

Ejercicio 1

Punto Número 2

De acuerdo con el enunciado de la tarea, se requería crear dos tablas en el datawarehouse de HIVE. Se creó en HIVE una base de datos llamada “*airport_trips_data*”. En esta base de datos se crearon las siguientes tablas:

- a. *aeropuerto_tabla*
- b. *aeropuerto_detalle_tabla*

A continuación, los screen shots de ambas tablas:

aeropuerto_tabla

Column Name	#	Data Type	Length	Scale	Not Null	Auto Generated	Auto Increment	Default	Description
fecha	1	DATE	10		[]	[]	[]		
hora_aut	2	STRING			[]	[]	[]		
clase_de_vuelo	3	STRING			[]	[]	[]		
clasificacion_de_vuelo	4	STRING			[]	[]	[]		
tipo_de_movimiento	5	STRING			[]	[]	[]		
aeropuerto	6	STRING			[]	[]	[]		
origen_destino	7	STRING			[]	[]	[]		
aerolinea_nombre	8	STRING			[]	[]	[]		
aeronave	9	STRING			[]	[]	[]		
pasajeros	10	INT	10		[]	[]	[]		

aeropuerto_detalle_tabla

Column Name	#	Data Type	Length	Scale	Not Null	Auto Generated	Auto Increment	Default	Description
aeropuerto	1	STRING			[]	[]	[]		
oac	2	STRING			[]	[]	[]		
iata	3	STRING			[]	[]	[]		
tipo	4	STRING			[]	[]	[]		
denominacion	5	STRING			[]	[]	[]		
coordenadas	6	STRING			[]	[]	[]		
latitud	7	STRING			[]	[]	[]		
longitud	8	STRING			[]	[]	[]		
elev	9	FLOAT	7	7	[]	[]	[]		
uom_elev	10	STRING			[]	[]	[]		
ref	11	STRING			[]	[]	[]		
distancia_ref	12	FLOAT	7	7	[]	[]	[]		
direccion_ref	13	STRING			[]	[]	[]		
condicion	14	STRING			[]	[]	[]		
control	15	STRING			[]	[]	[]		
region	16	STRING			[]	[]	[]		
uso	17	STRING			[]	[]	[]		
trafico	18	STRING			[]	[]	[]		
sna	19	STRING			[]	[]	[]		
concesionado	20	STRING			[]	[]	[]		
provincia	21	STRING			[]	[]	[]		

Punto Número 3

El proceso de ingesta se hizo de forma automática por medio de Apache Airflow. Para esto, se creo un DAG (en formato archivo .py). A continuación, un screen shot del DAG

```
1 from datetime import datetime, timedelta
2 from airflow import DAG
3 from airflow.operators.bash import BashOperator
4 from airflow.operators.python import PythonOperator
5 import subprocess
6
7 # Default arguments for the DAG
8 default_args = {
9     'owner': 'airflow',
10     'depends_on_past': False,
11     'start_date': datetime(2025, 6, 20),
12     'email_on_failure': False,
13     'email_on_retry': False,
14     # 'retries': 1,
15     # 'retry_delay': timedelta(minutes=5),
16 }
17
18 # Let's define the DAG
19 dag = DAG(
20     dag_id='exercise_1_ingest_pyspark_dag',
21     default_args=default_args,
22     description='DAG that runs shell script then PySpark job',
23     schedule_interval='@daily',
24     catchup=False,
25     tags=['spark', 'shell', 'ingest', 'pipeline'],
26 )
27
28 def run_shell_script():
29     result = subprocess.run(['/bin/bash', '/home/hadoop/scripts/job1.sh'],
30                             capture_output=True, text=True)
31     print(f"Return code: {result.returncode}")
32     print(f"Output: {result.stdout}")
33     if result.stderr:
34         print(f"Error: {result.stderr}")
35     if result.returncode != 0:
36         raise Exception(f"Script failed with return code {result.returncode}")
37
38 run_script_task = PythonOperator(
39     task_id='run_job1_script',
40     python_callable=run_shell_script,
41     dag=dag,
42 )
43
44 run_pyspark_job = BashOperator(
45     task_id='run_pyspark_job',
46     bash_command='sshpass -p "edvai" ssh hadoop@localhost /home/hadoop/spark/bin/spark-submit --files /home/hadoop/hive/conf/hive-site.xml /home/hadoop/scripts/pyspark_shell.py', # Update this path
```

Este DAG contiene dos procesos

- Un primer proceso corre el archivo .sh que tiene como propósito la ingesta de los archivos y almacenado en HDFS
- El segundo proceso, corre un archivo .py que contiene las instrucciones de PySpark para procesar la información

Para mayor detalle en la carpeta **Tarea_1** está el archivo para su revisión. El archivo se llama ***"exercise_1_dag_final.py"***

