Ejercicio 2:

Alquiler de automóviles

Una de las empresas líderes en alquileres de automóviles solicita una serie de dashboards y reportes para poder basar sus decisiones en datos. Entre los indicadores mencionados se encuentran total de alquileres, segmentación por tipo de combustible, lugar, marca y modelo de automóvil, valoración de cada alquiler, etc.
Como Data Engineer debe crear y automatizar el pipeline para tener como resultado los datos listos para ser visualizados y responder las preguntas de negocio.

1. Crear en hive una database car_rental_db y dentro una tabla llamada car_rental_analytics, con estos campos:

campos	tipo
fuelType	string
rating	integer
renterTripsTaken	integer
reviewCount	integer
city	string
state_name	string
owner_id	integer
rate_daily	integer
make	string
model	string
year	integer

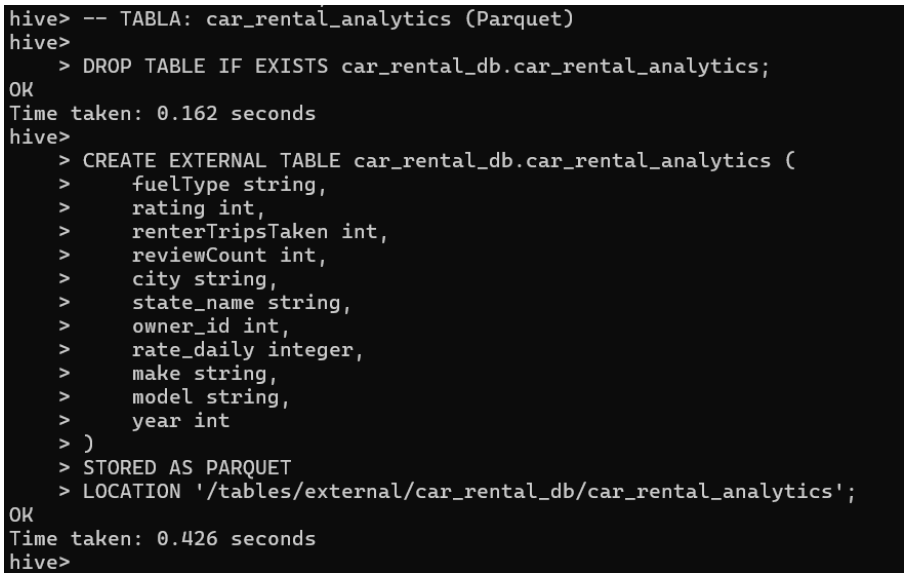
En primer lugar, creamos la base de datos **car_rental_db** en **Hive**:

```
hive> create database car_rental_db;
OK
Time taken: 3.068 seconds
hive> show databases;
OK
aeropuertos
car_rental_db
default
f1
northwind_analytics
titanic
tripdata
trips
Time taken: 0.105 seconds, Fetched: 8 row(s)
hive>
```

Posteriormente, creamos la tabla **car_rental_analytics** (en formato parquet) en la base de datos:

```
-- TABLA: car_rental_analytics (Parquet)

DROP TABLE IF EXISTS car_rental_db.car_rental_analytics;

CREATE EXTERNAL TABLE car_rental_db.car_rental_analytics (
  fuelType string,
  rating int,
  renterTripsTaken int,
  reviewCount int,
  city string,
  state_name string,
  owner_id int,
  rate_daily integer,
  make string,
  model string,
  year int
)
STORED AS PARQUET
LOCATION '/tables/external/car_rental_db/car_rental_analytics';
```

Vista desde el **bash**:



```
hive> -- TABLA: car_rental_analytics (Parquet)
hive>
    > DROP TABLE IF EXISTS car_rental_db.car_rental_analytics;
OK
Time taken: 0.162 seconds
hive>
    > CREATE EXTERNAL TABLE car_rental_db.car_rental_analytics (
    >     fuelType string,
    >     rating int,
    >     renterTripsTaken int,
    >     reviewCount int,
    >     city string,
    >     state_name string,
    >     owner_id int,
    >     rate_daily integer,
    >     make string,
    >     model string,
    >     year int
    > )
    > STORED AS PARQUET
    > LOCATION '/tables/external/car_rental_db/car_rental_analytics';
OK
Time taken: 0.426 seconds
hive>
```

Visualizamos que las tablas se crearon correctamente:

```
hive> describe formatted car_rental_db.car_rental_analytics;
OK
# col_name              data_type               comment

fueltype                string
rating                  int
rentertripstaken        int
reviewcount             int
city                    string
state_name              string
owner_id                int
rate_daily              int
make                    string
model                   string
year                    int

# Detailed Table Information
Database:               car_rental_db
Owner:                  hadoop
CreateTime:             Mon Nov 17 01:14:01 ART 2025
LastAccessTime:         UNKNOWN
Retention:              0
Location:               hdfs://172.17.0.2:9000/tables/external/car_rental_db/car_rental_analytics
Table Type:             EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                TRUE
        transient_lastDdlTime   1763352841

# Storage Information
SerDe Library:          org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
InputFormat:            org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        serialization.format    1
Time taken: 0.169 seconds, Fetched: 36 row(s)
hive>
```

2. Crear script para el ingest de estos dos files

https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv

https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv

Sugerencia: descargar el segundo archivo con un comando similar al abajo mencionado, ya que al tener caracteres como '&' falla si no se le asignan comillas. Adicionalmente, el parámetro -O permite asignarle un nombre más legible al archivo descargado

wget -P ruta_destino -O ruta_destino/nombre_archivo.csv ruta_al_archivo

Info del dataset: https://www.kaggle.com/datasets/kushleshkumar/cornell-car-rental-dataset

Realizamos el **ingest** con el script **ingest_ejercicio_2.sh**, alojado en **/home/hadoop/scripts**. Este script permite descargar los archivos al directorio **/home/hadoop/landing** y posteriormente ingestarlos en **hdfs** en el directorio **/ingest**.

```
################################################################################################
# Archivo: ingest_ejercicio_2.sh  Ubicacion: /home/hadoop/scripts
#
# Descarga e Ingest en HDFS de los archivos:
# https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv
# https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv
#
################################################################################################
```

Punto 2

Descarga datos desde el repositorio al directorio /home/hadoop/landing

wget -P /home/hadoop/landing https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv
wget -O /home/hadoop/landing/georef_usa.csv https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv

Lleva el archivo a HDFS al directorio /ingest
hdfs dfs -put /home/hadoop/landing/CarRentalData.csv /ingest
hdfs dfs -put /home/hadoop/landing/georef_usa.csv /ingest

Borra el archivo starwars.csv del directorio /home/hadoop/landing/
rm /home/hadoop/landing/CarRentalData.csv
rm /home/hadoop/landing/georef_usa.csv

