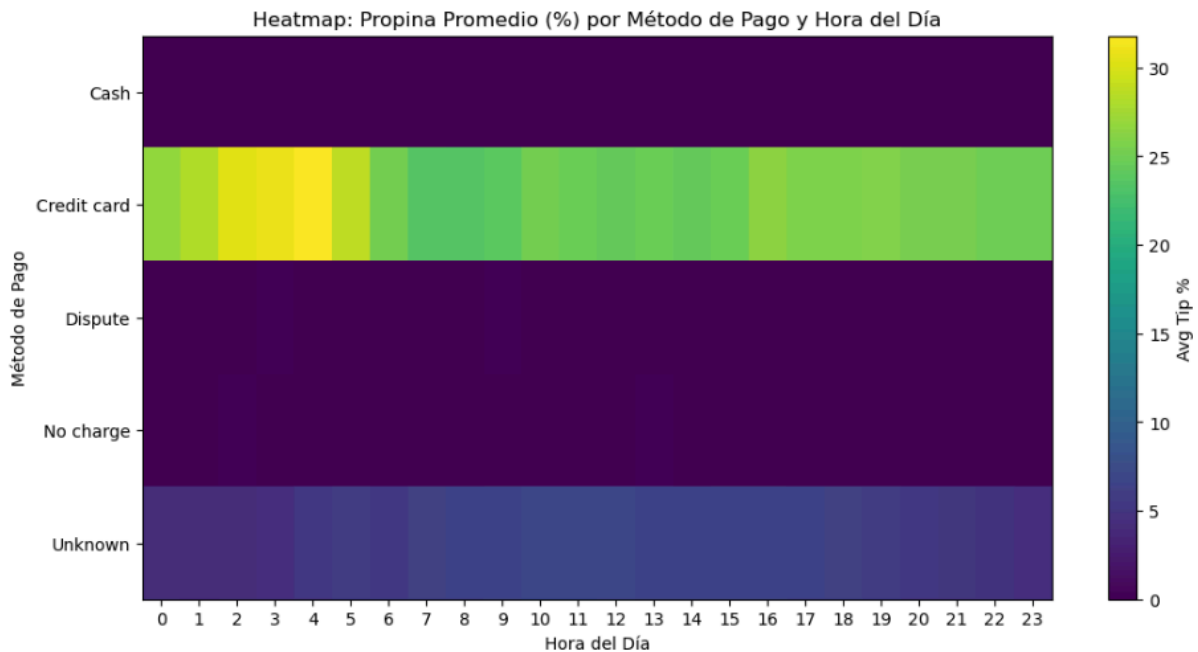
El criterio que se utilizó fue que menor a 10 minutos es short, entre 10 y 30 minutos es medium, y cualquier otra cosa es largo.

o. Diferencias por vendor en avg_speed_mph y trip_duration_min

	VENDOR_NAME	AVG_TRIP_DURATION	AVG_SPEED
0	VIP	25.41	18.27
1	VeriFone Transportation Systems (VTS)	18.67	17.59
2	Creative Mobile Technologies (CMT)	16.21	34.28
3	Unknown	15.56	10.46
4	DDS	14.16	10.62

VIP es el vendedor con mayor duración promedio de viajes y los que tienen la segunda velocidad promedio más alta. CMT tiene la mayor velocidad promedio.

p. Relación método de pago ↔ tip_amount por hora.

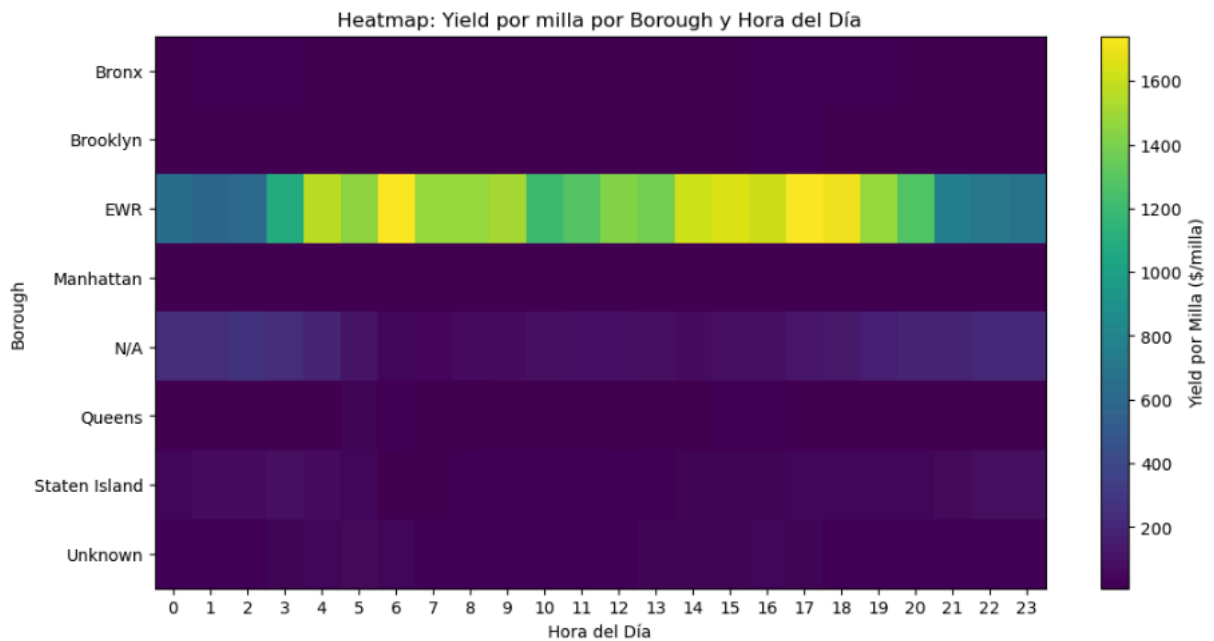


Los pagos con tarjeta de crédito son los que mayor porcentaje de propina dejan. Vemos que estos son incluso mayores en las horas de la madrugada, sobre todo a las 4 de la mañana.

q. Zonas con percentil 99 de duración/distancia fuera de rango (posible congestión/eventos).