A continuación, se presenta el archivo .sh que hace el proceso de ingesta desde la página web y almacena los archivos en el HDFS. El archivo se llama **job1_final.sh**. Para mayor detalle ver los archivos dentro de la carpeta **Tarea_1**

```
# Add Hadoop environment setup at the top of job1.sh
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 # Adjust path as needed
export HADOOP_HOME=/home/hadoop/hadoop # Adjust path as needed
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH

# Variables
FILE_1_URL="https://data-engineer-edvai-public.s3.amazonaws.com/2021-informe-ministerio.csv"
FILE_2_URL="https://data-engineer-edvai-public.s3.amazonaws.com/202206-informe-ministerio.csv"
FILE_3_URL="https://data-engineer-edvai-public.s3.amazonaws.com/aeropuertos_detalle.csv"

# Downloading file number 1 from site
echo "Downloading File Number 1"
wget -O /home/hadoop/landing/informe_1.csv $FILE_1_URL

# Check if first file was downloaded successfully
if [ $? -eq 0 ]; then
    echo "File number 1 successfully downloaded"
else
    echo "Error downloading File number 1"
    exit 1
fi

# Downloading file number 2 from site
echo "Downloading File Number 2"

wget -O /home/hadoop/landing/informe_2.csv $FILE_2_URL

# Check if second file was downloaded successfully
if [ $? -eq 0 ]; then
    echo "File number 2 successfully downloaded"
else
    echo "Error downloading File number 2"
    exit 1
fi

# Downloading file number 2 from site
echo "Downloading File Number 2"

wget -O /home/hadoop/landing/aeropuerto.csv $FILE_3_URL

# Check if third file was downloaded successfully
if [ $? -eq 0 ]; then
```

```

    echo "Error downloading File number 2"
    exit 1
fi

# Downloading file number 2 from site
echo "Downloading File Number 2"

wget -O /home/hadoop/landing/aeropuerto.csv $FILE_3_URL

# Check if third file was downloaded successfully
if [ $? -eq 0 ]; then
    echo "File number 3 successfully downloaded"
else
    echo "Error downloading File number 3"
    exit 1
fi

echo "Moving files to ingest directory"

echo "Sending files to HDFS..."

hdfs dfs -put /home/hadoop/landing/aeropuerto.csv /ingest
hdfs dfs -put /home/hadoop/landing/informe_1.csv /ingest
hdfs dfs -put /home/hadoop/landing/informe_2.csv /ingest

if [ $? -eq 0 ]; then
    echo "Files successfully moved to HDFS!!"
else
    echo "Error moving files to HDFS"
    exit 1
fi

```

Punto Número 4

Las instrucciones de transformación de la data se encuentran dentro del archivo *pyspark_shell_fina.py*. A continuación, un screen shot del archivo. En este script se trabajó lo requerido en el punto 4. Para mayor detalle en la carpeta **Tarea_1** está el archivo para su revisión

```

1 from pyspark.sql import SparkSession
2
3 from pyspark.sql.functions import to_date
4 import sys
5
6 def main():
7
8     # This works
9     spark = SparkSession.builder.master("spark://localhost:7077").appName("HiveLocalConnection").config("hive.metastore.uris", "thrift://localhost:9083").enableHiveSupport().getOrCreate()
10
11     # spark = SparkSession.builder.master("spark://localhost:7077").getOrCreate()
12     try:
13         print("Starting the Spark session...")
14         # PARTE CERO - AEROPUERTOS DETALLES
15         print("Processing aeropuerto.csv...")
16         df = spark.read.option("header", "true").csv("hdfs://172.17.0.2:9000/ingest/aeropuerto.csv", sep=',')
17         cols_drop = ['fin', 'inial']
18         df_drop = df.drop(*cols_drop)
19         df_select = df.select.withColumn('distancia_ref', df.select['distancia_ref'].cast('float'))
20         df_select = df.select.withColumn('elev', df_select['elev'].cast('float'))
21         df_select = df.select.na.fill(0, subset=['distancia_ref'])
22         df_select = df.select.na.fill(0, subset=['elev'])
23         new_col_names = ['aeropuerto', 'loc', 'lata', 'tipo', 'denominacion', 'coordenadas', 'latitud', 'longitud', 'elev', 'uoe_elev', 'ref', 'distancia_ref', 'direccion_ref', 'condicion', 'control', 'region', 'uso', 'trafico', 'sna', 'concesionado', 'prov']
24         df_select = df.select.toDF(*new_col_names)
25         df_select.write.mode('append').insertInto('airport_trips_data.aeropuerto_detalle_tabla')
26
27         print("File aeropuerto.csv processed successfully.")
28
29         # PARTE 1 - INFORME 1
30         print("Processing informe_1.csv...")
31         df2 = spark.read.option("header", "true").csv("hdfs://172.17.0.2:9000/ingest/informe_1.csv", sep=',')
32         cols_drop = ['calidad dato']
33         df2_drop = df2.drop(*cols_drop)
34         df2_select = df2_select.withColumn('fecha', to_date(df2_select['fecha'], 'mm/dd/yyyy'))
35         df2_select = df2_select.withColumn('Pasajeros', df2_select['Pasajeros'].cast('int'))
36         df2_select = df2_select.na.fill(0, subset=['Pasajeros'])
37         new_col_names = ['fecha', 'horaUTC', 'clase de vuelo', 'clasificacion de vuelo', 'tipo de movimiento', 'aeropuerto', 'origen_destino', 'aerolinea_nombre', 'aeronave', 'pasajeros']
38         df2_select = df2_select.toDF(*new_col_names)
39         df2_select = df2_select.filter(df2_select['clasificacion de vuelo'].isin(['Domestico', 'Domestico']))
40         df2_select.write.mode('append').insertInto('airport_trips_data.aeropuerto_detalle_tabla')
41
42         print("File informe_1.csv processed successfully.")
43
44         # PARTE 2 - INFORME 2
45         print("Processing informe_2.csv...")
46         df3 = spark.read.option("header", "true").csv("hdfs://172.17.0.2:9000/ingest/informe_2.csv", sep=',')
47         cols_drop = ['calidad dato']
48         df3_drop = df3.drop(*cols_drop)

```

Punto Número 5

A continuación, screen shots de la creación de la base de datos y la descripción de las tablas.