Creación del archivo en **bash**:

```
hadoop@615cf53bef6c:~/scripts$ cat > ingest_ejercicio_2.sh
#################################################################################################
# Archivo: ingest_ejercicio_2.sh  Ubicacion: /home/hadoop/scripts
#
# Descarga e Ingest en HDFS de los archivos:
# https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv
# https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv
#
#################################################################################################

# Punto 2

# Descarga datos desde el repositorio al directorio /home/hadoop/landing
wget -P /home/hadoop/landing https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv
wget -O /home/hadoop/landing/georef_usa.csv https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv


# Lleva el archivo a HDFS al directorio /ingest
hdfs dfs -put /home/hadoop/landing/CarRentalData.csv /ingest
hdfs dfs -put /home/hadoop/landing/georef_usa.csv /ingest


# Borra el archivo starwars.csv del directorio /home/hadoop/landing/
rm /home/hadoop/landing/CarRentalData.csv
rm /home/hadoop/landing/georef_usa.csv
hadoop@615cf53bef6c:~/scripts$
```

A continuación, le asignamos permisos al citado archivo:

```
hadoop@615cf53bef6c:~/scripts$ ls
Notebook               ingest_ejercicio_2.sh  query_db_2_hdfs_parquet.sh    tl6_run.log          transform_load_4.py  weather_str_transform.py
QueryResult.java       landing.sh             query_db_2_hdfs_parquet_2.sh  tl7_run.log          transform_load_5.py  yahoo_weather2kafka.py
derby.log              landing_2.sh           query_db_2_hdfs_parquet_3.sh  transform_load.py    transform_load_6.py
ingest.sh              landing_3.sh           spark-warehouse               transform_load_2.py  transform_load_7.py
ingest_ejercicio_1.sh  pyspark_jupyter.sh     start-services.sh             transform_load_3.py  transformation.py
hadoop@615cf53bef6c:~/scripts$ chmod 777 ingest_ejercicio_2.sh
hadoop@615cf53bef6c:~/scripts$ ls
Notebook               ingest_ejercicio_2.sh  query_db_2_hdfs_parquet.sh    tl6_run.log          transform_load_4.py  weather_str_transform.py
QueryResult.java       landing.sh             query_db_2_hdfs_parquet_2.sh  tl7_run.log          transform_load_5.py  yahoo_weather2kafka.py
derby.log              landing_2.sh           query_db_2_hdfs_parquet_3.sh  transform_load.py    transform_load_6.py
ingest.sh              landing_3.sh           spark-warehouse               transform_load_2.py  transform_load_7.py
ingest_ejercicio_1.sh  pyspark_jupyter.sh     start-services.sh             transform_load_3.py  transformation.py
hadoop@615cf53bef6c:~/scripts$
```

Probamos que el archivo funcione correctamente:

```
hadoop@615cf53bef6c:~/scripts$ ./ingest_ejercicio_2.sh
--2025-11-17 01:44:39--  https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv
Resolving data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)... 3.5.30.161, 52.216.59.153, 54.231.226.129, ...
Connecting to data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)|3.5.30.161|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 533157 (521K) [text/csv]
Saving to: '/home/hadoop/landing/CarRentalData.csv'

CarRentalData.csv                   100%[===================================================================================>] 520.66K   123KB/s    in 4.2s

2025-11-17 01:44:49 (123 KB/s) - '/home/hadoop/landing/CarRentalData.csv' saved [533157/533157]

--2025-11-17 01:44:49--  https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv
Resolving data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)... 52.216.51.161, 52.217.205.185, 16.15.179.44, ...
Connecting to data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)|52.216.51.161|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3380726 (3.2M) [text/csv]
Saving to: '/home/hadoop/landing/georef_usa.csv'

/home/hadoop/landing/georef_usa.csv    100%[===================================================================================>]   3.22M  1.04MB/s    in 3.1s

2025-11-17 01:44:55 (1.04 MB/s) - '/home/hadoop/landing/georef_usa.csv' saved [3380726/3380726]

hadoop@615cf53bef6c:~/scripts$
```

Revisamos si en efecto están correctamente guardados los archivos en hdfs en **/ingest**:

```
hadoop@615cf53bef6c:/$ hdfs dfs -ls /ingest
Found 9 items
-rw-r--r--   1 hadoop supergroup   32322556 2025-11-15 19:47 /ingest/2021-informe-ministerio.csv
-rw-r--r--   1 hadoop supergroup   22833520 2025-11-15 19:47 /ingest/202206-informe-ministerio.csv
-rw-r--r--   1 hadoop supergroup     533157 2025-11-17 01:44 /ingest/CarRentalData.csv
-rw-r--r--   1 hadoop supergroup     136007 2025-11-15 19:47 /ingest/aeropuertos_detalle.csv
-rw-r--r--   1 hadoop supergroup      17478 2025-10-25 04:05 /ingest/constructors.csv
-rw-r--r--   1 hadoop supergroup      94367 2025-10-25 04:05 /ingest/drivers.csv
-rw-r--r--   1 hadoop supergroup    3380726 2025-11-17 01:45 /ingest/georef_usa.csv
-rw-r--r--   1 hadoop supergroup     164344 2025-10-25 04:05 /ingest/races.csv
-rw-r--r--   1 hadoop supergroup    1721961 2025-10-25 04:05 /ingest/results.csv
hadoop@615cf53bef6c:/$
```

Se observa que los archivos fueron ingestados correctamente en **hdfs**

3. Crear un script para tomar el archivo desde HDFS y hacer las siguientes transformaciones:

- En donde sea necesario, modificar los nombres de las columnas. Evitar espacios y puntos (reemplazar por _). Evitar nombres de columna largos
- Redondear los float de 'rating' y castear a int.
- Joinear ambos files
- Eliminar los registros con rating nulo
- Cambiar mayúsculas por minúsculas en 'fuelType'
- Excluir el estado Texas

Finalmente insertar en Hive el resultado

El script que realiza la tarea es **transform_load_8.py** y esta alojado en **/home/hadoop/scripts**:

