

Ejercicio 1

1. Hacer ingest de los siguientes files relacionados con transporte aéreo de Argentina

```
hadoop@f769ed737105:~/scripts$ ./ingest_aeropuertos_arg.sh
Descargando informe del ministerio 2021...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 30.8M  100 30.8M    0     0 2808k      0  0:00:11  0:00:11 --:--:-- 3173k
Descargando informe del ministerio 2022...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 21.7M  100 21.7M    0     0 3108k      0  0:00:07  0:00:07 --:--:-- 3864k
Descargando detalles de aeropuertos...
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 132k  100 132k    0     0 118k      0  0:00:01  0:00:01 --:--:-- 118k
Todos los archivos han sido descargados y subidos a hdfs://172.17.0.2:9000/ingest en HDFS.
```

```
hadoop@f769ed737105:~/scripts$ hdfs dfs -ls /ingest
Found 9 items
-rw-r--r--  1 hadoop supergroup 32322556 2024-09-18 10:32 /ingest/2021-informe-ministerio.csv
-rw-r--r--  1 hadoop supergroup 22833520 2024-09-18 10:32 /ingest/202206-informe-ministerio.csv
-rw-r--r--  1 hadoop supergroup  136007 2024-09-18 10:33 /ingest/aeropuertos_detalle.csv
```

2. Crear 2 tablas en el datawarehouse, una para los vuelos realizados en 2021 y 2022 (2021-informe-ministerio.csv y 202206-informe-ministerio) y otra tabla para el detalle de los aeropuertos (aeropuertos_detalle.csv)

```
hive> show tables;
OK
aeropuerto_detalle_tabla
aeropuerto_tabla
Time taken: 0.031 seconds, Fetched: 2 row(s)
hive> █
```

```
hive> describe aeropuerto_tabla;
OK
fecha                date
horautc              string
clase_de_vuelo       string
clasificacion_de_vuelo string
tipo_de_movimiento   string
aeropuerto           string
origen_destino        string
aerolinea_nombre     string
aeronave             string
pasajeros            int
```

```
hive> describe aeropuerto_detalle_tabla;
OK
aeropuerto           string
oac                  string
iata                  string
tipo                  string
denominacion          string
coordenadas           string
latitud               string
longitud              string
elev                  float
uom_elev              string
ref                   string
distancia_ref          float
direccion_ref          string
condicion              string
control               string
region                string
uso                    string
trafico                string
sna                    string
concesionado           string
provincia              string
```

3. Realizar un proceso automático orquestado por airflow que ingeste los archivos previamente mencionados entre las fechas 01/01/2021 y 30/06/2022 en las dos columnas creadas.

Los archivos 202206-informe-ministerio.csv y 202206-informe-ministerio.csv → en la tabla aeropuerto_tabla

El archivo aeropuertos_detalle.csv → en la tabla aeropuerto_detalle_tabla

```
dag_id='aeropuerto_arg_dag',
default_args= args,
description='Pipeline con ingest y process',
schedule_interval='0 0 * * *',
start_date=days_ago(2),
dagrun_timeout=timedelta(minutes=60),
tags=['ingest', 'transform'],
params={"example_key": "example_value"},
) as dag:

    inicia_proceso = DummyOperator(
        task_id='inicia_proceso',
    )

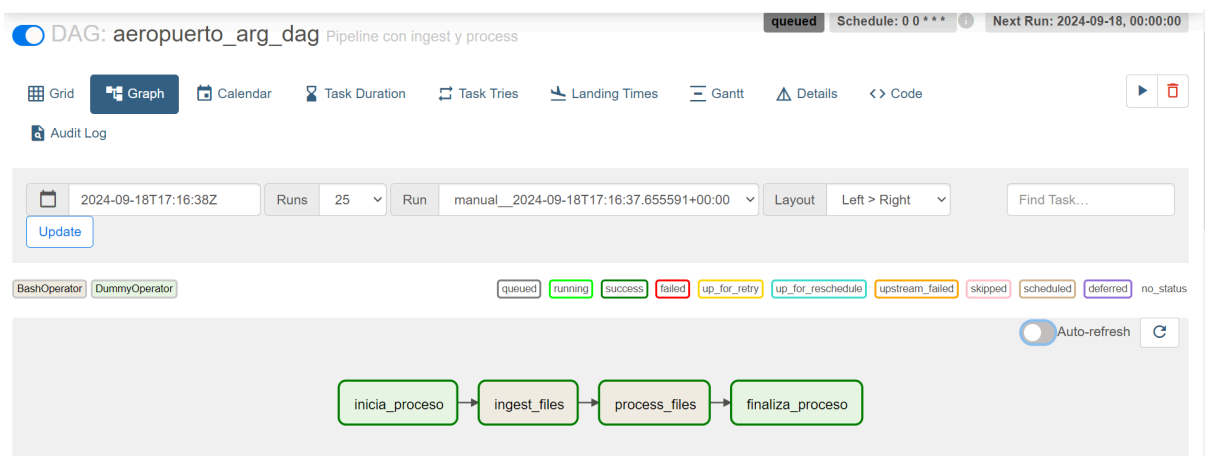
    ingest_files = BashOperator(
        task_id='ingest_files',
        bash_command='usr/bin/sh /home/hadoop/scripts/ingest_aeropuertos_arg.sh ',
    )

    process_files = BashOperator(
        task_id='process_files',
        bash_command='ssh hadoop@172.17.0.2 /home/hadoop/spark/bin/spark-submit --files /home/hadoop/hive/conf/hive-site.xml /home/hadoop/scripts/aeropuerto_arg_trans.py ',
    )

    finaliza_proceso = DummyOperator(
        task_id='finaliza_proceso',
    )

    inicia_proceso >> ingest_files >> process_files >> finaliza_proceso

if __name__ == "__main__":
    dag.cli()
```



```
hive> select * from aeropuerto_tabla limit 5;
OK
2021-01-01    00:02    Vuelo Privado con Matrícula Nacional    Domestico    Despegue    P
AR    ROS    0    PA-PA-28-181    0
2021-01-01    00:24    Regular Domestico    Aterrizaje    EZE    GRA    AEROLINEAS ARGENTINAS
SA    BO-B737-8MB    70
2021-01-01    00:26    Regular Domestico    Aterrizaje    EZE    ECA    AEROLINEAS ARGENTINAS
SA    BO-737-800    70
2021-01-01    00:29    Regular Domestico    Aterrizaje    EZE    SAL    AEROLINEAS ARGENTINAS
SA    BO-B-737-76N    12
2021-01-01    00:37    Regular Domestico    Aterrizaje    EZE    TUC    AEROLINEAS ARGENTINAS
SA    EMB-ERJ190100IGW    26
Time taken: 0.111 seconds, Fetched: 5 row(s)
```

```
hive> select * from aeropuerto_detalle_tabla limit 5;
OK
ACB    NULL    NULL    Aeródromo    CORONEL BOGADO/AGROSERVICIOS    "33°16'20""S 60°34'14""W"    -
60.57066000    -33.27226000    44.0    Metros    Coronel Bogado    6.0    NE    PRIVADO    N
OCONTROL    RACE    AEROAPP Nacional    NO    NO    SANTA FÉ
ACH    NULL    NULL    Aeródromo    GENERAL ACHA    "37°24' 6""S 64°36'49""W"    -
64.61351000    -37.40164000    277.0    Metros    General Acha    3.0    SO    PUBLICO    N
OCONTROL    RACE    CIVIL Nacional    NO    NO    LA PAMPA
ACM    NULL    NULL    Aeródromo    ARRECIFES/LA CURA MALAL    "34° 4'33""S 60° 8'30""W"    -
60.14170000    -34.07574000    37.0    Metros    Arrecifes    4.0    OSO    PRIVADO    N
OCONTROL    RACE    CIVIL Nacional    NO    NO    BUENOS AIRES
ADO    SAWD    PUD    Aeródromo    PUERTO DESEADO    "47°44' 6""S 65°54'15""W"    -
65.90410000    -47.73511000    82.0    Metros    Puerto Deseado    2.0    N    PUBLICO    A
ERADIO    RASU    CIVIL Nacional    NO    NO    SANTA CRUZ
ADT    NULL    NULL    Aeródromo    BANDERA/AGROSERVICIOS DOÑA TERESA    "28°51'19""S 62°15'5
3""W"    -62.26462000    -28.85541000    75.0    Metros    Bandera    4.0    N    PRIVADO    N
OCONTROL    RANO    AEROAPP Nacional    NO    NO    SANTIAGO DEL ESTERO
Time taken: 0.147 seconds, Fetched: 5 row(s)
hive>
```