Creación de la base de datos airport_trips_data

```

tripdata
tripsdb
Time taken: 0.412 seconds, Fetched: 3 row(s)
hive> create database airport_trips_data;
OK
Time taken: 0.575 seconds
hive> show databases;
OK
airport_trips_data
default
tripdata
tripsdb
Time taken: 0.027 seconds, Fetched: 4 row(s)
hive>

```

Creación de la tabla aeropuerto_tabla

```

OK
Time taken: 0.055 seconds
hive> create table if not exists aeropuerto_tabla (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);
OK
Time taken: 0.361 seconds
hive> show tables;
OK
aeropuerto_tabla
Time taken: 0.04 seconds, Fetched: 1 row(s)
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
Time taken: 0.066 seconds, Fetched: 10 row(s)
hive> |

```

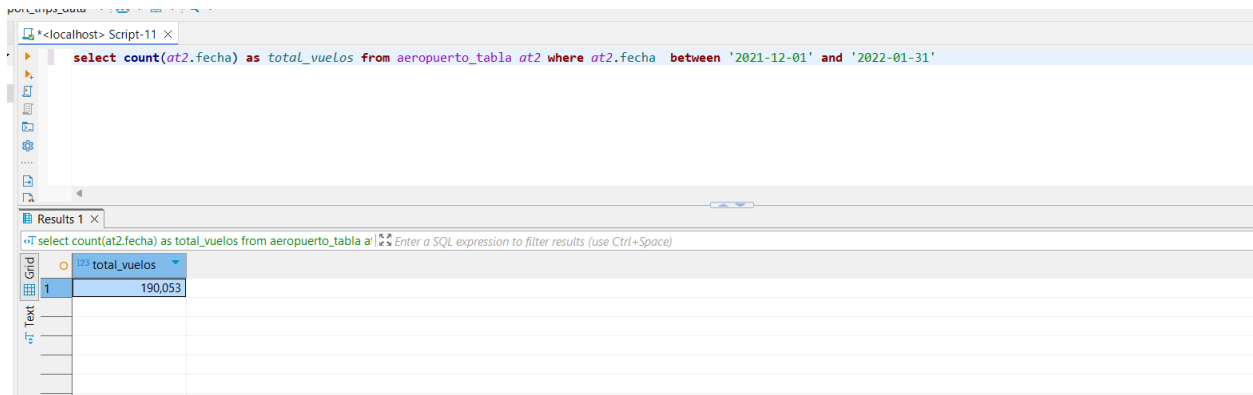
Creación de la tabla aeropuerto_detalle_tabla;

```
pasajeros          int
Time taken: 0.066 seconds, Fetched: 10 row(s)
hive> create table if not exists aeropuerto_detalle_tabla (aeropuerto STRING, oac STRING, Display all 574 possibilities
? (y or n)
hive> create table if not exists aeropuerto_detalle_tabla (aeropuerto STRING, oac STRING, iata STRING, tipo STRING, denom
inacion 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 Displa
y all 574 possibilities? (y or n)
hive> create table if not exists aeropuerto_detalle_tabla (aeropuerto STRING, oac STRING, iata STRING, tipo STRING, denom
inacion 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, Display all 574 possibilities? (y or n)
hive> create table if not exists aeropuerto_detalle_tabla (aeropuerto STRING, oac STRING, iata STRING, tipo STRING, denom
inacion 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);
OK
Time taken: 0.138 seconds
hive> show tables;
OK
aeropuerto_detalle_tabla
```

```
hive
hive> show tables;
OK
aeropuerto_detalle_tabla
aeropuerto_tabla
Time taken: 0.039 seconds, Fetched: 2 row(s)
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
Time taken: 0.071 seconds, Fetched: 21 row(s)
hive> |
hive> create table if not exists aeropuerto_detalle_tabla (aeropuerto STRING, oac STRING, Display all
574 possibilities? (y or n)
```

Punto Número 6

A continuación, screen shots de los queries requeridos en el trabajo 1. Para mayor detalle dentro de la carpeta Tarea_1 se encontrarán los archivos de soporte empleados. ***El query fue modificado de acuerdo a lo requerido***

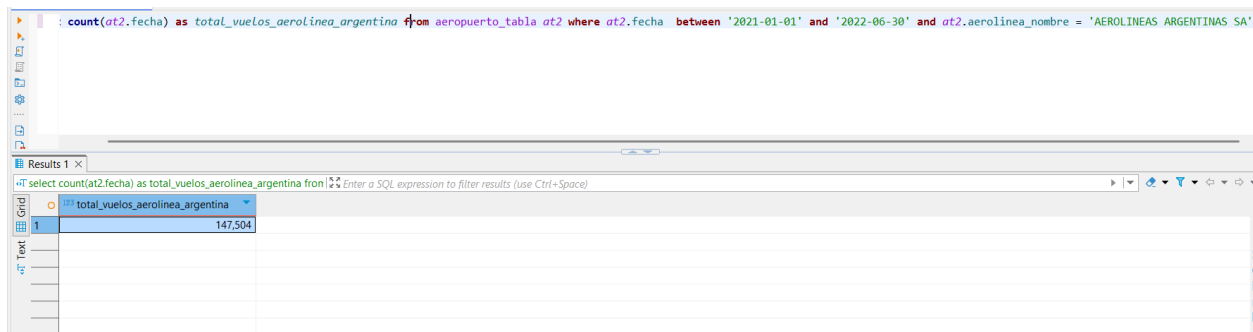


The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query in a script editor: `select count(at2.fecha) as total_vuelos from aeropuerto_tabla at2 where at2.fecha between '2021-12-01' and '2022-01-31'`. The bottom pane shows the results of the query in a grid view. The grid has one column labeled 'total_vuelos' and one row with the value 190,053.