transform_load_8.py 2

C: > Users > Carlos Rubén Bageta > Desktop > Cursos > Data Science > EDVAI > Bootcamp Data Engineering > Examen Final > Ejercicio_2 > transform_load_8.py > main

```python
##########################################################################################################
# Archivo: transform_load_8.py
##########################################################################################################

# Importamos librerias
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

def main():
    # --- Parámetros de HDFS ---
    NN_URI = "hdfs://172.17.0.2:9000"  # ajusta si tu NameNode/puerto son otros
    INPUT_1  = f"{NN_URI}/ingest/CarRentalData.csv"
    INPUT_2  = f"{NN_URI}/ingest/georef_usa.csv"
    OUTPUT_TABLA_1_DW = f"{NN_URI}/tables/external/car_rental_db/car_rental_analytics"


    # 1) Crear la sesión de Spark con soporte Hive y FS por defecto en HDFS
    spark = (
        SparkSession
            .builder
            .appName("transform_load_7")
            .enableHiveSupport()
            .config("spark.hadoop.fs.defaultFS", NN_URI)
            .getOrCreate()
    )

    # 2) Rutas de entrada (csv desde HDFS)
    df1 = spark.read.csv(INPUT_1, header=True, inferSchema=True, sep=",")
    df2 = spark.read.csv(INPUT_2, header=True, inferSchema=True, sep=";")

    # 3) Transformaciones (casteo de tipos)

    # Casteo de df1 y df2 (solo las columnas que nos hacen falta de este ultimo)

    df1_cast = df1.select(
        F.col("fuelType").cast("string").alias("fueltype"),
        F.round(F.col("rating"), 0).cast("int").alias("rating"), # Redondeamos el rating y casteamos a int
        F.col("renterTripsTaken").cast("int").alias("rentertripstaken"),
        F.col("reviewCount").cast("int").alias("reviewcount"),
        F.col("`location.city`").cast("string").alias("city"),
        F.col("`location.state`").cast("string").alias("state_id1"),
        F.col("`owner.id`").cast("int").alias("owner_id"),
        F.col("`rate.daily`").cast("int").alias("rate_daily"),
        F.col("`vehicle.make`").cast("string").alias("make"),
        F.col("`vehicle.model`").cast("string").alias("model"),
        F.col("`vehicle.year`").cast("int").alias("year")
    )
    df2_cast = df2.select(
        F.col("United States Postal Service state abbreviation").cast("string").alias("state_id2"), #sirve de columna para joinear con df1
        F.col("Official Name State").cast("string").alias("state_name")
        )

    # hacemos el join de ambos dataframes

    dfjoin = df1_cast.join(df2_cast, df1_cast.state_id1 == df2_cast.state_id2, "left")

    # Reordenamos las columnas en el dataframe, para que queden alineadas con la tabla de Hive

    df = dfjoin.select(
        dfjoin.fueltype,
        dfjoin.rating,
        dfjoin.rentertripstaken,
        dfjoin.reviewcount,
        dfjoin.city,
        dfjoin.state_name,
        dfjoin.owner_id,
        dfjoin.rate_daily,
        dfjoin.make,
        dfjoin.model,
        dfjoin.year
        )

    # Creamos una vista temporal df_v para hacer consultas SQL

    df.createOrReplaceTempView("df_v")

    # Filtramos Texas y eliminamos registros nulos en la columna 'rating'

    df_clean = spark.sql("""
        SELECT *
        FROM df_v
        WHERE state_name <> 'Texas' AND rating IS NOT NULL
    """)

    # 4) Load (Parquet a HDFS)
    df_clean.write.mode("overwrite").parquet(OUTPUT_TABLA_1_DW)

    spark.stop()

if __name__ == "__main__":
    main()
```

Vista de la creación del script en el directorio señalado en **bash**:

```
hadoop@615cf53bef6c:~/scripts$ cat > transform_load_8.py
################################################################################################################
# Archivo: transform_load_8.py
################################################################################################################

# Importamos librerias
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

def main():
    # === Parámetros de HDFS ===
    NN_URI = "hdfs://172.17.0.2:9000"  # ajusta si tu NameNode/puerto son otros
    INPUT_1   = f"{NN_URI}/ingest/CarRentalData.csv"
    INPUT_2   = f"{NN_URI}/ingest/georef_usa.csv"
    OUTPUT_TABLA_1_DW = f"{NN_URI}/tables/external/car_rental_db/car_rental_analytics"


    # 1) Crear la sesión de Spark con soporte Hive y FS por defecto en HDFS
    spark = (
        SparkSession
            .builder
            .appName("transform_load_7")
            .enableHiveSupport()
            .config("spark.hadoop.fs.defaultFS", NN_URI)
            .getOrCreate()
    )

    # 2) Rutas de entrada (csv desde HDFS)
    df1 = spark.read.csv(INPUT_1, header=True, inferSchema=True, sep=",")
    df2 = spark.read.csv(INPUT_2, header=True, inferSchema=True, sep=";")

    # 3) Transformaciones (casteo de tipos)

    # Casteo de df1 y df2 (solo las columnas que nos hacen falta de este ultimo)

    df1_cast = df1.select(
        F.col("fuelType").cast("string").alias("fueltype"),
        F.round(F.col("rating"), 0).cast("int").alias("rating"), # Redondeamos el rating y casteamos a int
        F.col("renterTripsTaken").cast("int").alias("rentertripstaken"),
        F.col("reviewCount").cast("int").alias("reviewcount"),
        F.col("`location.city`").cast("string").alias("city"),
        F.col("`location.state`").cast("string").alias("state_id1"),
        F.col("`owner.id`").cast("int").alias("owner_id"),
        F.col("`rate.daily`").cast("int").alias("rate_daily"),
        F.col("`vehicle.make`").cast("string").alias("make"),
        F.col("`vehicle.model`").cast("string").alias("model"),
        F.col("`vehicle.year`").cast("int").alias("year")
    )
    df2_cast = df2.select(
        F.col("United States Postal Service state abbreviation").cast("string").alias("state_id2"), #sirve de columna para joinear con df1
        F.col("Official Name State").cast("string").alias("state_name")
        )

    # hacemos el join de ambos dataframes

    dfjoin = df1_cast.join(df2_cast, df1_cast.state_id1 == df2_cast.state_id2, "left")

    # Reordenamos las columnas en el dataframe, para que queden alineadas con la tabla de Hive

    df = dfjoin.select(
        dfjoin.fueltype,
        dfjoin.rating,
        dfjoin.rentertripstaken,
        dfjoin.reviewcount,
        dfjoin.city,
        dfjoin.state_name,
        dfjoin.owner_id,
        dfjoin.rate_daily,
        dfjoin.make,
        dfjoin.model,
        dfjoin.year
        )

    # Creamos una vista temporal df_v para hacer consultas SQL

    df.createOrReplaceTempView("df_v")

    # Filtramos Texas y eliminamos registros nulos en la columna 'rating'

    df_clean = spark.sql("""
        SELECT *
        FROM df_v
        WHERE state_name <> 'Texas' AND rating IS NOT NULL
    """)

    # 4) Load (Parquet a HDFS)
    df_clean.write.mode("overwrite").parquet(OUTPUT_TABLA_1_DW)

    spark.stop()

if __name__ == "__main__":
    main()
hadoop@615cf53bef6c:~/scripts$
```

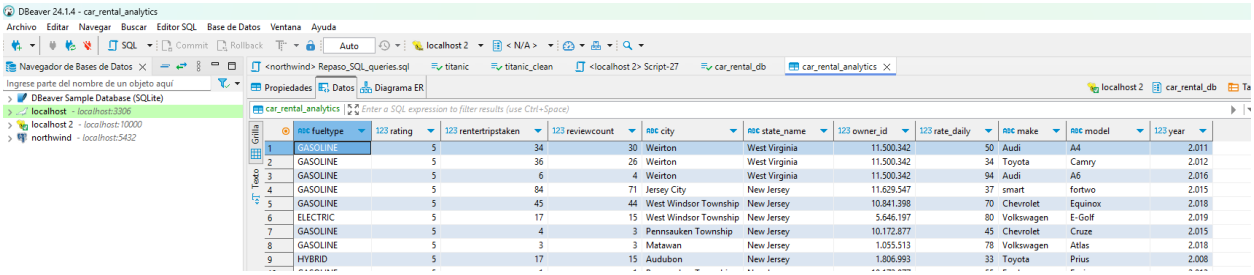
Verificamos que el script funciona correctamente con el siguiente **smoke run**:

```
hadoop@615cf53bef6c:~/scripts$ /home/hadoop/spark/bin/spark-submit --master local[*] \
>   --conf spark.sql.warehouse.dir=/user/hive/warehouse \
>   --conf spark.sql.catalogImplementation=hive \
>   --files /home/hadoop/hive/conf/hive-site.xml \
  /home/hadoop/>   /home/hadoop/scripts/transform_load_8.py 2>&1 | tee /home/hadoop/scripts/tl8_run.log
```

