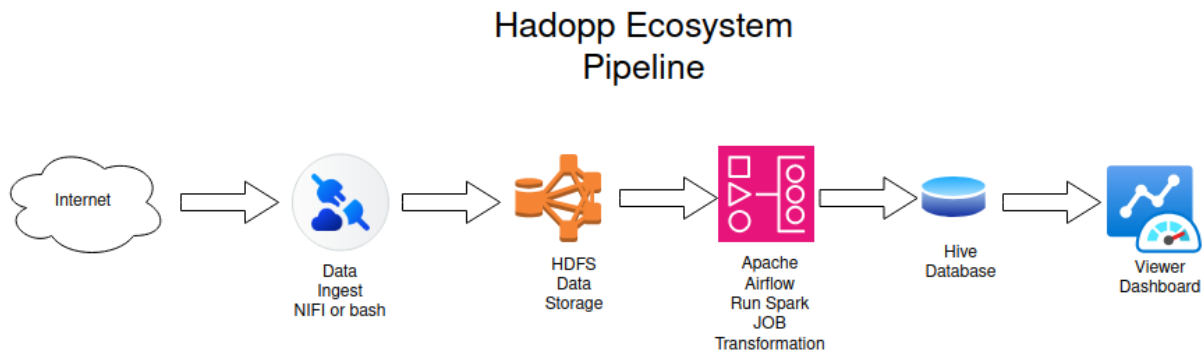


## EJ. 2 Rent a Car

Se realizó un análisis con herramientas open source del ecosistema Hadoop, con una arquitectura como la siguientes



### 1) INGESTA DE DATA

CREAMOS UN DIRECTORIO EN HDFS PARA INGESTAR LOS ARCHIVOS DE ANÁLISIS

```
hdfs dfs -mkdir /ingest2
```

### 2) cargamos los datos

```
https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv
```

```
https://dataengineerpublic.blob.core.windows.net/data-engineer/georef-united-states-of-america-state.csv
```

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

```
wget -P ./ "https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv"
```

```
wget -O ./georef_usa.csv "https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv"
```

**Ingesta completada**

```

hadoop@d4236fd64627:~$ wget -P / "https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv"
--2024-12-18 15:14:17-- https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv
Resolving dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)... 20.150.25.164
Connecting to dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)|20.150.25.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 533157 (521K) [text/csv]
Saving to: './CarRentalData.csv'

CarRentalData.csv                               100%[=====] 520.66K  436KB/s   in 1.2s

2024-12-18 15:14:19 (436 KB/s) - './CarRentalData.csv' saved [533157/533157]

hadoop@d4236fd64627:~$ wget -O ./georef_usa.csv "https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv"
--2024-12-18 15:14:34-- https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv
Resolving dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)... 20.150.25.164
Connecting to dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)|20.150.25.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 533157 (521K) [text/csv]
Saving to: './georef_usa.csv'

./georef_usa.csv                               100%[=====] 520.66K  364KB/s   in 1.4s

2024-12-18 15:14:36 (364 KB/s) - './georef_usa.csv' saved [533157/533157]

hadoop@d4236fd64627:~$

```

## Subimos los archivos a HDFS

```
hdfs dfs -put CarRentalData.csv /ingest2
```

```
hdfs dfs -put georef_usa.csv /ingest2
```

Verificamos si los datos están

```
hdfs dfs -ls /ingest2/
```

Tenemos los archivos cargados

```

hadoop@d4236fd64627:~$ hdfs dfs -mkdir /ingest2
hadoop@d4236fd64627:~$ hdfs dfs -put CarRentalData.csv /ingest2/
hadoop@d4236fd64627:~$ hdfs dfs -put georef_usa.csv /ingest2/
hadoop@d4236fd64627:~$ hdfs dfs -ls /ingest2/
Found 2 items
-rw-r--r--  1 hadoop supergroup      533157 2024-12-18 15:25 /ingest2/CarRentalData.csv
-rw-r--r--  1 hadoop supergroup      533157 2024-12-18 15:25 /ingest2/georef_usa.csv
hadoop@d4236fd64627:~$