	PU_ZONE	PU_BOROUGH	P99_DURATION_MIN	P99_DISTANCE	TOTAL_TRIPS
0	Bronx Park	Bronx	215.000	21.9828	33287
1	Arden Heights	Staten Island	184.980	40.1854	2203
2	Coney Island	Brooklyn	172.000	29.8000	191582
3	Rossville/Woodrow	Staten Island	170.039	45.9738	367
4	Mariners Harbor	Staten Island	160.340	36.4217	4784
...
189	Steinway	Queens	61.000	14.9000	1513338
190	Elmhurst/Maspeth	Queens	61.000	13.9700	475867
191	Long Island City/Hunters Point	Queens	61.000	15.6400	1798428
192	Chinatown	Manhattan	61.000	16.7500	1432092
193	Prospect Heights	Brooklyn	61.000	14.0200	558157

r. Yield por milla (total_amount/trip_distance) por borough y hora.



Borough	Comportamiento económico
EWR (Aeropuerto Newark)	Área más rentable por viaje corto y tarifa fija
Manhattan / Queens	Rentabilidad moderada, ingresos por volumen no milla

Borough	Comportamiento económico
Staten Island / Bronx	Viajes largos, bajo margen por milla

s. Cambios YoY en volumen y ticket promedio por service_type.

	YEAR	SERVICE_TYPE	TOTAL_TRIPS	AVG_TICKET	YOY_TRIPS_PCT	YOY_TICKET_PCT
0	2015	GREEN	19233765	14.84	NaN	NaN
1	2016	GREEN	16385541	14.64	-14.81	-1.35
2	2017	GREEN	11736906	14.24	-28.37	-2.73
3	2018	GREEN	8899314	16.09	-24.18	12.99
4	2019	GREEN	6300814	18.33	-29.20	13.92
5	2020	GREEN	1734166	20.16	-72.48	9.98
6	2021	GREEN	1068729	23.93	-38.37	18.70
7	2022	GREEN	840394	19.32	-21.37	-19.26
8	2023	GREEN	787055	23.86	-6.35	23.50
9	2024	GREEN	660204	24.26	-16.12	1.68

	YEAR	SERVICE_TYPE	TOTAL_TRIPS	AVG_TICKET	YOY_TRIPS_PCT	YOY_TICKET_PCT
10	2025	GREEN	397923	24.85	-39.73	2.43
11	2015	YELLOW	133298197	16.19	NaN	NaN
12	2016	YELLOW	131131805	16.38	-1.63	1.17
13	2017	YELLOW	47442646	16.12	-63.82	-1.59
14	2018	YELLOW	102870524	16.43	116.83	1.92
15	2019	YELLOW	84597309	19.19	-17.76	16.80
16	2020	YELLOW	24649266	18.42	-70.86	-4.01
17	2021	YELLOW	30903983	19.70	25.37	6.95
18	2022	YELLOW	39655622	21.67	28.32	10.00
19	2023	YELLOW	38310138	28.46	-3.39	31.33
20	2024	YELLOW	37655406	27.86	-1.71	-2.11
21	2025	YELLOW	31556417	26.42	-16.20	-5.17

El taxi verde ha perdido su base de usuarios desde 2015 y no ha logrado recuperarse ni siquiera tras COVID, sobreviviendo solo con tarifas más elevadas y nichos urbanos. En contraste, el taxi amarillo, aunque afectado por la pandemia, ha demostrado resiliencia ya que recupera parte de su volumen y aumenta significativamente el ticket promedio.

t. Días con alta congestion_surcharge: efecto en total_amount vs días "normales"

	DAY_TYPE	OVERALL_AVG_TICKET
0	High Congestion Day	22.93
1	Normal Day	16.17

"High Congestion Day" es un día donde los taxis acumulan más de \$10,000 en recargos por congestión, lo cual indica condiciones de tráfico anormales o eventos urbanos mayores.

Un día altamente congestionado en NYC incrementa el ticket promedio del taxi en torno a un 40%. Esto confirma que el tráfico urbano no solo ralentiza la movilidad, sino que introduce un coste adicional significativo para las personas, convirtiendo al taxi en un servicio sensiblemente más caro durante picos viales.

Checklist

- ☒ Docker Compose levanta **Spark y Jupyter Notebook**.
- ☒ Todas las credenciales/parámetros provienen de **variables de ambiente** (`.env`).
- ☒ **Cobertura 2015–2025** (Yellow/Green) cargada en **raw** con **matriz** y **conteos** por lote.
- ☒ **analytics.obt_trips** creada con columnas mínimas, derivadas y metadatos.
- ☒ **Idempotencia** verificada reingestando al menos un mes.
- ☒ **Validaciones** básicas documentadas (nulos, rangos, coherencia).
- ☒ **20 preguntas** respondidas (texto) usando la OBT.
- ☒ README claro: pasos, variables, esquema, decisiones, troubleshooting.