4. Realizar las siguientes transformaciones en los pipelines de datos:

- Eliminar la columna inhab ya que no se utilizará para el análisis
- Eliminar la columna fir ya que no se utilizará para el análisis
- Eliminar la columna "calidad del dato" ya que no se utilizará para el análisis
- Filtrar los vuelos internacionales ya que solamente se analizarán los vuelos domésticos
- En el campo pasajeros si se encuentran campos en Null convertirlos en 0 (cero)
- En el campo distancia_ref si se encuentran campos en Null convertirlos en 0 (cero)

```
hadoop@f769ed737105:~/scripts$ cat aeropuerto_arg_trans.py
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import col, split, concat_ws, to_date

sc = SparkContext('local')
spark = SparkSession(sc)
from pyspark.sql import HiveContext
hc = HiveContext(sc)

# Lectura de archivos
informe_2021 = spark.read.csv("hdfs://172.17.0.2:9000/ingest/2021-informe-ministerio.csv", sep=";",
header=True)
informe_2022 = spark.read.csv("hdfs://172.17.0.2:9000/ingest/202206-informe-ministerio.csv", sep=";",
, header=True)
ae_detalle = spark.read.csv("hdfs://172.17.0.2:9000/ingest/aeropuertos_detalle.csv", sep=";", header
=True)

# Unión de 2021 y 2022
informe_2021_2022 = informe_2021.unionByName(informe_2022)

# Transformaciones a informe_2021_2022
informe_2021_2022 = informe_2021_2022.drop("Calidad dato")
informe_2021_2022 = informe_2021_2022.filter(informe_2021_2022["Clasificación Vuelo"].isin("Domestic
o", "Doméstico"))
informe_2021_2022 = informe_2021_2022.fillna({"Pasajeros": 0})

# Modificación del string de la columna fecha a formato yyyy-MM-dd
informe_2021_2022 = informe_2021_2022.withColumn(
    "fecha",
    concat_ws("-", split(col("fecha"), "/")[2], split(col("fecha"), "/")[1], split(col("fecha"), "/"
)[0])
)

# Transformaciones ae_detalle
ae_detalle = ae_detalle.drop("inhab", "fir")
ae_detalle = ae_detalle.fillna({"distancia_ref": 0})
```

```

# Renombrar columnas de informe_2021_2022
informe_2021_2022 = informe_2021_2022 \
    .withColumnRenamed("Fecha", "fecha") \
    .withColumnRenamed("Hora UTC", "horaUTC") \
    .withColumnRenamed("Clase de Vuelo (todos los vuelos)", "clase_de_vuelo") \
    .withColumnRenamed("Clasificación Vuelo", "clasificacion_de_vuelo") \
    .withColumnRenamed("Tipo de Movimiento", "tipo_de_movimiento") \
    .withColumnRenamed("Aeropuerto", "aeropuerto") \
    .withColumnRenamed("Origen / Destino", "origen_destino") \
    .withColumnRenamed("Aerolinea Nombre", "aerolinea_nombre") \
    .withColumnRenamed("Aeronave", "aeronave") \
    .withColumnRenamed("Pasajeros", "pasajeros")

# Cast de columnas de informe_2021_2022
informe_2021_2022 = informe_2021_2022 \
    .withColumn("fecha", to_date(col("fecha"), "yyyy-MM-dd")) \
    .withColumn("horaUTC", col("horaUTC").cast("string")) \
    .withColumn("clase_de_vuelo", col("clase_de_vuelo").cast("string")) \
    .withColumn("clasificacion_de_vuelo", col("clasificacion_de_vuelo").cast("string")) \
    .withColumn("tipo_de_movimiento", col("tipo_de_movimiento").cast("string")) \
    .withColumn("aeropuerto", col("aeropuerto").cast("string")) \
    .withColumn("origen_destino", col("origen_destino").cast("string")) \
    .withColumn("aerolinea_nombre", col("aerolinea_nombre").cast("string")) \
    .withColumn("aeronave", col("aeronave").cast("string")) \
    .withColumn("pasajeros", col("pasajeros").cast("int"))

# Renombrar columnas de ae_detalle
ae_detalle = ae_detalle \
    .withColumnRenamed("local", "aeropuerto") \
    .withColumnRenamed("oaci", "oac") \
    .withColumnRenamed("iata", "iata") \
    .withColumnRenamed("tipo", "tipo") \
    .withColumnRenamed("denominacion", "denominacion") \
    .withColumnRenamed("coordenadas", "coordenadas") \
    .withColumnRenamed("latitud", "latitud") \
    .withColumnRenamed("longitud", "longitud") \
    .withColumnRenamed("elev", "elev") \
    .withColumnRenamed("uom_elev", "uom_elev") \

```

```

.withColumnRenamed("ref", "ref") \
.withColumnRenamed("distancia_ref", "distancia_ref") \
.withColumnRenamed("direccion_ref", "direccion_ref") \
.withColumnRenamed("condicion", "condicion") \
.withColumnRenamed("control", "control") \
.withColumnRenamed("region", "region") \
.withColumnRenamed("uso", "uso") \
.withColumnRenamed("trafico", "trafico") \
.withColumnRenamed("sna", "sna") \
.withColumnRenamed("concesionado", "concesionado") \
.withColumnRenamed("provincia", "provincia")

# Cast de ae_detalle
ae_detalle = ae_detalle \
.withColumn("aeropuerto", col("aeropuerto").cast("string")) \
.withColumn("oac", col("oac").cast("string")) \
.withColumn("iata", col("iata").cast("string")) \
.withColumn("tipo", col("tipo").cast("string")) \
.withColumn("denominacion", col("denominacion").cast("string")) \
.withColumn("coordenadas", col("coordenadas").cast("string")) \
.withColumn("latitud", col("latitud").cast("string")) \
.withColumn("longitud", col("longitud").cast("string")) \
.withColumn("elev", col("elev").cast("float")) \
.withColumn("uom_elev", col("uom_elev").cast("string")) \
.withColumn("ref", col("ref").cast("string")) \
.withColumn("distancia_ref", col("distancia_ref").cast("float")) \
.withColumn("direccion_ref", col("direccion_ref").cast("string")) \
.withColumn("condicion", col("condicion").cast("string")) \
.withColumn("control", col("control").cast("string")) \
.withColumn("region", col("region").cast("string")) \
.withColumn("uso", col("uso").cast("string")) \
.withColumn("trafico", col("trafico").cast("string")) \
.withColumn("sna", col("sna").cast("string")) \
.withColumn("concesionado", col("concesionado").cast("string")) \
.withColumn("provincia", col("provincia").cast("string"))

# Inserción en Hive
informe_2021_2022.write.insertInto('aeropuertos_arg.aeropuerto_tabla')

ae_detalle.write.insertInto('aeropuertos_arg.aeropuerto_detalles_tabla')

# Detener Spark
spark.stop()

hadoop@f769ed737105:~/scripts$ █