```

Para la automatización realizamos un script de ingesta con un archivo .sh

Creamos el script por consola en un .sh

```
#!/bin/bash
```

```
# Mensaje de inicio
```

```
echo "***** Inicio Ingesta Rent a Car *****"
```

```
# Directorio landing en hadoop
```

```
LANDING_DIR="/home/hadoop/landing"
```

```
# Directorio destino en HDFS
```

```
DEST_DIR="/ingest"
```

```
# Nombre archivos
```

```
rentacar="CarRentalData.csv"
```

```
georef="georef-united-states-of-america-state.csv"
```

```
# Descarga archivos
```

```
wget -P $LANDING_DIR -O $LANDING_DIR/$rentacar
"https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv"
wget -P $LANDING_DIR -O $LANDING_DIR/$georef
"https://dataengineerpublic.blob.core.windows.net/data-engineer/georef-united-states-of-america-
state.csv"
```

```
# Mover archivos a HDFS
hdfs dfs -put $LANDING_DIR/$rentacar $DEST_DIR
hdfs dfs -put $LANDING_DIR/$georef $DEST_DIR
```

```
# Remueve archivos, asegurando que el archivo existe
rm -f $LANDING_DIR/$rentacar
rm -f $LANDING_DIR/$georef
```

```
# Mensaje de finalización
echo "\n***** Fin Ingesta rent a car *****"
```

```
nano /home/hadoop/scripts/ingest_car.sh
```

y debemos darle permiso de ejecución

```
chmod +x /home/hadoop/scripts/ingest_car.sh.sh
```

lo ejecutamos para verificar el funcionamiento y chequeemos la carga de los datos

```
bash /home/hadoop/scripts/ingest_car.sh
```

el bash está funcionando

```
hadoop@d5e109c82f52:~/scripts$ chmod +x ingest_car.sh
hadoop@d5e109c82f52:~/scripts$ bash /home/hadoop/scripts/ingest_car.sh
***** Inicio Ingesta Rent a Car *****
--2025-01-05 11:42:34-- https://dataengineerpublic.blob.core.windows.net/data-engineer/CarRentalData.csv
Resolving dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)... 20.150.25.164
Connecting to dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)|20.150.25.164|: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  410KB/s   in 1.3s

2025-01-05 11:42:36 (410 KB/s) - '/home/hadoop/Landing/CarRentalData.csv' saved [533157/533157]

--2025-01-05 11:42:36-- https://dataengineerpublic.blob.core.windows.net/data-engineer/georef-united-states-of-america-state.csv
Resolving dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)... 20.150.25.164
Connecting to dataengineerpublic.blob.core.windows.net (dataengineerpublic.blob.core.windows.net)|20.150.25.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3380726 (3.2M) [text/csv]
Saving to: '/home/hadoop/Landing/georef-united-states-of-america-state.csv'

georef-united-states-of-america-state.csv  100%[=====] 3.22M  946KB/s   in 3.5s

2025-01-05 11:42:41 (946 KB/s) - '/home/hadoop/Landing/georef-united-states-of-america-state.csv' saved [3380726/3380726]

put: '/ing': File exists
\n***** Fin Ingesta rent a car *****
hadoop@d5e109c82f52:~/scripts$
```

y cargado los archivos en HDFS y le damos los permisos

Damos los permisos necesarios de los archivos

```
hdfs dfs -chmod 644 /ingest/*.csv
```

y del directorios

```
hdfs dfs -chmod 755 /ingest
```

```

hadoop@5e109c82f52:~/scripts$ hdfs dfs -ls /ingest
Found 6 items
-rw-r--r-- 1 hadoop supergroup 533157 2025-01-05 11:47 /ingest/CarRentalData.csv
-rw-r--r-- 1 hadoop supergroup 3380726 2025-01-05 11:47 /ingest/georef-united-states-of-america-state.csv
-rwxr-xr-x 1 hadoop supergroup 14334674 2025-01-02 06:40 /ingest/part-00000-6a6df198-426c-4de1-93fb-ca9329563b7e-c000
-rwxr-xr-x 1 hadoop supergroup 10864128 2025-01-02 06:40 /ingest/part-00001-6a6df198-426c-4de1-93fb-ca9329563b7e-c000
-rwxr-xr-x 1 hadoop supergroup 9930312 2025-01-02 06:40 /ingest/part-00002-6a6df198-426c-4de1-93fb-ca9329563b7e-c000
-rwxr-xr-x 1 hadoop supergroup 6885510 2025-01-02 06:40 /ingest/part-00003-6a6df198-426c-4de1-93fb-ca9329563b7e-c000