	total_vuelos
1	190,053

Punto Número 7

A continuación, screen shots de los queries requeridos en el trabajo 1. Para mayor detalle dentro de la carpeta Tarea_1 se encontrarán los archivos de soporte empleados. ***El query fue modificado de acuerdo a lo requerido***



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query in a script editor: `select count(at2.fecha) as total_vuelos_aerolinea_argentina from aeropuerto_tabla at2 where at2.fecha between '2021-01-01' and '2022-06-30' and at2.aerolinea_nombre = 'AEROLINEAS ARGENTINAS SA'`. The bottom pane shows the results of the query in a grid view. The grid has one column labeled 'total_vuelos_aerolinea_argentina' and one row with the value 147,504.

	total_vuelos_aerolinea_argentina
1	147,504

Punto Número 8

A continuación, screen shots de los queries requeridos en el trabajo 1. Para mayor detalle dentro de la carpeta Tarea_1 se encontrarán los archivos de soporte empleados. El ordenamiento se hizo en base a la fecha de forma descendiente.

--query numero 3

```
select at2.fecha,at2.hora,at2.aeropuerto as aeropuerto_salida,q1.provincia as ciudad_salida,at2.pasajeros,at2.origen_destino as aeropuerto_arriba,q2.provincia as ciudad_arriba
join (select at2.aeropuerto,adt.provincia from aeropuerto_tabla at2
join aeropuerto_detalle_tabla adt on at2.aeropuerto = adt.aeropuerto
group by at2.aeropuerto,adt.provincia) q1
on at2.aeropuerto = q1.aeropuerto
join (select at2.origen_destino ,adt.provincia from aeropuerto_tabla at2
join aeropuerto_detalle_tabla adt on at2.origen_destino = adt.aeropuerto
group by at2.origen_destino ,adt.provincia) q2
on at2.origen_destino = q2.origen_destino
where at2.fecha between '2021-01-01' and '2022-06-30' and at2.tipo_de_movimiento = 'Despegue' and at2.pasajeros>0
order by at2.fecha desc
```

Results 1

select at2.fecha,at2.hora,at2.aeropuerto as aeropuerto_salida, Enter a SQL expression to filter results (use Ctrl+Space)

	fecha	hora	aeropuerto_salida	ciudad_salida	pasajeros	aeropuerto_arriba	ciudad_arriba
1	2022-01-06	07:08	DOZ	MENDOZA	69	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES
2	2022-01-06	07:37	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	83	USU	TIERRA DEL FUEGO ANTÁRTIDA E ISLAS I
3	2022-01-06	08:16	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	80	USU	TIERRA DEL FUEGO ANTÁRTIDA E ISLAS I
4	2022-01-06	08:28	EZE	BUENOS AIRES	76	IGU	MISIONES
5	2022-01-06	08:32	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	89	JUJ	JUJUY
6	2022-01-06	08:33	EZE	BUENOS AIRES	94	SAL	SALTA
7	2022-01-06	08:36	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	60	DOZ	MENDOZA
8	2022-01-06	08:46	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	85	BAR	RÍO NEGRO
9	2022-01-06	09:11	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	77	NEU	NEUQUÉN
10	2022-01-06	09:22	AER	CIUDAD AUTÓNOMA DE BUENOS AIRES	60	NEU	NEUQUÉN
11	2022-01-06	10:10	EZE	BUENOS AIRES	95	BAR	RÍO NEGRO

Punto Número 9

A continuación, screen shots de los queries requeridos en el trabajo 1. Para mayor detalle dentro de la carpeta Tarea_1 se encontrarán los archivos de soporte empleados.

--query numero 4

```
select at2.aerolinea_nombre,sum(at2.pasajeros) as numero_pasejeros from aeropuerto_tabla at2 where at2.aerolinea_nombre is not NULL and at2.fecha between '2021-01-01' and '2022-06-30'
group by at2.aerolinea_nombre order by numero_pasejeros desc LIMIT 10
```

Results 1

select at2.aerolinea_nombre,sum(at2.pasajeros) as numero_pasejeros, Enter a SQL expression to filter results (use Ctrl+Space)

	aerolinea_nombre	numero_pasejeros
1	AEROLINEAS ARGENTINAS SA	7,484,860
2	JETSMART AIRLINES SA.	1,511,650
3	FB LINEAS AÉREAS - FLYBONDI	1,482,473
4	0	92,592
5	AMERICAN JET SA.	25,789
6	LA D.E.	15,074
7	BAIRES FLY SA	4,960
8	LADE	3,895
9	FUERZA AEREA ARGENTINA	3,855
10	FUERZA AEREA ARGENTINA (FAA)	3,138

Refresh Save Cancel Export data 200 10 10 row(s) fetched - 4.481s (0.014s fetch), on 2025-06-21 at 20:55:57

Punto Número 10

A continuación, screen shots de los queries requeridos en el trabajo 1. Para mayor detalle dentro de la carpeta Tarea_1 se encontrarán los archivos de soporte empleados. **Se modifico el query de acuerdo a los requerimientos.**

The screenshot shows a SQL query in a script editor and its results in a table. The query is as follows:

```
select at2.aeronave, count(at2.aeronave) as numero_vuelos from aeropuerto_tabla at2
join aeropuerto_detalle_tabla adt on at2.aeropuerto = adt.aeropuerto
where at2.fecha between '2021-01-01' and '2022-06-30' and adt.provincia in ('BUENOS AIRES','CIUDAD AUTÓNOMA DE BUENOS AIRES') and at2.tipo_de_movimiento = 'Despegue'
GROUP BY at2.aeronave order by numero_vuelos desc LIMIT 10
```