A continuación, verificamos que la tabla de Hive, **car_rental_analytics** (dentro de la base creada **car_rental_db** se cargó correctamente (mostramos los 10 primeros registros):

```
hive> select * from car_rental_db.car_rental_analytics limit 10;
OK
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
GASOLINE        5       34      30      Weirton West Virginia   11500342        50      Audi    A4      2011
GASOLINE        5       36      26      Weirton West Virginia   11500342        34      Toyota  Camry   2012
GASOLINE        5       6       4       Weirton West Virginia   11500342        94      Audi    A6      2016
GASOLINE        5       84      71      Jersey City     New Jersey      11629547        37      smart   fortwo  2015
GASOLINE        5       45      44      West Windsor Township   New Jersey      10841398        70      Chevrolet       Equinox 2018
ELECTRIC        5       17      15      West Windsor Township   New Jersey      5646197 80      Volkswagen      E-Golf  2019
GASOLINE        5       4       3       Pennsauken Township     New Jersey      10172877        45      Chevrolet       Cruze   2015
GASOLINE        5       3       3       Matawan New Jersey      1055513 78      Volkswagen      Atlas   2018
HYBRID  5       17      15      Audubon New Jersey      1806993 33      Toyota  Prius   2008
GASOLINE        5       1       1       Pennsauken Township     New Jersey      10172877        55      Ford    Fusion  2013
Time taken: 3.41 seconds, Fetched: 10 row(s)
hive>
```

Desde **DBeaver**, nos conectamos a Hive para ver la tabla y realizar las queries de los ejercicios posteriores:



Se puede observar que el insert en Hive fue correcto.

4. Realizar un proceso automático en Airflow que orqueste los pipelines creados en los puntos anteriores. Crear dos tareas:
   a. Un DAG padre que ingente los archivos y luego llame al DAG hijo
   b. Un DAG hijo que procese la información y la cargue en Hive

A continuación, se muestra el script del DAG_padre, alojado en :
/home/hadoop/airflow/dags/DAG_padre_ejercicio_2_trabajo_final.py

DAG_padre_ejercicio_2_trabajo_final.py > ...

```python
##################################################################
#
# DAG padre creado para el ejercicio 2 Trabajo Final
# /home/hadoop/airflow/dags/DAG_padre_ejercicio_2_trabajo_final.py
#
##################################################################

# Importamos librerias

from datetime import timedelta
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.dates import days_ago
from airflow.operators.trigger_dagrun import TriggerDagRunOperator


# Definimos los argumentos por defecto del DAG
args = {
    "owner": "airflow",
}

with DAG(
    dag_id="dag_padre",
    default_args=args,
    schedule_interval="0 0 * * *",
    start_date=days_ago(1),
    catchup=False,
    dagrun_timeout=timedelta(minutes=60),
    tags=["ingest"],
) as dag:

    inicio_proceso = DummyOperator(task_id="inicio_proceso")


    # Tareas Ingest

    ingest = BashOperator(
        task_id="ingest",
        bash_command="""
            set -e
            /bin/bash /home/hadoop/scripts/ingest_ejercicio_2.sh
        """,
    )

    # Disparar el DAG hijo

    trigger_target = TriggerDagRunOperator(
        task_id="trigger_target",
        trigger_dag_id="dag_hijo",
        execution_date="{{ ds }}",
        reset_dag_run=True,
        wait_for_completion=True,
        poke_interval=30,
    )


    # Inicio del flujo de tareas Extraccion y disparo del DAG hijo

    inicio_proceso >> ingest >> trigger_target
```

Desde **bash:**

```
hadoop@615cf53bef6c:~/airflow/dags$ cat > DAG_padre_ejercicio_2_trabajo_final.py
#####################################################################
#
# DAG padre creado para el ejercicio 2 Trabajo Final
# /home/hadoop/airflow/dags/DAG_padre_ejercicio_2_trabajo_final.py
#
#####################################################################

# Importamos librerias

from datetime import timedelta
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.dates import days_ago
from airflow.operators.trigger_dagrun import TriggerDagRunOperator


# Definimos los argumentos por defecto del DAG
args = {
    "owner": "airflow",
}

with DAG(
    dag_id="dag_padre",
    default_args=args,
    schedule_interval="0 0 * * *",
    start_date=days_ago(1),
    catchup=False,
    dagrun_timeout=timedelta(minutes=60),
    tags=["ingest"],
) as dag:

    inicio_proceso = DummyOperator(task_id="inicio_proceso")


    # Tareas Ingest

    ingest = BashOperator(
        task_id="ingest",
        bash_command="""
            set -e
            /bin/bash /home/hadoop/scripts/ingest_ejercicio_2.sh
        """,
    )

    # Disparar el DAG hijo

    trigger_target = TriggerDagRunOperator(
        task_id="trigger_target",
        trigger_dag_id="dag_hijo",
        execution_date="{{ ds }}",
        reset_dag_run=True,
        wait_for_completion=True,
        poke_interval=30,
    )


    # Inicio del flujo de tareas Extraccion y disparo del DAG hijo

    inicio_proceso >> ingest >> trigger_target
hadoop@615cf53bef6c:~/airflow/dags$
```

Y el script correspondiente al DAG hijo, alojado en
/home/hadoop/airflow/dags/DAG_hijo_ejercicio_2_trabajo_final.py

DAG_hijo_ejercicio_2_trabajo_final.py > ...

```python
#################################################################
#
# DAG hijo creado para el ejercicio 2 Trabajo Final
# /home/hadoop/airflow/dags/DAG_hijo_ejercicio_2_trabajo_final.py
#
#################################################################

# Importamos librerias

from datetime import timedelta
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.dates import days_ago


# Definimos los argumentos por defecto del DAG
args = {
    "owner": "airflow",
}

with DAG(
    dag_id="dag_hijo",
    default_args=args,
    schedule_interval="0 0 * * *",
    start_date=days_ago(1),
    catchup=False,
    dagrun_timeout=timedelta(minutes=60),
    tags=["transform_load"],
) as dag:


    # Tareas Transform - Load

    transform_load = BashOperator(
        task_id='transform_load',
        bash_command="""
            set -e
            /home/hadoop/spark/bin/spark-submit \
                --master local[*] \
                --deploy-mode client \
                --files /home/hadoop/hive/conf/hive-site.xml \
                /home/hadoop/scripts/transform_load_8.py
        """,
    )

    finaliza_proceso = DummyOperator(task_id="finaliza_proceso")

    # Finalizacion del flujo de tareas transform & load
    transform_load >> finaliza_proceso
```