```

### 3.- Armamos las tablas en Hive:

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

campos	tipo
<code>fuelType</code>	string

<code>rating</code>	integer
<code>renterTripsTaken</code>	integer
<code>reviewCount</code>	integer
<code>city</code>	string
<code>state_name</code>	string
<code>owner_id</code>	integer
<code>rate_daily</code>	integer
<code>make</code>	string
<code>model</code>	string
<code>year</code>	integer

### Pasos previos

show databases;  
con esto vemos que tenemos

creamos una nueva data base

```
-- create database car_rental_db;
```

le indico en que DB voy a trabajar

```
-- use car_rental_db;
```

para saber donde estamos parados

```
-- select current_database();
```

Creamos la tabla car\_rental\_analytics,

```
CREATE EXTERNAL TABLE car_rental_analytics (  
  fuelType STRING,  
  rating INTEGER,  
  renterTripsTaken INTEGER,  
  reviewCount INTEGER,  
  city STRING,  
  state_name STRING,  
  rate_daily INTEGER,  
  make STRING,  
  model STRING,  
  year INTEGER  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
STORED AS TEXTFILE  
LOCATION '/ingest';
```

-

-

Ç

```
hive> show databases;
OK
car_rental_db
default
tripdata
vuelosdb
Time taken: 0.369 seconds, Fetched: 4 row(s)
hive> use car_rental_db;
OK
Time taken: 0.02 seconds
hive> CREATE EXTERNAL TABLE car_rental_analytics (
  >   fuelType STRING,
  >   tipo STRING,
  >   rating INTEGER,
  >   renterTripsTaken INTEGER,
  >   reviewCount INTEGER,
  >   city STRING,
  >   state_name STRING,
  >   rate_daily INTEGER,
  >   make STRING,
  >   model STRING,
  >   year INTEGER
  > )
  > ROW FORMAT DELIMITED
  > FIELDS TERMINATED BY ','
  > STORED AS TEXTFILE
  > LOCATION '/ingest';
OK
Time taken: 0.093 seconds
hive> show tables;
OK
car_rental_analytics
Time taken: 0.025 seconds, Fetched: 1 row(s)
hive> █
```

Tabla creada en HIVE

### 3.- Transformaciones:

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

Script para leer y transformar los datos

```
df_rentacar = spark.read.option("header", "true").option("sep",
",").csv("hdfs://172.17.0.2:9000/ingest/CarRentalData.csv")
```

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import lit
>>> from pyspark.sql import Functions as F
>>> from pyspark.sql.types import FloatType
>>> from pyspark.sql.functions import to_date, col
>>> from pyspark.sql.functions import lit
>>> df_rentacar = spark.read.option("header", "true").option("sep", ",").csv("hdfs://172.17.0.2:9000/ingest/CarRentalData.csv")
>>> df_rentacar.show(5, truncate=False)
```

fuelType	rating	renterTripsTaken	reviewCount	location.city	location.country	location.latitude	location.longitude	location.state	owner.id	rate.daily	vehicle.make	vehicle.model	vehicle.type	vehicle.year
ELECTRIC	5	13	12	Seattle	US	47.449107	-122.308841	WA	12847615	135	Tesla	Model X	suv	2019
ELECTRIC	5	2	1	Tlajeras	US	35.11106	-106.276551	NM	15621242	190	Tesla	Model X	suv	2018
HYBRID	4.92	28	24	Albuquerque	US	35.127163	-106.566681	NM	10199250	35	Toyota	Prius	car	2012
GASOLINE	5	121	20	Albuquerque	US	35.149726	-106.711425	NM	9365406	175	Ford	Mustang	car	2018
GASOLINE	5	13	11	Albuquerque	US	35.208659	-106.601008	NM	3533565	147	Chrysler	Sebring	car	2010

only showing top 5 rows

Con el segundo set de datos, tuve que realizar un ajuste de los datos en Collab ya que no podía leer los datos en PySpark, reemplacé los nombres de las columnas por unos más cortos, borre la columna 'Geo\_Shape' que estaba trayendo problemas con los datos de esa columna y lo convertí a un archivo us\_state\_clean.csv nuevo. Una vez creado lo subí de mi máquina al contenedor de hadoop