The results table shows the top 10 aircraft by flight count:

aeronave	numero_vuelos
EMB-ERJ190100IGW	12,470
CE-150-L	8,117
CE-152	7,980
0	6,729
CE-150-M	6,081
AIB-A320-232	5,345
BO-737-800	4,537
CE-150-J	3,012
CE-150-G	2,871
BO-B737-800	2,736

Punto Número 11

Personalmente considero que el dataset está bueno para el propósito que se quiere. Yo agregaría los siguientes datos:

- Número de vuelo.** La tabla ya contiene los datos de despegue y aterrizaje por nombre de aerolínea, fecha de salida y arribo, número de pasajeros y aeronave, pero es complicado determinar si son el mismo vuelo. Esto ayudaría a mejorar un poco los analíticos que se quieren extraer.
- Otro dato que se pudiera colocar es el **tiempo teórico del vuelo**, por ejemplo, si fuera aerolíneas argentinas podría determinar razones de eficiencia. Tiempo real vs tiempo teórico.
- Otro dato pudiera ser **horas de vuelo de la aeronave y rutinas de mantenimiento preventivo y/o correctivo**.
- Finalmente, alguna meta data, como **comentarios** de los cuales podemos extraer ciertas palabras para clasificar vuelos o determinar patrones

Punto Número 12

De acuerdo con la información extraída de los queries:

- Desde inicios de diciembre 2021 hasta finales de enero 2022, ha habido una cantidad total de un poco más de 190K vuelos.
- Por otro lado, desde la fecha de enero 2021 hasta finales de junio 2022. Aerolíneas Argentinas ha tenido 147,504 vuelos. Los rangos de tiempo no son comparables respecto al resultado presentado en el punto anterior.
- La aerolínea Aerolíneas Argentinas en el mismo periodo movilizó la mayor cantidad de pasajeros, con más de 7 millones de pasajeros.
- El mismo periodo el avión marca modelo Embraer generó mayor cantidad de vuelos, con 12,470 desde la provincia de Buenos Aires. Basado en esta información podemos inferir que hay una mayor cantidad de vuelos de otras aeronaves partiendo de provincias o ciudades distintas a Buenos Aires.

Punto Número 13

Yo considero que el dinamismo de la data que se extrajo para este trabajo es recomendable pensar en una arquitectura en la nube. La data tiene un poco más de 500K registros, eso fue en dos archivos en casi dos años de data. Dada la cantidad de data, mejorar considerar una arquitectura en la nube de la siguiente forma:

- Google Cloud Storage para almacenar la data

- b. Google Dataproc
- c. Google BigQuery

Todo orquestrado por Google Cloud Composer. Respecto a una herramienta para manejar calidad de data conectaría a Google Big Query una herramienta como Great Expectations (GX Cloud)

Ejercicio 2

Punto 1

De acuerdo con lo requerido en esta tarea se ha creado una base de datos en HIVE llamada **car_rental_db**

```
WARNING: All illegal access operations will be denied in a future release
hive> show databases;
OK
airport_trips_data
car_rental_db
default
tripdata
Time taken: 0.997 seconds, Fetched: 4 row(s)
hive>
```

Y se ha creado la tabla **car_rental_analytics**

```
FAILED: SemanticException [Error 10072]: Database does not exist: car_rental_db
hive> use car_rental_db;
OK
Time taken: 0.055 seconds
hive> show tables;
OK
car_rental_analytics
Time taken: 0.14 seconds, Fetched: 1 row(s)
hive>
```

A continuación, se presenta la descripción de la tabla, campos y tipos de campos

```
hive> describe car_rental_analytics;
OK
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
Time taken: 0.056 seconds, Fetched: 11 row(s)
hive>
```

Punto 2

Se anexa una imagen de pantalla del archivo .sh que se empleó para extraer los archivos. El archivo se llama **job2_final.sh**. Para mayor detalle ver el contenido de la carpeta **Tarea_2**.

```
>_ job2_final.sh X
Tarea_2 >>_ job2_final.sh
1  #!/bin/bash
2
3  # Add Hadoop environment setup at the top of job1.sh
4  export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64 # Adjust path as needed
5  export HADOOP_HOME=/home/hadoop/hadoop # Adjust path as needed
6  export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
7
8  # Variables
9  FILE_1_URL="https://data-engineer-edvai-public.s3.amazonaws.com/CarRentalData.csv"
10 FILE_2_URL="https://data-engineer-edvai-public.s3.amazonaws.com/georef-united-states-of-america-state.csv"
11
12 # Downloading file number 1 from site
13 echo "Downloading File Number 1"
14 wget -O /home/hadoop/landing/cars_data.csv $FILE_1_URL
15
16 # Check if first file was downloaded successfully
17 if [ $? -eq 0 ]; then
18     echo "File number 1 successfully downloaded"
19 else
20     echo "Error downloading File number 1"
21     exit 1
22 fi
23
24 # Downloading file number 2 from site
25 echo "Downloading File Number 2"
26
27 wget -O /home/hadoop/landing/geo_data.csv $FILE_2_URL
28
29 # Check if second file was downloaded successfully
30 if [ $? -eq 0 ]; then
31     echo "File number 2 successfully downloaded"
32 else
33     echo "Error downloading File number 2"
34     exit 1
35 fi
36
37 echo "Moving files to ingest directory"
38
39
40
41 echo "Sending files to HDFS..."
42
43 hdfs dfs -put /home/hadoop/landing/cars_data.csv /ingest
44 hdfs dfs -put /home/hadoop/landing/geo_data.csv /ingest
45
46
47 if [ $? -eq 0 ]; then
48     echo "Files successfully moved to HDFS!!"
49 else
50     echo "Error moving files to HDFS"
51     exit 1
52 fi
53
```