```

5. Mostrar mediante una impresión de pantalla, que los tipos de campos de las tablas sean los solicitados en el datawarehouse (ej: fecha date, aeronave string, pasajeros integer, etc.)

```
>>> informe_2021_2022.printSchema()
root
 |-- fecha: date (nullable = true)
 |-- horaUTC: string (nullable = true)
 |-- clase_de_vuelo: string (nullable = true)
 |-- clasificacion_de_vuelo: string (nullable = true)
 |-- tipo_de_movimiento: string (nullable = true)
 |-- aeropuerto: string (nullable = true)
 |-- origen_destino: string (nullable = true)
 |-- aerolinea_nombre: string (nullable = true)
 |-- aeronave: string (nullable = true)
 |-- pasajeros: integer (nullable = true)
```

```
>>> ae_detalle.printSchema()
root
 |-- aeropuerto: string (nullable = true)
 |-- oac: string (nullable = true)
 |-- iata: string (nullable = true)
 |-- tipo: string (nullable = true)
 |-- denominacion: string (nullable = true)
 |-- coordenadas: string (nullable = true)
 |-- latitud: string (nullable = true)
 |-- longitud: string (nullable = true)
 |-- elev: float (nullable = true)
 |-- uom_elev: string (nullable = true)
 |-- ref: string (nullable = true)
 |-- distancia_ref: float (nullable = true)
 |-- direccion_ref: string (nullable = true)
 |-- condicion: string (nullable = true)
 |-- control: string (nullable = true)
 |-- region: string (nullable = true)
 |-- uso: string (nullable = true)
 |-- trafico: string (nullable = true)
 |-- sna: string (nullable = true)
 |-- concesionado: string (nullable = true)
 |-- provincia: string (nullable = true)
```



6. Determinar la cantidad de vuelos entre las fechas 01/12/2021 y 31/01/2022. Mostrar consulta y Resultado de la query

```
--6
SELECT COUNT(*) as cantidad_de_vuelos
from aeropuertos_arg.aeropuerto_tabla
WHERE
    fecha BETWEEN '2021-12-01' AND '2022-01-31'
and tipo_de_movimiento = 'Despegue';
```

```
--7
```

```
select sum(pasajeros) as cantidad_pasajeros
from aeropuertos_arg.aeropuerto_tabla
WHERE fecha BETWEEN '2021-01-01' AND '2022-06-30'
AND aerolinea_nombre = 'AEROLINEAS ARGENTINAS SA'
;
```

results 1 ×

SELECT COUNT(*) as cantidad_de_vuelos from a  Enter a SQL expression to filter results (u.

	123 cantidad_de_vuelos
1	41,710

7. Cantidad de pasajeros que viajaron en Aerolíneas Argentinas entre el 01/01/2021 y 30/06/2022. Mostrar consulta y Resultado de la query

```
--7
select sum(pasajeros) as cantidad_pasajeros
from aeropuertos_arg.aeropuerto_tabla
WHERE fecha BETWEEN '2021-01-01' AND '2022-06-30'
AND aerolinea_nombre = 'AEROLINEAS ARGENTINAS SA'
;

--8

SELECT
    fecha
```

results 1 ×

select sum(pasajeros) as cantidad_pasajeros from

Enter a SQL expression to filter results (use Ctrl+Space)

	123 cantidad_pasajeros
1	10,648,641

8. Mostrar fecha, hora, código aeropuerto salida, ciudad de salida, código de aeropuerto de arribo, ciudad de arribo, y cantidad de pasajeros de cada vuelo, entre el 01/01/2022 y el 30/06/2022 ordenados por fecha de manera descendiente. Mostrar consulta y Resultado de la query

```
--8
SELECT
    fecha,
    horaUTC AS hora,
    CASE
        WHEN tipo_de_movimiento = 'Despegue' THEN aeropuerto
        ELSE NULL
    END AS codigo_aeropuerto_salida,
    CASE
        WHEN tipo_de_movimiento = 'Despegue' THEN origen_destino
        ELSE NULL
    END AS ciudad_salida,
    CASE
        WHEN tipo_de_movimiento = 'Aterrizaje' THEN aeropuerto
        ELSE NULL
    END AS codigo_aeropuerto_arribo,
    CASE
        WHEN tipo_de_movimiento = 'Aterrizaje' THEN origen_destino
        ELSE NULL
    END AS ciudad_arribo,
    pasajeros
FROM
    aeropuertos_arg.aeropuerto_tabla
WHERE
    fecha BETWEEN '2022-01-01' AND '2022-06-30'
ORDER BY
    fecha DESC;
```

Results 1 x

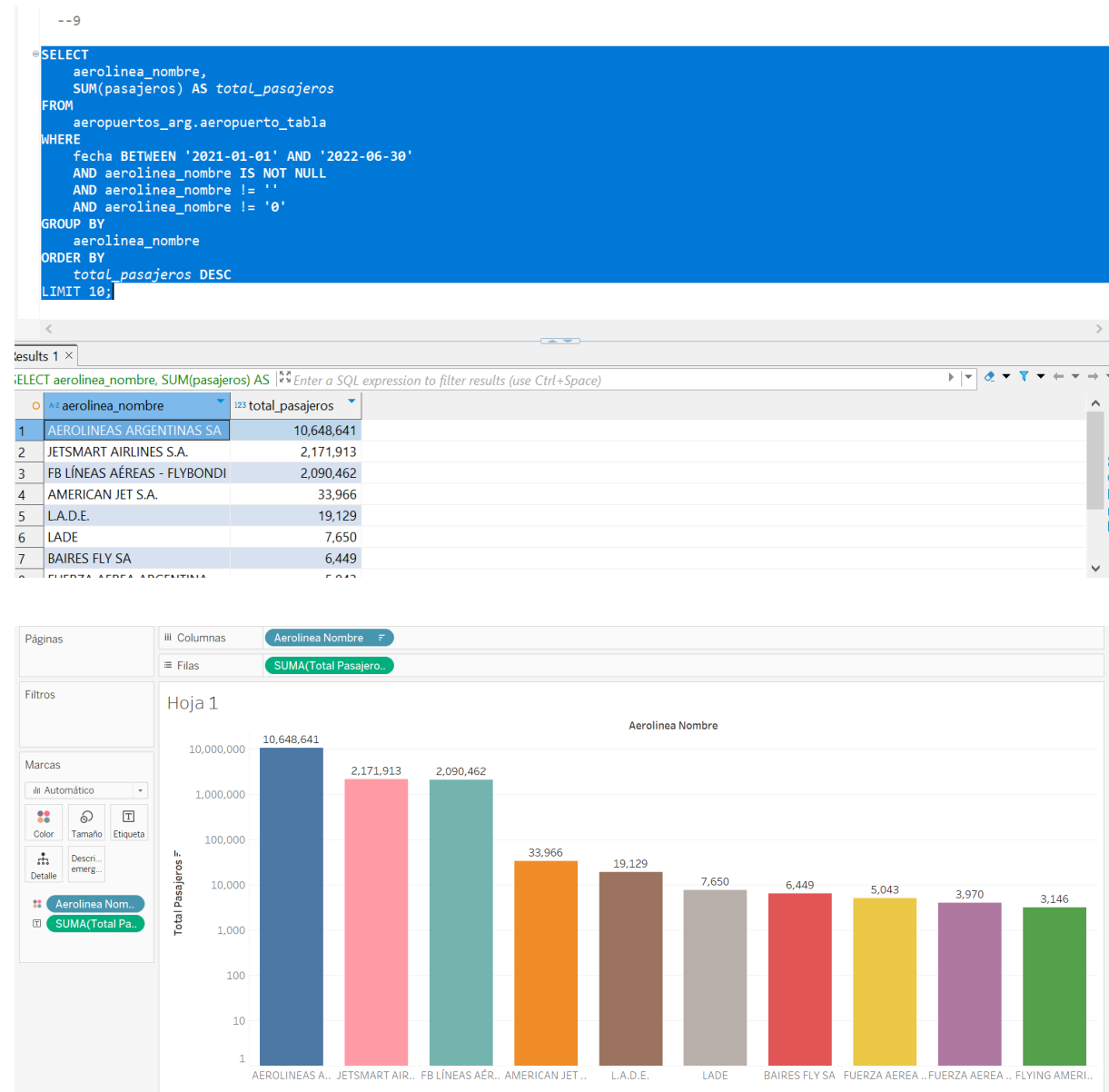
SELECT fecha, horaUTC AS hora, CASE WHEN t Enter a SQL expression to filter results (use Ctrl+Space)