```
docker cp /home/demo/Bootcam\ DE\
2024/20241812_final_exam_data_engineering/Ejercicios_data_engineering/renta_car/us_state_clean.csv d5e109c82f52:/home/hadoop/landing/
```

y se está a HDFS.

```
hdfs dfs -put /home/hadoop/landing/us_state_clean.csv /ingest/
```

Ya en PySpark lo ingesté de esta manera

```
df_georef2 = spark.read.option("header", "true").option("sep",
",").csv("hdfs://172.17.0.2:9000/ingest/us_state_clean.csv")
```

obteniendo el nuevo DF a trabajar

```
>>> df_georef2 = spark.read.option("header", "true").option("sep", ",").csv("hdfs://172.17.0.2:9000/ingest/us_state_clean.csv")
>>> df_georef2.show(5, truncate=False)
+-----+-----+-----+-----+-----+-----+-----+-----+
|Geo_Point|Year|Code_State|Name_State|Iso_area_code|Type|US_Postal_state|State_FIPS_Code|State_GNIS_Code|
+-----+-----+-----+-----+-----+-----+-----+-----+
|31.44720801453458, -99.11711684283118|2022|48|Texas|USA|state|TX|null|1779801|
|38.64257169984573, -88.61369740655968|2022|54|West Virginia|USA|state|WV|null|1779805|
|18.215706855012684, -66.41465042883969|2022|72|Puerto Rico|PRI|outlying area|PR|null|1779808|
|40.10998794109383, -74.65593977351195|2022|34|New Jersey|USA|state|NJ|null|1779795|
|20.995094555897666, -158.10992727137122|2022|15|Hawaii|USA|state|HI|null|1779782|
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
>>>
```

## Modificacion de rating

# 2 Redondear la columna 'rating' y castear a int

df\_rentacar = df\_rentacar.withColumn("rating", round(col("rating")).cast("int"))

```
>>> df_rentacar = df_rentacar.withColumn("rating", round(col("rating")).cast("int"))
>>> df_rentacar.show(5, truncate=False)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|FuelType|rating|renterTripsTaken|reviewCount|location.city|location.country|location.latitude|location.longitude|location.state|owner.id|rate.daily|vehicle.make|vehicle.model|vehicle.type|vehicle.year|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|ELECTRIC|5|13|12|Seattle|US|47.449107|-122.308841|WA|12847615|135|Tesla|Model X|suv|2019|
|ELECTRIC|5|2|1|Tijeras|US|35.11106|-106.276551|NM|15621242|190|Tesla|Model X|suv|2018|
|HYBRID|5|128|24|Albuquerque|US|35.127103|-106.566681|NM|10199250|35|Toyota|Prius|car|2012|
|GASOLINE|5|121|126|Albuquerque|US|35.149726|-106.711425|NM|9365496|175|Ford|Mustang|car|2018|
|GASOLINE|5|13|1|Albuquerque|US|35.208659|-106.601008|NM|3553565|147|Chrysler|Sebring|car|2010|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
>>>
```

En el join elijo un “inner join” para tener los datos coincidentes de ambos DF y no generar nulos con otras uniones, solo nos enfocamos en los resultados donde ambos DF tiene coincidencia

Tenemos que hacer un run diccionario para poder unir los dos DF

# vamos a hacer un mapping entre los codigo y las ciudades para unir los dos DF

# Crear el diccionario de mapeo basado en el archivo

# Seleccionar las columnas Code\_State y US\_Postal\_state

df\_georef2\_mapping = df\_georef2.select("Code\_State", "US\_Postal\_state").distinct()

# Convertir el DataFrame a un RDD y coleccionar los resultados

mapping\_list = df\_georef2\_mapping.rdd.collect()

# Crear el diccionario de mapeo

state\_mapping = {str(row["Code\_State"]): row["US\_Postal\_state"] for row in mapping\_list}