Punto 3

A continuación, se presenta una imagen de pantalla del código en **pyspark** realizado para poder transformar la data de acuerdo con los requerimientos de la tarea. El archivo se llama **exercise_2_pyspark_shell_final.py**. Para mayor detalle ver el contenido de la carpeta **Tarea_2**

```

1 area 2 / exercises_2/pyspark-shell final.py / ~
2 from pyspark.sql import SparkSession
3 from pyspark.sql.functions import round
4 from pyspark.sql.functions import lower, col
5 import sys
6
7 def main():
8     spark = SparkSession.builder.master("spark://localhost:7077").appName("HiveLocalConnection").config("hive.metastore.uris", "thrift://localhost:9083").enableHiveSupport().getOrCreate()
9     # spark = SparkSession.builder.master("spark://localhost:7077").getOrCreate()
10    try:
11        print("Starting the Spark session...")
12        print("Processing cars data csv and geo data csv files...")
13
14        df = spark.read.option("header", "true").csv("hdfs://172.17.0.2:9000/ingest/cars_data.csv", sep=',')
15        df_geo = spark.read.option("header", "true").csv("hdfs://172.17.0.2:9000/ingest/geo_data.csv", sep=',')
16
17        print("File cars_data.csv and geo_data.csv read successfully.")
18
19        print("Processing data...")
20
21        cols_to_drop = ['location.latitude', 'location.country', 'location.longitude', 'vehicle.type']
22        df_select = df.drop(*cols_to_drop)
23        df_geo_select = df_geo.select("United States Postal Service state abbreviation", "Official Name State")
24        df_geo_select = df_geo_select.withColumnRenamed("United States Postal Service state abbreviation", "state_code").withColumnRenamed("Official Name State", "state_name")
25
26        new_col_names = ['fueltype', 'rating', 'rentertripstaken', 'reviewcount', 'city', 'state_code', 'owner_id', 'rate_daily', 'make', 'model', 'year']
27        df_select = df_select.toDF(*new_col_names)
28
29        print("Data processing completed. Merging dataframes...")
30        merged_df = df_select.join(df_geo_select, "state_code", "left")
31        merged_df = merged_df.filter(merged_df.state_name != "Texas")
32        merged_df = merged_df.withColumn("fueltype", lower(col("fueltype")))
33        merged_df = merged_df.na.drop(subset=['rating'])
34        merged_df = merged_df.withColumn("rating", merged_df['rating'].cast('float'))
35        merged_df = merged_df.withColumn("rentertripstaken", merged_df['rentertripstaken'].cast('int'))
36        merged_df = merged_df.withColumn("owner_id", merged_df['owner_id'].cast('int'))
37        merged_df = merged_df.withColumn("rate_daily", merged_df['rate_daily'].cast('int'))
38        merged_df = merged_df.withColumn("reviewcount", merged_df['reviewcount'].cast('int'))
39        merged_df = merged_df.withColumn("year", merged_df['year'].cast('int'))
40        merged_df = merged_df.withColumn("rating", round(merged_df['rating'], 0).cast('int'))
41        col_to_drop = ['state_code']
42        merged_df = merged_df.drop(*col_to_drop)
43        column_order = ['fueltype', 'rating', 'rentertripstaken', 'reviewcount', 'city', 'state_name', 'owner_id', 'rate_daily', 'make', 'model', 'year']
44        merged_df = merged_df.select(*column_order)
45        print("Dataframes merged successfully.")
46
47        print("Writing the merged dataframe to the car_rental_analytics table...")
48
49        merged_df.write.mode('append').insertInto('car_rental_db.car_rental_analytics')
50
51        print("Data written successfully to the car_rental_analytics table.")
52
53    except Exception as e:
54        print(f"An error occurred: {e}", str(e.args))
55        sys.exit(1)

```

Punto 4

Se anexa una imagen del DAG que corre el proceso de ingesta y corre el llamado al DAG hijo. Para mayor detalle ver los contenidos de la carpeta **Tarea_2**. El DAG principal se encuentra en el archivo **“dag_exercise2.py”**, mientras el DAG hijo se encuentra en el archivo **“dag-hijo-final.py”**

Tarea_2 > dag_exercise2.py > ...

```
1  from datetime import datetime, timedelta
2  from airflow import DAG
3  from airflow.operators.bash import BashOperator
4  from airflow.operators.python import PythonOperator
5  from airflow.operators.trigger_dagrun import TriggerDagRunOperator
6  import subprocess
7
8  # Default arguments for the DAG
9  default_args = {
10     'owner': 'airflow',
11     'depends_on_past': False,
12     'start_date': datetime(2025, 6, 20),
13     'email_on_failure': False,
14     'email_on_retry': False,
15     'retries': 1,
16     'retry_delay': timedelta(minutes=1),
17 }
18
19 # Let's define the DAG
20 dag_ingest = DAG(
21     dag_id='exercise2-dag-edvai',
22     default_args=default_args,
23     description='DAG that runs shell script then trigger another DAG',
24     schedule_interval='@daily',
25     catchup=False,
26     tags=['spark', 'shell', 'trigger', 'son_dagcat'],
27 )
28
29 def run_shell_script():
30     result = subprocess.run(['/bin/bash', '/home/hadoop/scripts/job2.sh'],
31                             capture_output=True, text=True)
32     print(f"Return code: {result.returncode}")
33     print(f"Output: {result.stdout}")
34     if result.stderr:
35         print(f"Error: {result.stderr}")
36     if result.returncode != 0:
37         raise Exception(f"Script failed with return code {result.returncode}")
38
39 run_script_task = PythonOperator(
40     task_id='run_job2_script',
41     python_callable=run_shell_script,
```