	fecha	hora	codigo_aeropuerto_salida	ciudad_salida	codigo_aeropuerto_arribo	ciudad_arribo	pasajeros
1	2022-06-30	21:26	[NULL]	[NULL]	LAR	AER	[NULL]
2	2022-06-30	01:33	LAR	AER	[NULL]	[NULL]	[NULL]
3	2022-06-30	13:08	[NULL]	[NULL]	VIE	AER	[NULL]
4	2022-06-30	16:26	VIE	BAR	[NULL]	[NULL]	[NULL]
5	2022-06-30	23:59	DOZ	AER	[NULL]	[NULL]	[NULL]
6	2022-06-30	23:59	[NULL]	[NULL]	ECA	AER	[NULL]
7	2022-06-30	23:58	[NULL]	[NULL]	SAL	CBA	[NULL]
8	2022-06-30	23:55	[NULL]	[NULL]	AER	TUC	[NULL]
9	2022-06-30	23:53	[NULL]	[NULL]	CBA	NEU	[NULL]
10	2022-06-30	23:49	DOZ	CBA	[NULL]	[NULL]	0
11	2022-06-30	23:48	AER	CBA	[NULL]	[NULL]	[NULL]

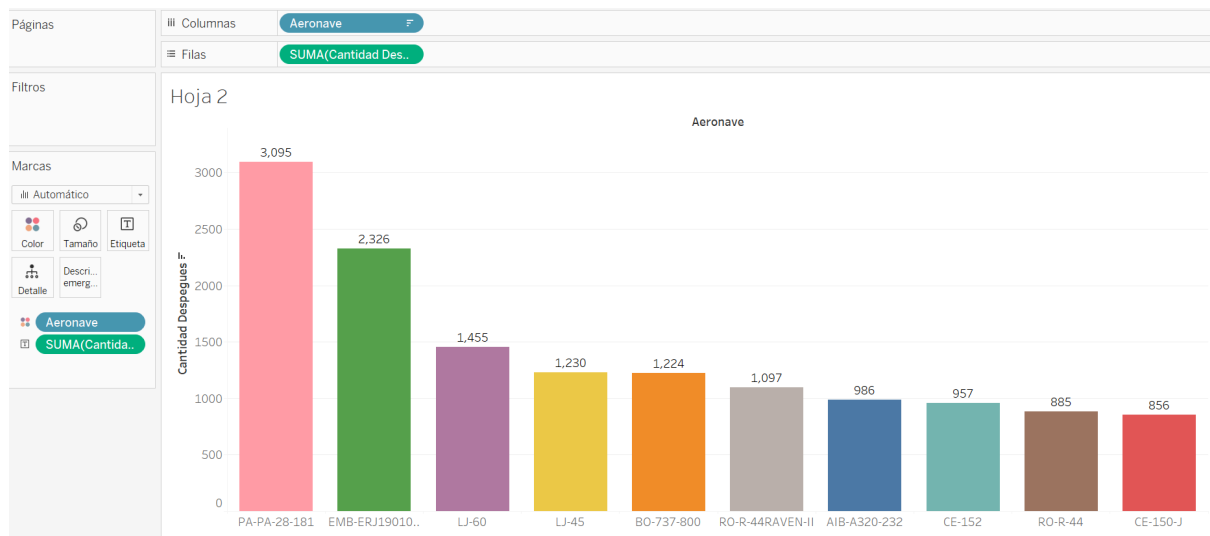
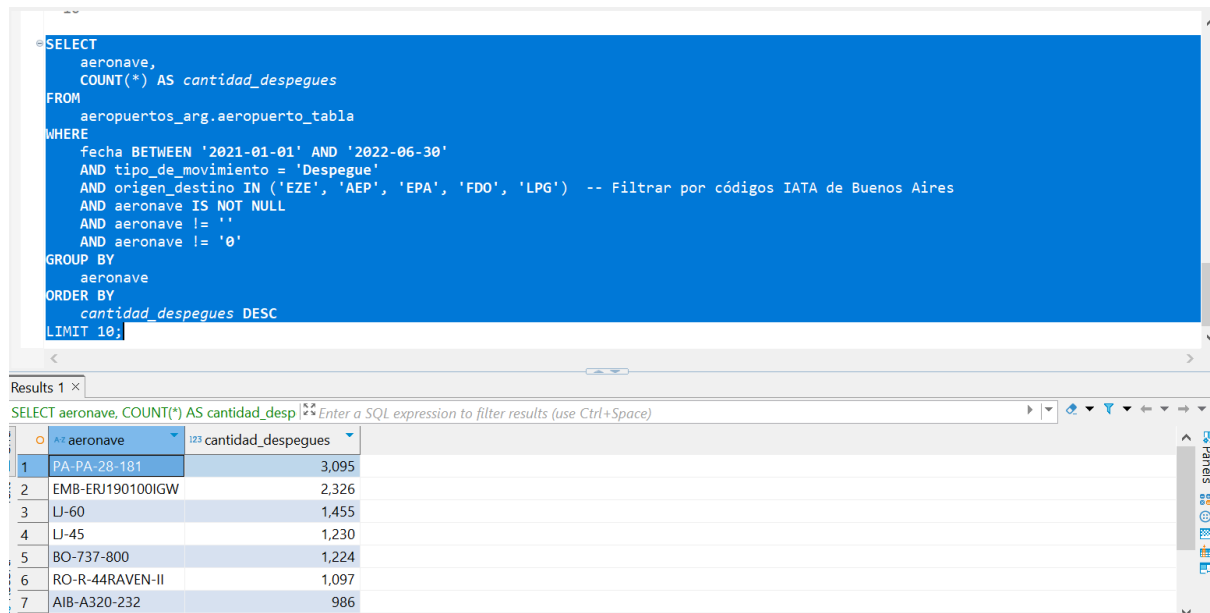
```
--
select fecha, horautc, tipo_de_movimiento, aeropuerto,
origen_destino , pasajeros
from aeropuertos_arg.aeropuerto_tabla
WHERE tipo_de_movimiento = 'Despegue';
```

```
select fecha, horautc, tipo_de_movimiento, aeropuerto,
origen_destino , pasajeros
from aeropuertos_arg.aeropuerto_tabla
WHERE tipo_de_movimiento = 'Aterrizaje';
```

9. Cuales son las 10 aerolíneas que más pasajeros llevaron entre el 01/01/2021 y el 30/06/2022 exceptuando aquellas aerolíneas que no tengan nombre. Mostrar consulta y Visualización



10. Cuales son las 10 aeronaves más utilizadas entre el 01/01/2021 y el 30/06/22 que despegaron desde la Ciudad autónoma de Buenos Aires o de Buenos Aires, exceptuando aquellas aeronaves que no cuentan con nombre. Mostrar consulta y Visualización



11. Qué datos externos agregaría en este dataset que mejoraría el análisis de los datos

- Costos del combustible y consumo promedio por tipo de aeronave.
- Datos sobre emisiones de CO2 por vuelo, lo cual es relevante para análisis de impacto ambiental y sostenibilidad
- Tarifas promedio de los vuelos para entender la demanda y segmentación del mercado.