```
>>> state_mapping = {str(row["Code_State"]): row["US_Postal_state"] for row in mapping_list}
>>> state_mapping
{'9': 'CT', '54': 'WV', '16': 'ID', '11': 'DC', '20': 'KS', '50': 'VT', '53': 'VA', '42': 'PA', '22': 'LA', '40': 'OK', '78': 'VI', '55': 'WI', '36': 'NY', '25': 'MA', '53': 'MA', '37': 'NC', '18': 'IN', '29': 'MO', '39': 'OH', '44': 'RI', '5': 'AR', '24': 'MD', '47': 'TN', '41': 'AZ', '48': 'TX', '41': 'OR', '60': 'AS', '13': 'GA', '21': 'KY', '46': 'SD', '11': 'AL', '12': 'FL', '26': 'ME', '56': 'WY', '69': 'MP', '61': 'CA', '128': 'MS', '32': 'NV', '17': 'IL', '45': 'SC', '30': 'MT', '66': 'GU', '10': 'DE', '23': 'NE', '8': 'CO', '15': 'HI', '34': 'ND', '19': 'IA', '2': 'AK', '35': 'NM', '31': 'NE', '27': 'MN', '72': 'PR', '38': 'I', 'MD', '33': 'NH', '49': 'UT'}
>>> from pyspark.sql import functions as F
```

# Ver el diccionario

state\_mapping

#agregamos la columna 'state\_code' al df\_georef2



```
df_georef2 = df_georef2.withColumn('state_code', F.lit(None))
```

```
>>> df_georef2 = df_georef2.withColumn('state_code', F.lit(None))
>>> for code, state in state_mapping.items():
...     df_georef2 = df_georef2.withColumn(
...         'state_code',
...         F.when(df_georef2['Code_State'] == code, F.lit(state)).otherwise(df_georef2['state_code'])
...     )
>>> df_georef2.show(5, truncate=False)
```

Geo_Point	Year	Code_State	Name_State	iso_area_code	Type	US_Postal_state	State_FIPS_Code	State_GNIS_Code	state_code
31.447200101453458, -99.31711684283118	2022	48	Texas	USA	state	TX	null	1779801	TX
38.64257169984573, -80.61369740655968	2022	54	West Virginia	USA	state	WV	null	1779805	WV
18.215706855012684, -66.41465042883969	2022	72	Puerto Rico	PRI	outlying area	PR	null	1779808	PR
40.10998794109383, -74.65593977351195	2022	34	New Jersey	USA	state	NJ	null	1779795	NJ
20.995094555897666, -158.10992727137122	2022	15	Hawaii	USA	state	HI	null	1779782	HI

only showing top 5 rows

# Realizar el inner join entre df\_rentacar y df\_georef2 usando las columnas correspondientes

```
renta_join = df_rentacar.join(df_georef2, df_rentacar['`location.state`'] ==
df_georef2['state_code'], 'inner')
```

sobre este nuevo DF hacer las siguientes operaciones

Eliminar los registros con rating nulo

Cambiar mayúsculas por minúsculas en 'fuelType'

Excluir el estado Texas

# Eliminar los registros con rating nulo

```
renta_join_cleaned = renta_join.filter(col("rating").isNotNull())
```

# Cambiar mayúsculas por minúsculas en la columna 'fuelType'

```
renta_join_cleaned = renta_join_cleaned.withColumn('fuelType',
col('fuelType').lower())
```

# Excluir el estado Texas (TX)

```
renta_join_cleaned = renta_join_cleaned.filter(col("`location.state`") != 'TX')
```

# Mostrar el resultado después de las operaciones