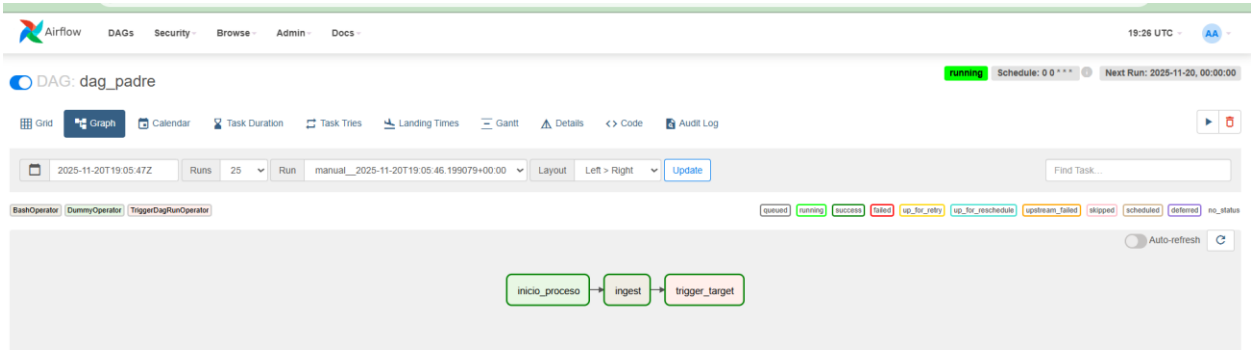
Desde **bash:**

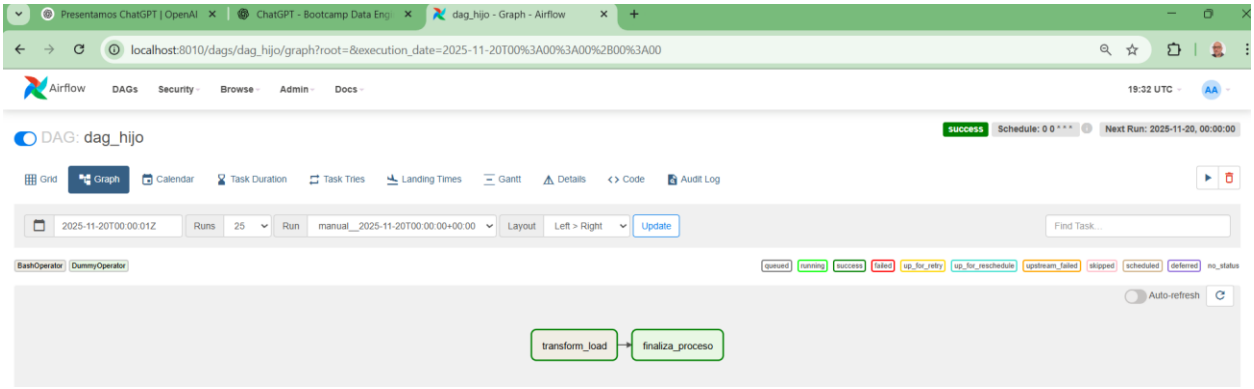
```
hadoop@615cf53bef6c:~/airflow/dags$ cat > DAG_hijo_ejercicio_2_trabajo_final.py
#####################################################################
#
# DAG hijo creado para el ejercicio 2 Trabajo Final
# /home/hadoop/airflow/dags/DAG_hijo_ejercicio_2_trabajo_final.py
#
#####################################################################

# Importamos librerias

from datetime import timedelta
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.dates import days_ago


# Definimos los argumentos por defecto del DAG
args = {
    "owner": "airflow",
}

with DAG(
    dag_id="dag_hijo",
    default_args=args,
    schedule_interval="0 0 * * *",
    start_date=days_ago(1),
    catchup=False,
    dagrun_timeout=timedelta(minutes=60),
    tags=["transform_load"],
) as dag:


    # Tareas Transform - Load

    transform_load = BashOperator(
        task_id='transform_load',
        bash_command="""
            set -e
            /home/hadoop/spark/bin/spark-submit \
                --master local[*] \
                --deploy-mode client \
                --files /home/hadoop/hive/conf/hive-site.xml \
                /home/hadoop/scripts/transform_load_8.py
        """,
    )

    finaliza_proceso = DummyOperator(task_id="finaliza_proceso")

    # Finalizacion del flujo de tareas transform & load
    transform_load >> finaliza_proceso
```

Asignamos permisos a cada archivo:

```
hadoop@615cf53bef6c:~/airflow/dags$ chmod 777 DAG_padre_ejercicio_2_trabajo_final.py
hadoop@615cf53bef6c:~/airflow/dags$ chmod 777 DAG_hijo_ejercicio_2_trabajo_final.py
hadoop@615cf53bef6c:~/airflow/dags$ ls -l
total 48
-rwxrwxrwx 1 hadoop hadoop 1679 Nov 16 12:35 DAG_ejercicio_1_trabajo_final.py
-rw-rw-r-- 1 hadoop hadoop 1663 Oct 19 12:59 DAG_ejercicio_5.py
-rwxrwxrwx 1 hadoop hadoop 2893 Oct 25 01:49 DAG_ejercicio_5_clase_8.py
-rwxrwxrwx 1 hadoop hadoop 1742 Nov  7 15:16 DAG_ejercicio_7_clase_10.py
-rwxrwxrwx 1 hadoop hadoop 4009 Nov  2 23:53 DAG_ejercicio_7_clase_9.py
-rwxrwxrwx 1 hadoop hadoop 1356 Nov 20 15:23 DAG_hijo_ejercicio_2_trabajo_final.py
-rwxrwxrwx 1 hadoop hadoop 1505 Nov 20 15:24 DAG_padre_ejercicio_2_trabajo_final.py
-rw-rw-r-- 1 hadoop hadoop 1496 Oct 31 21:15 DAG_str_weather.py
-rwxrwxrwx 1 hadoop hadoop  542 Nov  1 00:14 DAG_weather2kafka.py
drwxrwxrwx 2 hadoop hadoop 4096 Nov 20 15:27 __pycache__
-rw-rw-r-- 1 hadoop hadoop 1079 May  1  2022 example-DAG.py
-rw-rw-r-- 1 hadoop hadoop 1024 May  5  2022 ingest-transform.py
hadoop@615cf53bef6c:~/airflow/dags$
```

Ejecutamos los DAGs en Airflow:





Se observa que el DAG padre llama correctamente al DAG hijo y se finaliza el proceso satisfactoriamente.