12. Elabore sus conclusiones y recomendaciones sobre este proyecto.

- La integración de datos de pasajeros, aeronaves y aeropuertos ofrece una visión holística de la operación aérea en la región.
- Se pueden identificar patrones de tráfico y estacionalidad, observando picos de demanda en ciertos periodos.
- Existen oportunidades para optimizar las operaciones de vuelo y reducir los tiempos de espera.

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

Si se desea implementar una arquitectura en la nube para este proceso, se podría utilizar la siguiente estructura con servicios de AWS:

1. **Ingestión de Datos:**
 - **AWS S3:** Almacenamiento de archivos CSV, JSON o Parquet.
 - **AWS Glue:** Ingestión y transformación de datos. Se puede usar para limpiar y preprocesar los datos.
2. **Procesamiento de Datos:**
 - **AWS Glue o AWS EMR (con Spark):** Para el procesamiento distribuido y transformación de grandes volúmenes de datos.
 - **AWS Lambda:** Para procesamiento en tiempo real y funciones automatizadas basadas en eventos.
3. **Almacenamiento y Consulta:**
 - **Amazon Redshift:** Almacenamiento de datos estructurados y análisis SQL.
 - **Amazon Athena:** Consulta interactiva de datos en S3 utilizando SQL sin necesidad de un clúster de bases de datos.
4. **Visualización y Análisis:**
 - **Amazon QuickSight:** Creación de dashboards y reportes interactivos.
 - **Jupyter Notebooks en SageMaker:** Para análisis exploratorio y modelado predictivo.
5. **Automatización y Orquestación:**
 - **AWS Step Functions:** Orquestación de flujos de trabajo.
 - **Amazon CloudWatch:** Monitoreo y alertas.

Ejercicio 2

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

```
hive> show databases;
OK
aeropuertos_arg
car_rental_db
default
f1
northwind_analytics
titanic_data
tripdata
Time taken: 0.024 seconds, Fetched: 7 row(s)
hive> show tables;
OK
car_rental_analytics
Time taken: 0.028 seconds, Fetched: 1 row(s)
hive> █
```

2. Crear script para el ingest de estos dos files

```
hadoop@f769ed737105:~/scripts$ cat ingest_car_rental.sh
#!/bin/bash

# Definir el directorio de descarga en HDFS
download_dir="hdfs://172.17.0.2:9000/ingest"

# Crear el directorio en HDFS si no existe
hadoop fs -test -d "$download_dir"
if [ $? -ne 0 ]; then
    hadoop fs -mkdir -p "$download_dir"
fi

# Definir un directorio temporal local para almacenar los archivos descargados antes de subirlos a HDFS
temp_local_dir="/home/hadoop/landing_temp"
mkdir -p "$temp_local_dir"

# Descargar Car Rental Data
echo "Descargando Car Rental Data..."
curl -o "$temp_local_dir/CarRentalData.csv" https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv

# Descargar georef United States of America State
echo "Descargando georef United States of America State..."
curl -o "$temp_local_dir/georef-united-states-of-america-state.csv" https://dataengineerpublic.blob.core.windows.net/data-engineer/georef-united-states-of-america-state.csv

# Subir los archivos descargados a HDFS
hadoop fs -put "$temp_local_dir/CarRentalData.csv" "$download_dir"
hadoop fs -put "$temp_local_dir/georef-united-states-of-america-state.csv" "$download_dir"

# Eliminar el directorio temporal local
rm -rf "$temp_local_dir"

# Confirmar la subida completa
echo "Todos los archivos han sido descargados y subidos a $download_dir 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

```
hadoop@769ed737105:~/scripts$ cat car_rental_trans.py
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql import functions as F

sc = SparkContext('local')
spark = SparkSession(sc)
from pyspark.sql import HiveContext
hc = HiveContext(sc)

#lectura

georef = spark.read.option("header", True).option("inferSchema", True).option("delimiter", ";").csv(
"hdfs://172.17.0.2:9000/ingest/georef-united-states-of-america-state.csv")

car_rental = spark.read.option("header", True).option("inferSchema", True).option("delimiter", ",").
csv("hdfs://172.17.0.2:9000/ingest/CarRentalData.csv")

#quitar espacios y puntos de los nombres de columna

georef = georef.toDF(*[c.replace(' ', '_').replace('.', '_') for c in georef.columns])
car_rental = car_rental.toDF(*[c.replace(' ', '_').replace('.', '_') for c in car_rental.columns])

#cast de rating a INT

car_rental = car_rental.withColumn("rating", F.round(F.col("rating")).cast("int"))

# join

geo_car = car_rental.join(
    georef,
    car_rental["location_state"] == georef["United_States_Postal_Service_state_abbreviation"],
    how="outer"
)

#eliminar ratings nulos

geo_car = geo_car.dropna(subset=["rating"])
```



```

# Convertir los valores de 'fuelType' a minúsculas
#geo_car = geo_car.withColumn("fuelType", F.lower(F.col("fuelType")))

# Excluir las filas donde 'location_state' o 'United_States_Postal_Service_state_abbreviation' sea 'TX'
geo_car = geo_car.filter(
    (geo_car["location_state"] != "TX") &
    (geo_car["United_States_Postal_Service_state_abbreviation"] != "TX")
)

# drop de columnas que no estaran en hive

columns_to_drop = [
    "location_country",
    "location_latitude",
    "location_longitude",
    "location_state",
    "vehicle_type",
    "Geo_Shape",
    "Year",
    "Official_Code_State",
    "Iso_3166-3_Area_Code",
    "Type",
    "United_States_Postal_Service_state_abbreviation",
    "State_FIPS_Code",
    "State_GNIS_Code",
    "Geo_Point"
]

geo_car = geo_car.drop(*columns_to_drop)

# Renombrar las columnas especificadas
geo_car = geo_car.withColumnRenamed("location_city", "city") \
    .withColumnRenamed("Official_Name_State", "state_name") \
    .withColumnRenamed("vehicle_make", "make") \
    .withColumnRenamed("vehicle_model", "model") \

```

```

        .withColumnRenamed("vehicle_year", "year")

# Ordenar las columnas en el orden especificado
geo_car = geo_car.select(
    "fuelType",
    "rating",
    "renterTripsTaken",
    "reviewCount",
    "city",
    "state_name",
    "owner_id",
    "rate_daily",
    "make",
    "model",
    "year"
)