Punto Número 5.a

Script-13

```
select count(cra.fueltype) as total_rent_number from car_rental_analytics cra
where cra.fueltype in ('electric','hybrid') and cra.rate_daily >=4
```

Results 1

select count(cra.fueltype) as total_rent_number from car_rental_analytics cra

	total_rent_number
1	771

Punto Número 5.b

--query 2

```
select cra.state_name,count(cra.state_name) as rent_quantity from car_rental_analytics cra
group by cra.state_name order by rent_quantity asc limit 5
```

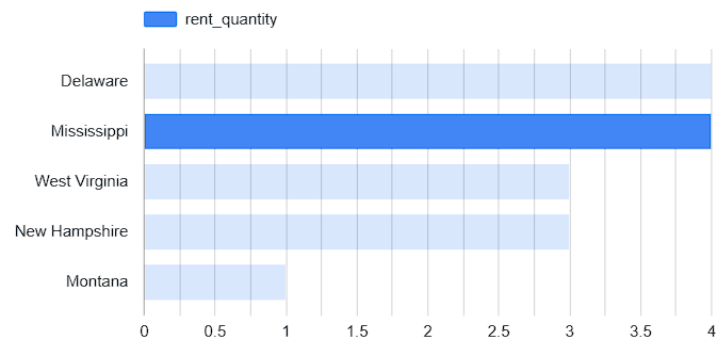
Results 1

Results 2

select cra.state_name,count(cra.state_name) as rent_quantity from car_rental_analytics cra

	state_name	rent_quantity
1	Montana	1
2	West Virginia	3
3	New Hampshire	3
4	Delaware	4
5	Mississippi	4

5 row(s) fetched - 3.382s (0.013s fetch), on 2025-06-22 at 19:08:15



Punto Número 5.c

SQL query 3

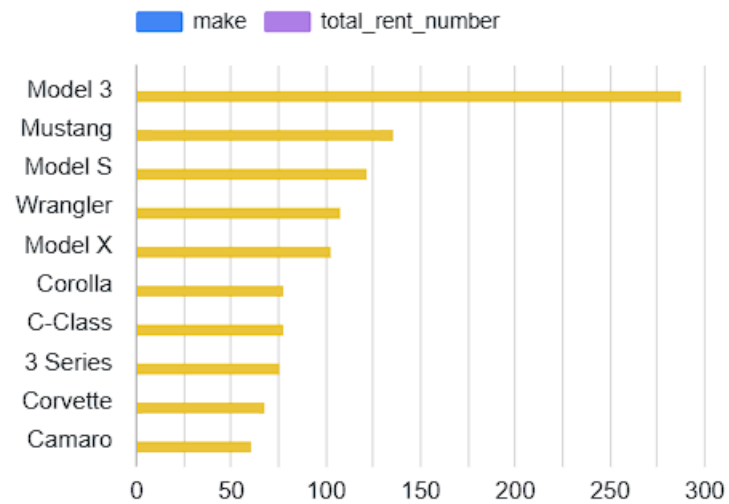
```
select cra.model,cra.make,count(cra.model) as total_rent_number from car_rental_analytics cra
group by cra.model,cra.make order by total_rent_number desc limit 10
```

Results 1 Results 2 Results 3

select cra.model,cra.make,count(cra.model) as total_rent_number from car_rental_analytics cra group by cra.model,cra.make order by total_rent_number desc limit 10

	model	make	total_rent_number
1	Model 3	Tesla	288
2	Mustang	Ford	136
3	Model S	Tesla	122
4	Wrangler	Jeep	108
5	Model X	Tesla	103
6	Corolla	Toyota	78
7	C-Class	Mercedes-Benz	78
8	3 Series	BMW	76
9	Corvette	Chevrolet	68
10	Camaro	Chevrolet	61

Refresh Save Cancel Export data 200 10 10 row(s) fetched - 3.593s (0.015s fetch), on 2025-06-22 at 19:11:13



Punto Número 5.d

--query 4

```
select cra.year as make_year, count(cra.year) as total_rent_number from car_rental_analytics cra
where cra.year between 2010 and 2015
group by cra.year order by cra.year desc
```

Results 1 Results 2 Results 3 Results 4

select cra.year as make_year, count(cra.year) as total_rent_number from car_rental_analytics cra where cra.year between 2010 and 2015 group by cra.year order by cra.year desc

	make_year	total_rent_number
1	2015	532
2	2014	382
3	2013	305
4	2012	225
5	2011	200
6	2010	144

Refresh Save Cancel Export data 200 6 6 row(s) fetched - 3.759s (0.016s fetch), on 2025-06-22 at 19:15:52

Save: Save 'localhost> queries_car_rental_table' changes... COT en Writable Smart Insert 17:45:736 Sel: 0 | 0

Punto Número 5.e

--query 5

```
select cra.city, count(cra.city) as total_rent_number from car_rental_analytics cra
where cra.fueltype in ('electric', 'hybrid') group by cra.city order by total_rent_number desc limit 5
```

Results 1 Results 2 Results 3 Results 4 Results 5

select cra.city, count(cra.city) as total_rent_number from car_rental_analytics cra where cra.fueltype in ('electric', 'hybrid') group by cra.city order by total_rent_number desc limit 5

	city	total_rent_number
1	San Diego	44
2	Las Vegas	34
3	Portland	20
4	Phoenix	17
5	San Jose	15

Refresh Save Cancel Export data 200 5 5 row(s) fetched - 3.146s (0.017s fetch), on 2025-06-22 at 19:18:39