5. Por medio de consultas SQL al data-warehouse, mostrar:

   a. Cantidad de alquileres de autos, teniendo en cuenta sólo los vehículos ecológicos (fuelType hibrido o eléctrico) y con un rating de al menos 4.

   b. los 5 estados con menor cantidad de alquileres (mostrar query y visualización)

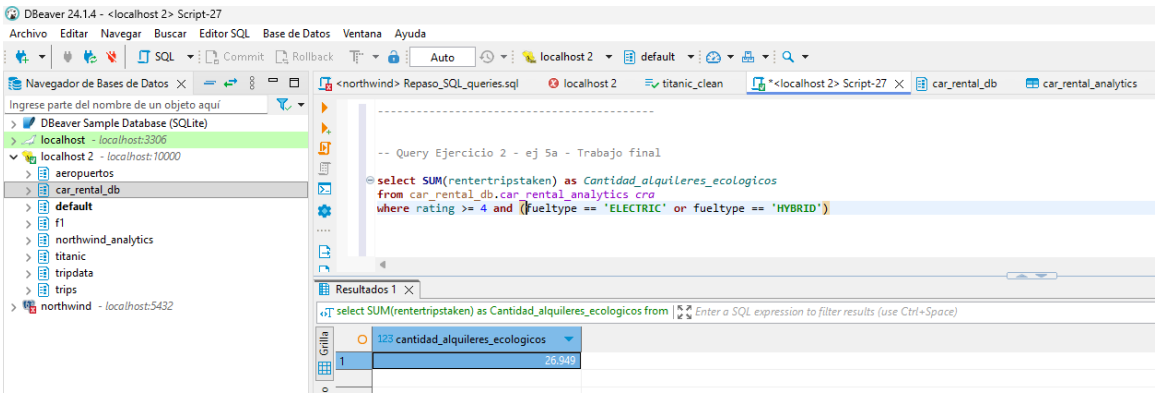
   c. los 10 modelos (junto con su marca) de autos más rentados (mostrar query y visualización)

   d. Mostrar por año, cuántos alquileres se hicieron, teniendo en cuenta automóviles fabricados desde 2010 a 2015

   e. las 5 ciudades con más alquileres de vehículos ecológicos (fuelType hibrido o electrico)

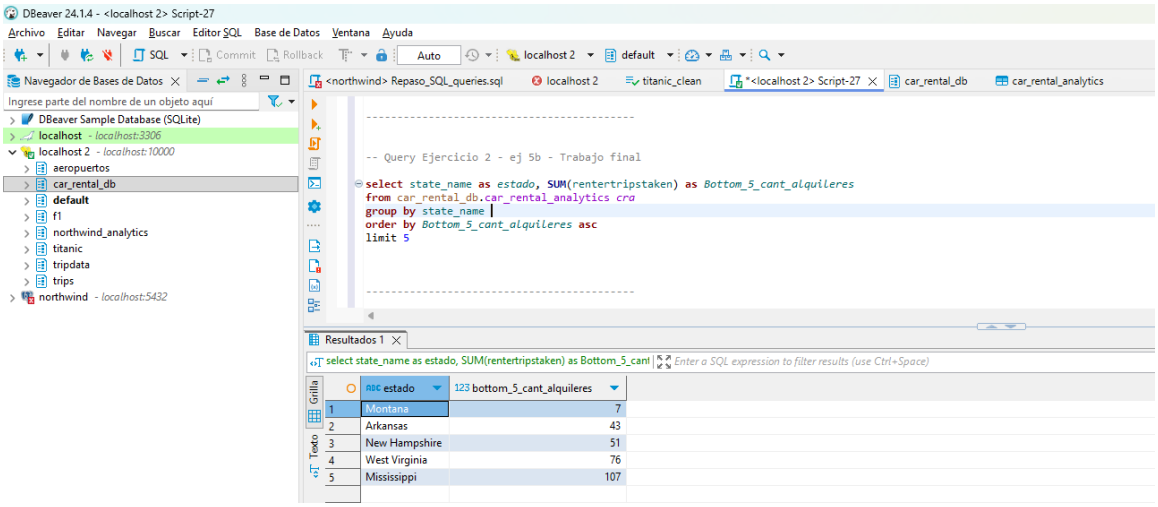
   f. el promedio de reviews, segmentando por tipo de combustible

a)



La consulta muestra que hay 26949 registros que corresponden a la cantidad de autos **ecológicos** (tipo de combustible Híbrido o eléctrico) con la puntuación deseada.

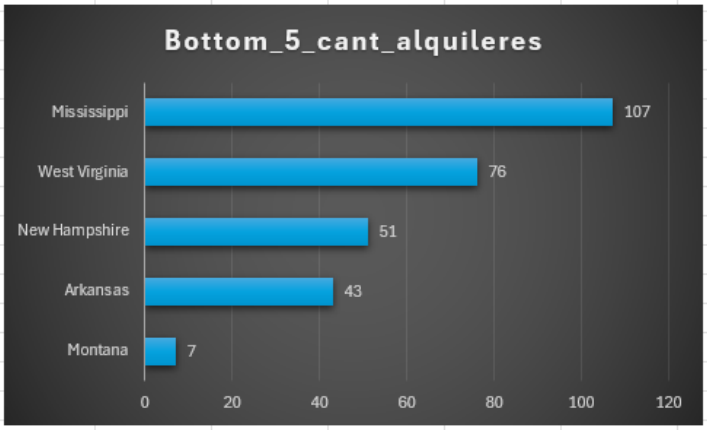
b)



Para la visualización, se procedió a exportar la tabla generada por la consulta en DBeaver **(Tabla ejercicio 2 ej5b - Trabajo final.csv)** para luego importarla en Excel como csv.

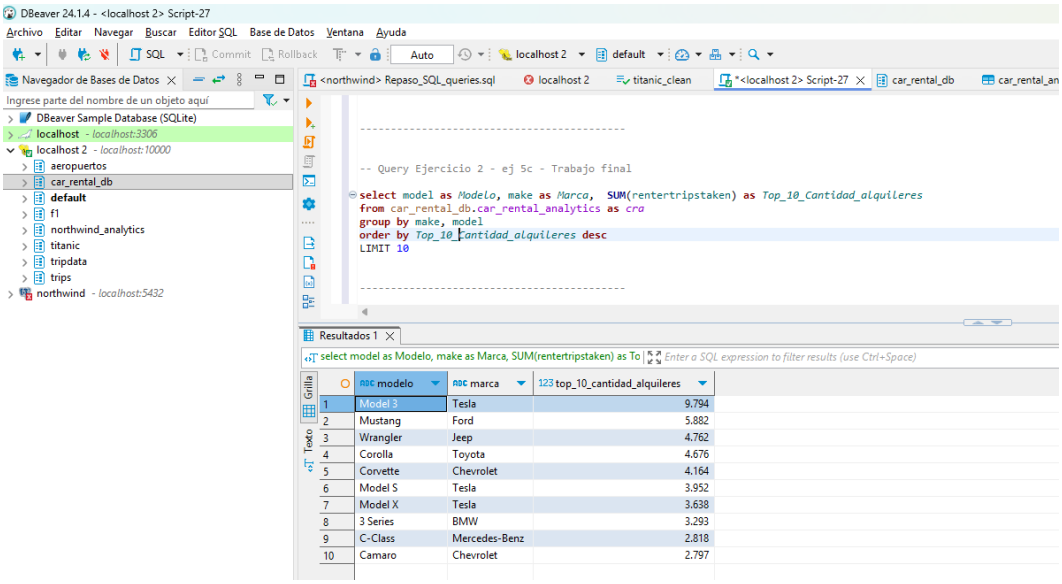
Una vez allí, se procedió a insertar gráfico y personalizarlo:

estado	bottom_5_cant_alquileres
Montana	7
Arkansas	43
New Hampshire	51
West Virginia	76
Mississippi	107



El gráfico muestra dentro del BOTTOM 5, a Montana, como Estado con menor cantidad de alquileres (7).

c)