#cast

geo_car = geo_car.withColumn("fuelType", F.col("fuelType").cast("string")) \
    .withColumn("rating", F.col("rating").cast("int")) \
    .withColumn("renterTripsTaken", F.col("renterTripsTaken").cast("int")) \
    .withColumn("reviewCount", F.col("reviewCount").cast("int")) \
    .withColumn("city", F.col("city").cast("string")) \
    .withColumn("state_name", F.col("state_name").cast("string")) \
    .withColumn("owner_id", F.col("owner_id").cast("int")) \
    .withColumn("rate_daily", F.col("rate_daily").cast("int")) \
    .withColumn("make", F.col("make").cast("string")) \
    .withColumn("model", F.col("model").cast("string")) \
    .withColumn("year", F.col("year").cast("int"))

geo_car.write.insertInto('car_rental_db.car_rental_analytics')

```

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

```

hadoop@f769ed737105:~/airflow/dags$ cat geocar_padre_dag.py
from datetime import timedelta
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.operators.dagrun_operator import TriggerDagRunOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.dates import days_ago

# Definición del DAG padre
geocar_padre_dag = DAG(
    dag_id="geocar_padre_dag",
    default_args={
        "owner": "airflow",
    },
    description="Ingesta y llama a geocar_hijo_dag",
    schedule_interval='0 0 * * *',
    start_date=days_ago(2),
)

# Definición de las tareas
inicia_proceso = DummyOperator(
    task_id='inicia_proceso',
    dag=geocar_padre_dag, # Asociar tarea con el DAG padre
)

ingest_files = BashOperator(
    task_id='ingest_files',
    bash_command='/usr/bin/sh /home/hadoop/scripts/ingest_car_rental.sh ',
    dag=geocar_padre_dag, # Asociar tarea con el DAG padre
)

trigger_geocar_hijo_dag = TriggerDagRunOperator(
    task_id="trigger_geocar_hijo_dag",
    trigger_dag_id="geocar_hijo_dag",
    dag=geocar_padre_dag, # Asociar tarea con el DAG padre
)

finaliza_proceso = DummyOperator(
    task_id='finaliza_proceso',
    dag=geocar_padre_dag, # Asociar tarea con el DAG padre
)

# Definición de la secuencia de tareas
inicia_proceso >> ingest_files >> trigger_geocar_hijo_dag >> finaliza_proceso

```

```

hadoop@f769ed737105:~/airflow/dags$ cat geocar_hijo_dag.py
from datetime import timedelta
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from airflow.operators.dummy import DummyOperator
from airflow.utils.dates import days_ago

# Definición del DAG hijo
geocar_hijo_dag = DAG(
    dag_id="geocar_hijo_dag",
    default_args={
        "owner": "airflow",
    },
    description="Transformación y carga en Hive",
    schedule_interval=None,
    start_date=days_ago(2),
)

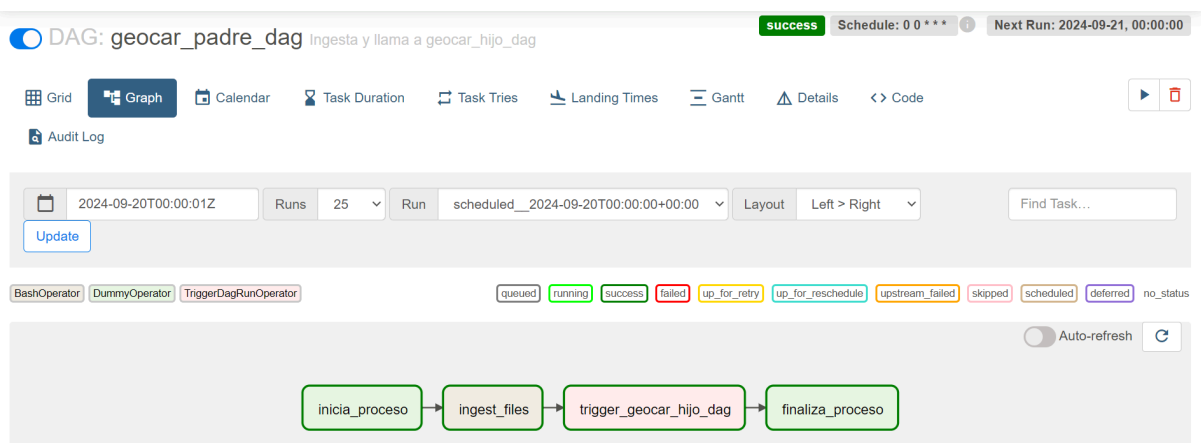
# Definición de las tareas
inicia_proceso = DummyOperator(
    task_id='inicia_proceso',
    dag=geocar_hijo_dag, # Asociar tarea con el DAG hijo
)

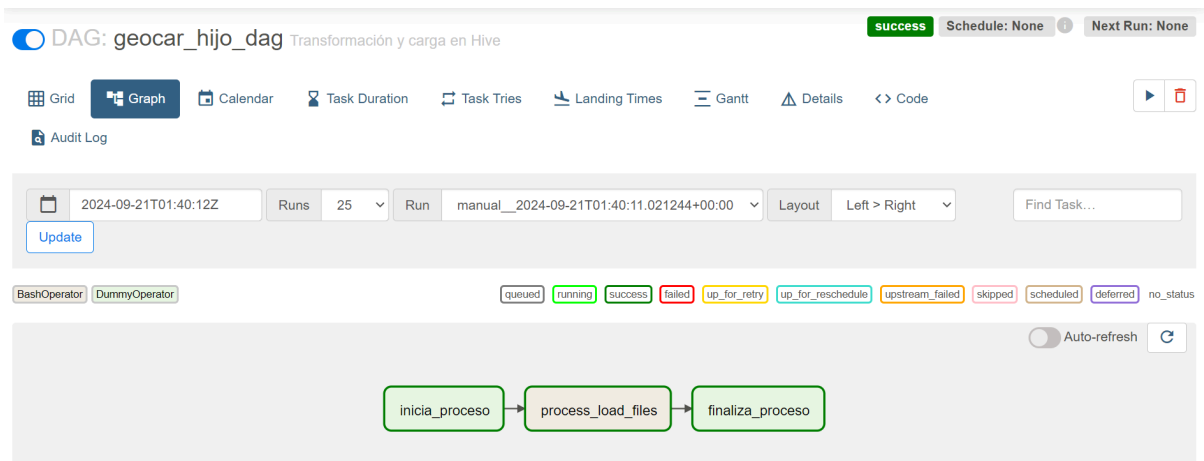
process_load_files = BashOperator(
    task_id='process_load_files',
    bash_command='ssh hadoop@172.17.0.2 /home/hadoop/spark/bin/spark-submit --files /home/hadoop/hive/conf/hive-site.xml /home/hadoop/scripts/car_rental_trans.py',
    dag=geocar_hijo_dag, # Asociar tarea con el DAG hijo
)

finaliza_proceso = DummyOperator(
    task_id='finaliza_proceso',
    dag=geocar_hijo_dag, # Asociar tarea con el DAG hijo
)

# Definición de la secuencia de tareas
inicia_proceso >> process_load_files >> finaliza_proceso

```





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.