Save: Save 'localhost> queries_car_rental_table' changes... COT en Writable Smart Insert 18:1:738 Sel: 0 | 0

Punto Número 5.f

--query 6

```
select cra.fueltype, avg(cra.rating) as avg_rating from car_rental_analytics cra where cra.fueltype is not null
group by cra.fueltype order by avg_rating desc
```

Results 1 Results 2 Results 3 Results 4 Results 5 Results 6

select cra.fueltype, avg(cra.rating) as avg_rating from car_rental_analytics cra where cra.fueltype is not null group by cra.fueltype order by avg_rating desc

	fueltype	avg_rating
1	hybrid	4.9912663755
2	electric	4.9870848708
3	diesel	4.9827586207
4	gasoline	4.9805728518

Refresh Save Cancel Export data 200 4 4 row(s) fetched - 3.578s (0.014s fetch), on 2025-06-22 at 19:22:50

Save: Save 'localhost> queries_car_rental_table' changes... COT en Writable Smart Insert 23:10:949 Sel: 0 | 0

Punto Número 6

De la información extraída de los queries podemos determinar los siguiente:

- a. Hay un total de 771 alquileres de vehículos híbridos y/o eléctricos que tuvieron un rating por encima de 4.
- b. Montana y West Virginia fueron las ciudades donde menos alquileres hubo.
- c. El Tesla Model 3, es el vehículo que más alquilaron. Un total de 288 veces.
- d. Los vehículos fabricados en el 2015 fueron los vehículos más alquilados con 532 veces.
- e. San Diego y las Vegas fueron las ciudades con mayor alquiler de vehículos híbridos y eléctricos
- f. Finalmente, los clientes fueron bastante positivos con los alquileres de los vehículos eléctricos e híbridos

Punto Número 7

El Proyecto podemos decir que es un proyecto de ciencia de datos, con poco volumen de data. Un poco menos de 5,000 registros. Basado, creo que lo más conveniente es conserva la arquitectura on premise. Ingesta de datos de HDFS, trabajo de los datos con pyspark y almacenar la data en HIVE. Como orquestrado nos podemos quedar con Apache Airflow.

[illegible]

a. ¿Para qué se utiliza Dataprep?

b. ¿Qué cosas que se pueden realizar con Dataprep?

- Explorar data
- Limpiar data
- Transformar data
- Preparar la información para posterior análisis, como reportes y o procesos de Machine Learning.

Si entiendo bien la pregunta, por ejemplo, Google Cloud DataPrep, podría fácilmente sustituir herramientas como Tableau, Microsoft SQL Server, IBM Cognos, MicroStrategy entre otras herramientas. Razón del porque haría esto es porque esta herramienta tiene el potencial de combinar varias herramientas dentro de ella dentro de un mismo ecosistema Google Cloud Platform.

Se había mencionado anteriormente, entre los posibles usos podemos listar:

- Explorar data de forma visual. Ver tipos data, patrones, anomalías en la data y potenciales problemas en la data. La herramienta te brinda estadísticos y visualizaciones para poder entender inicialmente la data.
- Limpiar la data, como por ejemplo corregir errores, lidiar con valores nulos y resolver inconsistencias en la data. Remover duplicados, cambiar de tipo de datos entre otros.
- Enriquecer la data. Como por ejemplo campos calculados, extraer información adicional de otras fuentes.
- Generar reportes a partir de una data limpia.

e. ¿Cómo se carga los datos en DataPrep de GCP?

En DataPrep se puede elegir la data a conectar mediante la creación de un dataset. Cuando se está definiendo el dataset nos podemos conectar a Google Cloud Storage o bien podemos conectarnos directamente a una tabla de BigQuery o subir archivos desde la computadora. Una vez que se crea el dataset, se proceder a crear un DataFlow.

f. ¿Qué tipo de datos se pueden preparar en DataPrep en GCP?

Basado en lo visto en el LAB, se pueden preparar datos números y datos categóricos.

g. ¿Qué pasos se pueden seguir para limpiar y transformar datos en DataPrep de GCP?

Una vez creado el dataset y se define el dataflow con un “recipe”, DataPrep genera un dashboard como un limitado número de registros del dataset (sampling). En este dashboard se presenta indicadores de distribución, valores nulos o faltantes y los tipos de datos. Seleccionando las columnas de interés, se puede definir reglas para limpiar, asignar, borrar y transformar datos. Por lo general se generan recomendaciones generadas por inteligencia artificial sobre que hacer con la columna seleccionada.

h. ¿Cómo se pueden automatizar tareas de preparación de datos en DataPrep de GCP?

Por lo general, en el “recipe” creado están definidas todas las reglas. Adicionalmente se puede especificar la frecuencia de corrida del flujo de datos.

i. ¿Qué tipo de visualizaciones se pueden crear en DataPrep de GCP?

Por lo que vi en el LAB no se pueden hacer visualizaciones directamente en DataPrep, pero se puede usar otra herramienta como Google Looker para generar gráficos o reportes tipo Tablau.

j. ¿Cómo se puede garantizar la calidad de datos en DataPrep de GCP?

En los “recipe” se puede limpiar la data y corregir errores, lidiar con valores nulos y resolver inconsistencias en la data. Remover duplicados, cambiar de tipo de datos entre otros.

Arquitectura.

En base a lo requerido, esta podría ser la recomendación.

- a. Para almacenamiento de datos se puede usar Google Cloud Storage.
- b. Para ingestar datos se puede usar el servicio de BigQuery DataTransfer Service.
- c. Para proceder los datos se puede usar Google Cloud Dataproc
- d. Una herramienta para BI, se puede usar looker.
- e. Finalmente una herramienta para podemos simple se puede usar BigQueryML