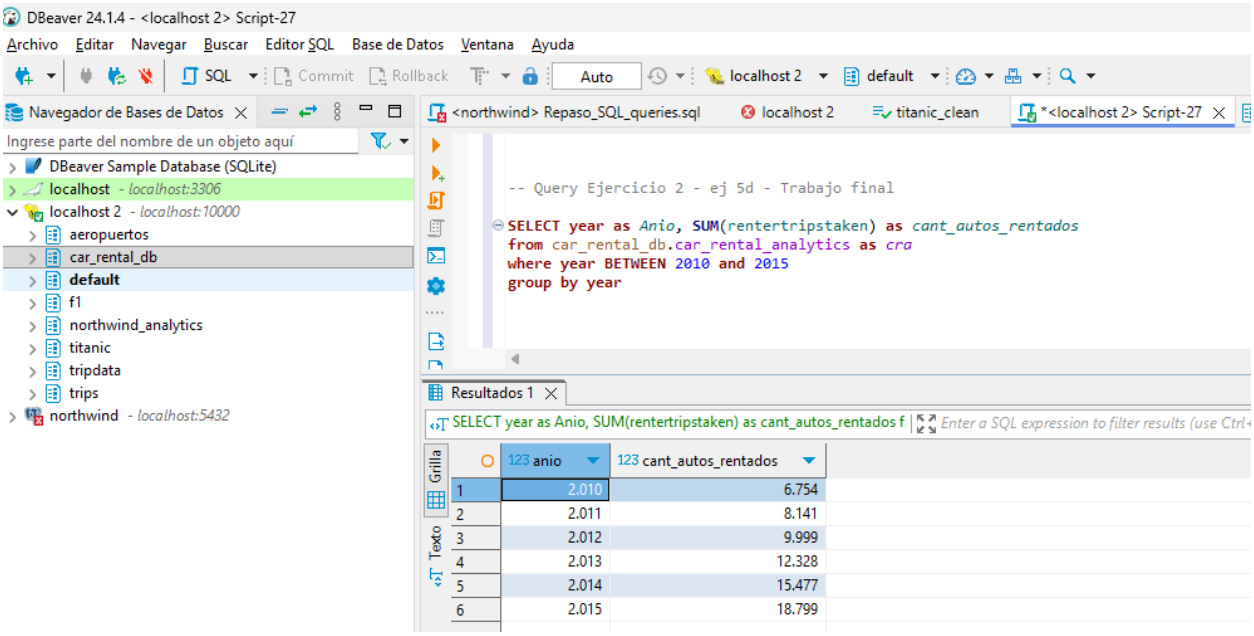
Para la visualización, repetimos el procedimiento del inciso anterior:

model	marca	top_10_cantidad_alquileres
Model 3	Tesla	9794
Mustang	Ford	5882
Wrangler	Jeep	4762
Corolla	Toyota	4676
Corvette	Chevrolet	4164
Model S	Tesla	3952
Model X	Tesla	3638
3 Series	BMW	3293
C-Class	Mercedes-Benz	2818
Camaro	Chevrolet	2797

En este caso, la tabla exportada desde DBeaver fue **Tabla ejercicio 2 ej5c - Trabajo final.csv**

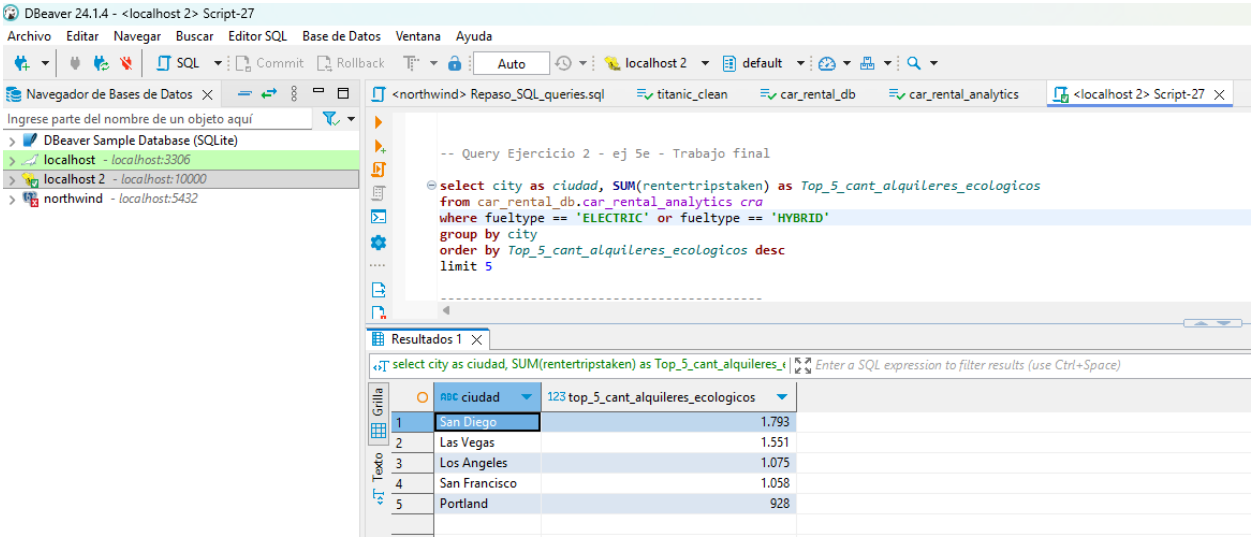
Se observa que en el TOP 10, al **Model 3** de **Tesla** es el auto que mayor cantidad de alquileres registra en el periodo considerado en la base de datos (Julio 2020, según el DataCard del Dataset original de Kaggle).

d)



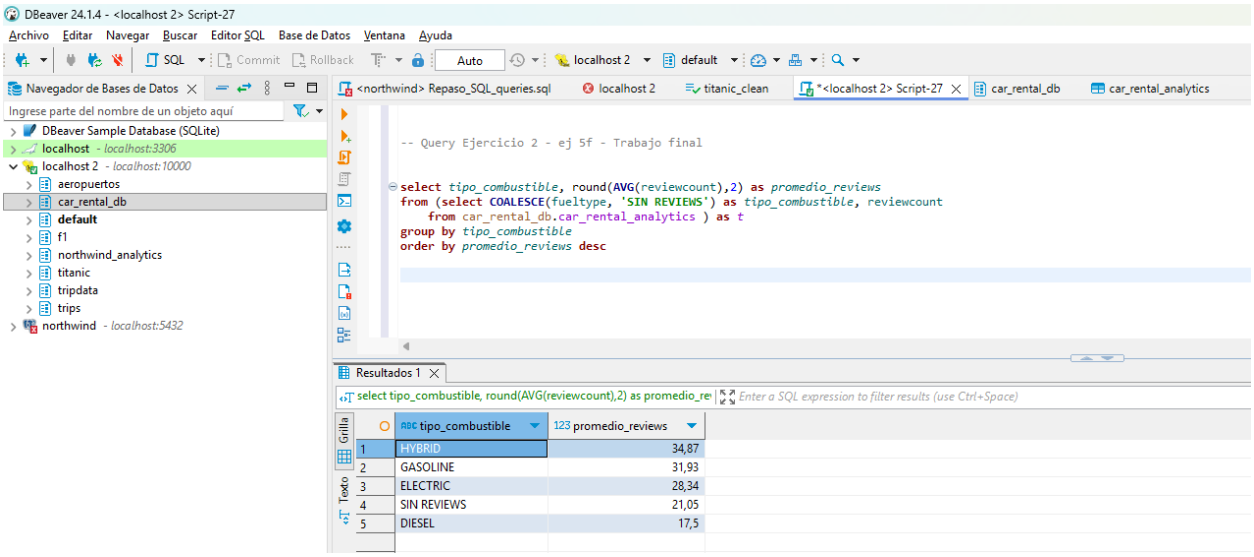
La consulta muestra que, entre 2010 y 2015, la cantidad de alquileres ha ido en aumento sostenido, registrando el máximo de 18799 unidades en 2015.