```
--a
SELECT COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
WHERE fuelType IN ('HYBRID', 'ELECTRIC')
AND rating >= 4;

--b
SELECT state_name, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
group by state_name
ORDER BY cantidad_alquileres ASC
```

Results 1 ×

SELECT COUNT(*) AS cantidad_alquileres

	cantidad_alquileres
1	458

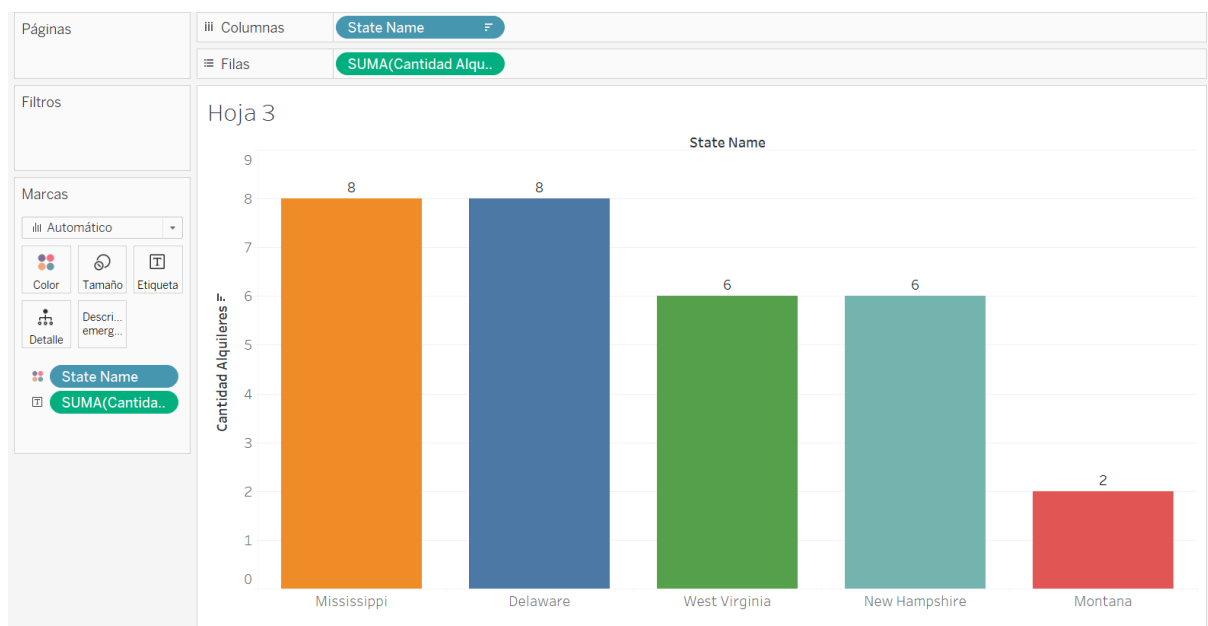
b. los 5 estados con menor cantidad de alquileres (mostrar query y visualización)

```
--b
SELECT state_name, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
group by state_name
ORDER BY cantidad_alquileres ASC
limit 5;
```

```
--c
SELECT make, model, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
group by make, model
ORDER BY cantidad_alquileres DESC ;
```

```
--d
SELECT `year`, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
```

state_name	cantidad_alquileres
Montana	2
West Virginia	6
New Hampshire	6
Delaware	8
Mississippi	8



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

--c

SELECT make, model, COUNT(*) AS cantidad_alquileres

FROM car_rental_db.car_rental_analytics cra

group by make, model

ORDER BY cantidad_alquileres DESC

LIMIT 10;

--d

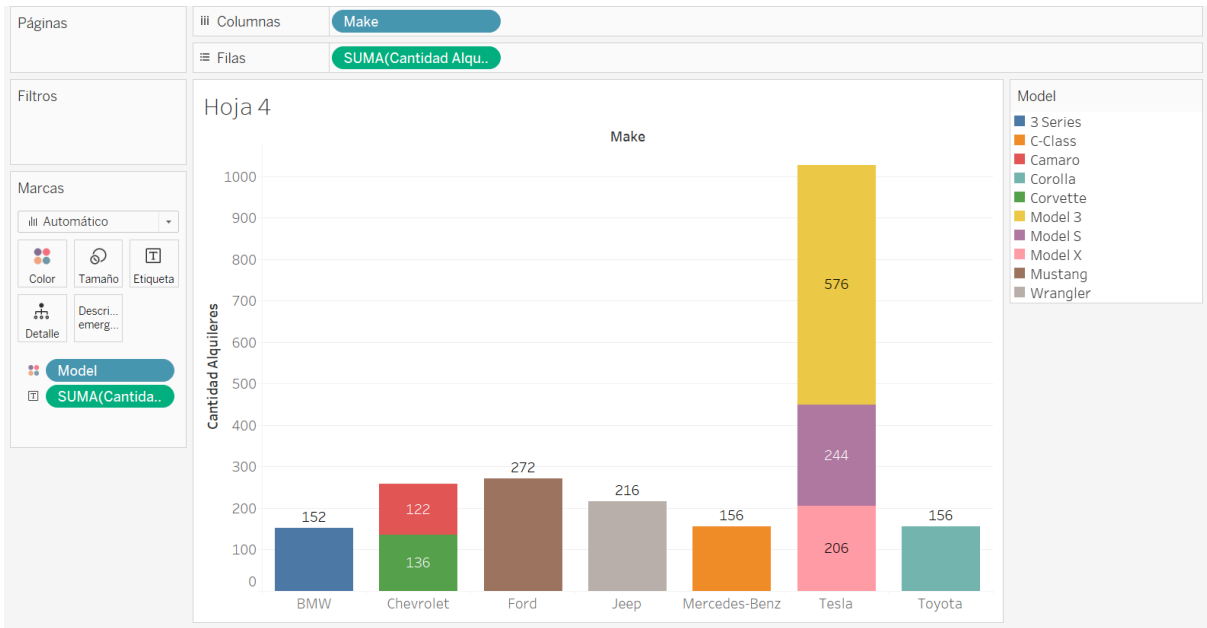
SELECT `year` , COUNT(*) AS cantidad_alquileres

FROM car_rental_db.car_rental_analytics cra

Results 1 x

SELECT make, model, COUNT(*) AS c, Enter a SQL expression to filter results (use Ctrl+Space)

	Az make	Az model	123 cantidad_alquileres
1	Tesla	Model 3	576
2	Ford	Mustang	272
3	Tesla	Model S	244
4	Jeep	Wrangler	216
5	Tesla	Model X	206
6	Toyota	Corolla	156
7	Mercedes-Benz	C-Class	156
8	BMW	3 Series	152
9	Chevrolet	Corvette	136
10	Chevrolet	Camaro	122



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

```
--d
SELECT `year`, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
WHERE `year` BETWEEN 2010 AND 2015
GROUP BY `year`;

--e

SELECT city, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
WHERE fuelType IN ('HYBRID', 'ELECTRIC')
GROUP BY city
ORDER BY cantidad_alquileres DESC
limit 5;
```

results 1 x

SELECT `year`, COUNT(*) AS cantidad_alquileres

	year	cantidad_alquileres
1	2,010	288
2	2,011	400
3	2,012	450
4	2,013	610
5	2,014	764
6	2,015	1,064

e. las 5 ciudades con más alquileres de vehículos ecológicos (fuelType híbrido o eléctrico)

```
--e

SELECT city, COUNT(*) AS cantidad_alquileres
FROM car_rental_db.car_rental_analytics cra
WHERE fuelType IN ('HYBRID', 'ELECTRIC')
GROUP BY city
ORDER BY cantidad_alquileres DESC
limit 5;
```

Results 1 x

SELECT city, COUNT(*) AS cantidad_alquileres

	city	cantidad_alquileres
1	San Diego	88
2	Las Vegas	68
3	Portland	40
4	Phoenix	34
5	San Jose	30

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

```
--f
SELECT fueltype, AVG(reviewcount) as promedio_reviews
FROM car_rental_db.car_rental_analytics cra
group by fueltype;
```

results 1 x

SELECT fueltype, AVG(reviewcount) as *Enter a SQL expression to filter results (use Ctrl+Space)*

	fueltype	promedio_reviews
1	[NULL]	21.0491803279
2	DIESEL	17.5
3	ELECTRIC	28.3394833948
4	GASOLINE	31.9270236613
5	HYBRID	34.8733624454

6. Elabore sus conclusiones y recomendaciones sobre este proyecto.

- La transformación de los datos es una de las mas críticas cuando se crea un pipeline, un carácter inadecuado, un espacio vacío dejado o incluso un cast mal planeado puede llevar a errores que pueden ser difíciles de rastrear si no se presta la debida atención.

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

- Las mismas que en el caso anterior.

Ejercicio 3

Realizar el siguiente LAB, al finalizar pegar un print screen donde se ve su perfil y el progreso final verificado

Transformation Pipeline with Cloud Dataprep

Once your Cloud Dataprep job is completed, refresh your BigQuery page and confirm that the output table **revenue_reporting** exists.

Note: If your job fails, try waiting a minute, pressing the back button on your browser, and running the job again with the same settings.

Click **Check my progress** to verify the objective.

✓

Verify if the Cloud Dataprep jobs output the data to BigQuery

[Check my progress](#)

Assessment Completed!

Congratulations!

The screenshot shows the 'Ecommerce Analytics Pipeline' in Google Cloud Data Studio. The job 'all_sessions_raw_dataprep' has been successfully transformed with a profile. The summary indicates: 80% valid values, 0% mismatching values, and 20% missing values. The 'Publish' step is also completed, creating the 'revenue_reporting' table in BigQuery.

Google Cloud
Dashboard Paths Explore Profile Subscriptions
★ 160 pts ? 🌐

Google Cloud Skills Boost

☰
Paths
≡
Activities
🏆
Leaderboard
🏅
Badges

Course Lab Quiz Game

In progress Finished

Activity	Type	Date started	Date finished	Score	Passed
Creating a Data Transformation Pipeline with Cloud Dataprep	Lab	1 hour ago	0 minutes ago	Assessment: 100%	✓
Introduction to Cloud Dataproc: Hadoop and Spark on Google Cloud	Lab	Sep 5, 2024	Sep 5, 2024	Assessment: 100%	✓
Dataproc: Qwik Start - Command Line	Lab	Sep 4, 2024	Sep 4, 2024	Assessment: 100%	✓
Dataproc: Qwik Start - Console	Lab	Sep 3, 2024	Sep 3, 2024	Assessment: 100%	✓

- Enriquecimiento de datos: Agregar nuevas columnas o datos derivados.
- Visualización de datos: Crear gráficos simples para entender la distribución y calidad de los datos.
- Automatización de flujos de trabajo: Crear y ejecutar flujos de preparación de datos de manera repetitiva.

3. ¿Por qué otra/s herramientas lo podrías reemplazar? ¿Por qué?

DataPrep puede ser reemplazado por herramientas como:

- Apache NiFi: Para flujos de trabajo de preparación de datos más complejos, especialmente con integraciones en tiempo real y control de flujo.
- Alteryx: Ofrece una interfaz intuitiva para preparar y analizar datos sin necesidad de código, similar a DataPrep.
- Databricks o Apache Spark: Para tareas más avanzadas de manipulación y análisis de grandes volúmenes de datos con capacidades de procesamiento distribuido.
- Tableau Prep: Especialmente para usuarios que también usan Tableau para análisis visual.

La elección depende del entorno, los volúmenes de datos, la complejidad de las transformaciones requeridas y la preferencia por herramientas con o sin código.

4. ¿Cuáles son los casos de uso comunes de Data Prep de GCP?

Los casos de uso comunes de Data Prep de GCP incluyen:

- Preparación de datos para Machine Learning: Limpieza y transformación de datos para su uso en modelos de aprendizaje automático.
- ETL (Extract, Transform, Load): Extracción de datos de múltiples fuentes, transformación y carga en almacenes de datos como BigQuery.
- Análisis de datos exploratorio: Análisis rápido y visualización de datos para descubrir patrones o anomalías.
- Creación de pipelines de datos: Automatización de procesos recurrentes de manipulación y limpieza de datos.

5. ¿Cómo se cargan los datos en Data Prep de GCP?

Los datos se pueden cargar en Data Prep de GCP desde diversas fuentes:

- Google Cloud Storage (GCS): Subiendo archivos CSV, JSON, Avro, etc.
- BigQuery: Conectándose directamente a conjuntos de datos y tablas.
- Bases de datos externas: Usando conectores a bases de datos como MySQL o PostgreSQL.
- APIs y otros servicios de GCP: A través de conectores nativos o mediante configuraciones personalizadas.

6. ¿Qué tipos de datos se pueden preparar en Data Prep de GCP?

Se pueden preparar diferentes tipos de datos, incluyendo:

- Datos estructurados como tablas en BigQuery o archivos CSV.
- Datos semi-estructurados como JSON o Avro.
- Datos no estructurados, que pueden necesitar procesos más avanzados de transformación.

7. ¿Qué pasos se pueden seguir para limpiar y transformar datos en Data Prep de GCP?

1. Importar los datos desde las fuentes deseadas.
2. Eliminar duplicados y registros inconsistentes.
3. Manejo de valores nulos o faltantes (imputación, eliminación, etc.).
4. Transformaciones de formato (cambio de tipo de datos, estandarización de fechas, etc.).
5. Crear nuevas columnas con datos calculados o derivados.
6. Filtrado de registros basados en condiciones específicas.
7. Unión de datos de diferentes tablas o archivos.
8. Verificación de la calidad de los datos mediante reglas definidas.

8. ¿Cómo se pueden automatizar tareas de preparación de datos en Data Prep de GCP?

Las tareas de preparación de datos se pueden automatizar mediante:

- Creación de recetas: Secuencias de pasos que se pueden guardar y reutilizar.
- Programación de flujos: Configuración de triggers para ejecutar recetas en horarios específicos o en respuesta a eventos.
- Uso de APIs: Para integrar Data Prep en pipelines de datos más amplios y ejecutar tareas automáticamente.

9. ¿Qué tipos de visualizaciones se pueden crear en Data Prep de GCP?

- Histogramas: Para visualizar la distribución de valores en columnas.
- Gráficos de barras: Para analizar datos categóricos.
- Gráficos de líneas y dispersión: Para detectar tendencias o relaciones entre variables.
- Resumen estadístico: Para mostrar métricas como media, mediana, máximo, mínimo y desviación estándar de los datos.

10. ¿Cómo se puede garantizar la calidad de los datos en Data Prep de GCP?

- Validación de datos: Creación de reglas y condiciones para verificar la consistencia y precisión de los datos.
- Perfilado de datos: Análisis exploratorio para identificar valores atípicos, distribuciones irregulares o problemas de calidad.
- Pruebas automatizadas: Configuración de pruebas para validar la integridad de los datos antes de continuar con los procesos.
- Monitoreo continuo: Supervisión de los pipelines de datos para detectar y resolver problemas en tiempo real.

Arquitectura:

El gerente de Analitica te pide realizar una arquitectura hecha en GCP que contemple el uso de esta herramienta ya que le parece muy fácil de usar y una interfaz visual que ayuda a sus desarrolladores ya que no necesitan conocer ningún lenguaje de desarrollo. (?)

- Transferimos los datos desde S3 a GCS con Cloud Storage Transfer.
- Conectamos DataPrep a Cloud SQL y GCS para preparar los datos.
- Limpiamos y transformamos los datos en DataPrep usando recetas automatizadas.
- Almacenamos los datos procesados en BigQuery para análisis posterior.
- Creamos dashboards y reportes en Data Studio, conectados a BigQuery.
- Entrenamos y evaluamos modelos de ML en BigQuery ML para aprovechar los datos almacenados.