e)



La consulta muestra que San Diego lidera el TOP 5 de estados con mayor cantidad de alquileres de autos ecológicos, con 1793 unidades en el periodo considerado por el dataset.

f)



De acuerdo con la consulta, los clientes son mas propensos a contestar reviews acerca de los autos Híbridos (para bien o para mal). Creo que sería mejor complementar este tipo de KPI con otros más robustos, como por ejemplo la **tasa agregada de reviews por tipo de combustible en cada segmento**:

$$\text{tasa\_segmento} = \frac{\sum(\text{reviewcount})}{\sum(\text{rentertripstaken})} \times 100$$

lo que nos da info acerca de que fracción de alquileres en cada segmento termina en review.

6. Elabore sus conclusiones y recomendaciones sobre este proyecto.

Los datos de la tabla provienen de un proceso de *web scraping*, por lo que su calidad puede no ser óptima ni representar con precisión la realidad del mercado de alquiler de autos (por ejemplo, debido a sesgos de la plataforma fuente o a restricciones de acceso a la información).

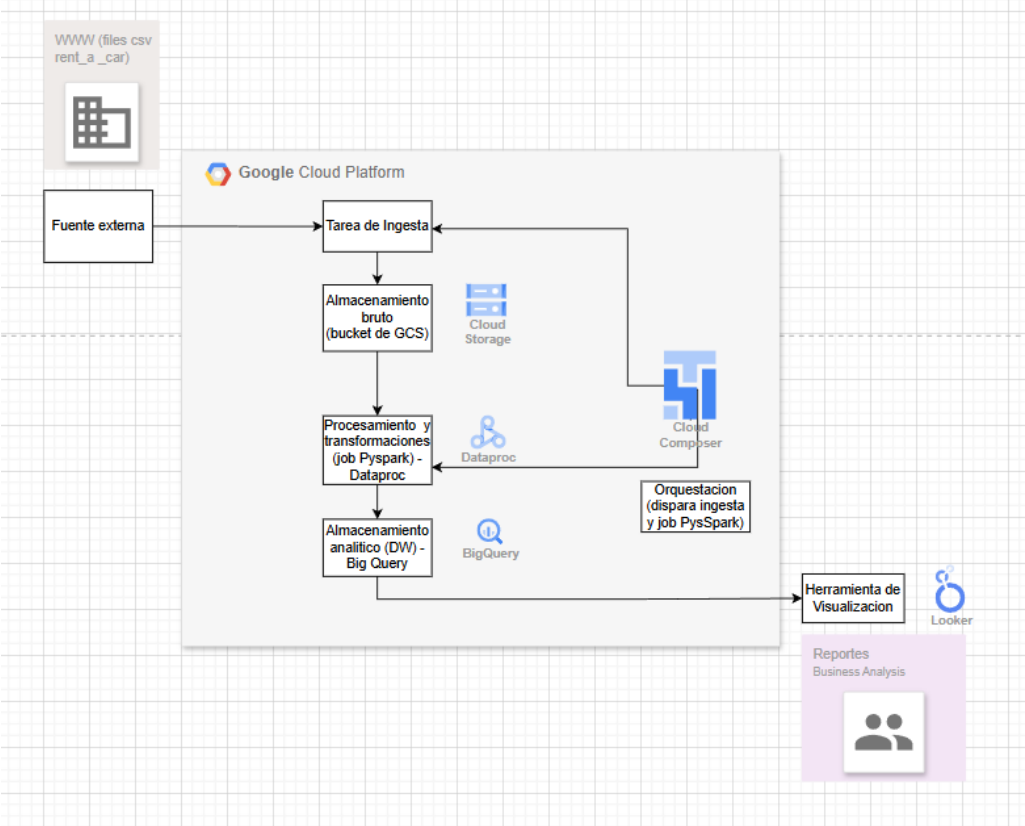
Desde una perspectiva de consultoría, una alternativa superadora sería trabajar con datos transaccionales de empresas de alquiler tradicionales, con el fin de optimizar tarifas dinámicas por ciudad y tipo de vehículo, mejorar la utilización de la flota, identificar patrones temporales y geográficos de demanda y detectar segmentos de usuarios de alto valor o de mayor riesgo. Ello podría traducirse, para estas empresas, en mayores ingresos (vía ocupación y tarifa media), mejor planificación de flota y precios, e informes periódicos con KPI accionables.

Como marco de colaboración, se podrían acordar entregables concretos —un informe ejecutivo con conclusiones y recomendaciones, y dashboards interactivos (por ejemplo, en Looker Studio o Power BI)— junto con un esquema de compensación y acceso acotado a datos transaccionales, respetando políticas de confidencialidad y los resguardos de seguridad correspondientes.

En la situación actual, la tabla disponible permite analizar el desempeño de cada vehículo a nivel agregado (alquileres totales, número de reseñas, rating promedio, etc.), lo que hace posible construir KPI sólidos por segmento (tipo de combustible, ciudad, estado, marca o modelo).
No obstante, ***al no disponer de información a nivel de cada alquiler individual***, el modelo de datos es insuficiente para estudiar en detalle la dinámica temporal de la demanda, la duración de los viajes, los patrones origen–destino o el desempeño económico de cada operación.
Por ello, se recomienda utilizar el esquema actual para análisis descriptivos agregados asociados (como los hechos en las queries del ejercicio 5), dejando explícitas sus limitaciones y ***evolucionar hacia un modelo más detallado basado en registros de alquiler individual*** (incluyendo campos como *pickup/dropoff_datetime* y *pickup/dropoff_city/state*). Esta ampliación permitiría definir KPI más alineados con las necesidades del negocio (duración de alquileres, estacionalidad, variación horaria, ingresos y rentabilidad por segmento) y transformar el proyecto en un sistema de analítica más completo y que permitiría responder a más preguntas de negocio.

7. Proponer una arquitectura alternativa para este proceso ya sea con herramientas on premise o cloud (Si aplica)

Proponemos la siguiente arquitectura, basada en Google Cloud Platform (GCP), idéntica a la del ejercicio 1:



La tarea de ingesta se hace a través de un archivo similar a **ingest_ejercicio_2.sh** alojado en un bucket de Google Cloud Storage. Este proceso lo dispara Cloud Composer.

El job de PySpark (archivo similar a **transform_load_8.py**) lo ejecuta Dataproc y también es disparado por Cloud Composer.

Las tareas de consulta en SQL se realizan dentro de Big Query y finalmente, desde afuera de GCP, podemos contectarnos con alguna herramienta de visualización como Looker, para generar reportes (Dashboards).