```
renta_join_cleaned.show(100, truncate=False)
```

Antes del Join preparamos el DF

# primero sacamos las columnas que no vamos a utilizar en HIVE

```
df_rental_hive = renta_join_cleaned.drop('Geo_Point', 'Year', 'Code_State',  
'Name_State', 'iso_area_code', 'Type', 'US_Postal_state', 'State_FIPS_Code',  
'State_GNIS_Code', 'state_code')
```

#renombramos y casteamos las columnas y datos

```
df_rental_hive = df_rental_hive.selectExpr(  
    "fuelType as fueltype",  
    "CAST(rating AS INT) as rating",  
    "CAST(renterTripsTaken AS INT) as rentertripstaken",  
    "CAST(reviewCount AS INT) as reviewcount",  
    "`location.city` as city", # Usar backticks para acceder a columnas con puntos en  
    el nombre  
    "`location.state` as state_name", # Usar backticks para acceder a columnas con  
    puntos en el nombre  
    "CAST(`rate.daily` AS INT) as rate_daily", # Usar backticks para rate.daily  
    "`vehicle.make` as make", # Usar backticks para vehicle.make  
    "`vehicle.model` as model", # Usar backticks para vehicle.model  
    "CAST(`vehicle.year` AS INT) as year" # Usar backticks para vehicle.year  
)
```

```
>>> df_rental_hive = df_rental_hive.selectExpr(
...     "fuelType as fueltype",
...     "CAST(rating AS INT) as rating",
...     "CAST(renterTripsTaken AS INT) as rentertripstaken",
...     "CAST(reviewCount AS INT) as reviewcount",
...     "`location.city` as city", # Usar backticks para acceder a columnas con puntos en el nombre
...     "`location.state` as state_name", # Usar backticks para acceder a columnas con puntos en el nombre
...     "CAST(`rate.daily` AS INT) as rate_daily", # Usar backticks para rate.daily
...     "`vehicle.make` as make", # Usar backticks para vehicle.make
...     "`vehicle.model` as model", # Usar backticks para vehicle.model
...     "CAST(`vehicle.year` AS INT) as year" # Usar backticks para vehicle.year
... )
>>> df_rental_hive.printSchema()
root
|-- fueltype: string (nullable = true)
|-- rating: integer (nullable = true)
|-- rentertripstaken: integer (nullable = true)
|-- reviewcount: integer (nullable = true)
|-- city: string (nullable = true)
|-- state_name: string (nullable = true)
|-- rate_daily: integer (nullable = true)
|-- make: string (nullable = true)
|-- model: string (nullable = true)
|-- year: integer (nullable = true)
>>>
```

```
>>> df_rental_hive.show(5, truncate=False)
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|fueltype|rating|rentertripstaken|reviewcount|city      |state_name|rate_daily|make   |model   |year|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|ELECTRIC|5      |13              |12          |Seattle   |WA        |135       |Tesla  |Model X|2019|
|ELECTRIC|5      |2               |1           |Tijeras   |NM        |190       |Tesla  |Model X|2018|
|HYBRID  |5      |28              |24          |Albuquerque|NM        |35        |Toyota |Prius  |2012|
|GASOLINE|5      |21              |20          |Albuquerque|NM        |75        |Ford   |Mustang|2018|
|GASOLINE|5      |3               |1           |Albuquerque|NM        |47        |Chrysler|Sebring|2010|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
>>>
```

Ahora ya tenemos la info lista para insertar en Hive

```
hive> DESCRIBE car_rental_analytics;
OK
fueltype                string
tipo                    string
rating                  int
rentertripstaken        int
reviewcount             int
city                    string
state_name              string
rate_daily              int
make                    string
model                   string
year                    int
Time taken: 0.056 seconds, Fetched: 11 row(s)
hive> █
```

#INSERTAMOS

```
df_rental_hive.write.saveAsTable('car_rental_analytics', mode='overwrite')
```

```
>>> df_rental_hive.write.saveAsTable('car_rental_analytics', mode='overwrite')
>>> █
```

```
>>> df_rental_hive.write.saveAsTable('car_rental_analytics', mode='overwrite')
>>> df_rental_hive.show(5, truncate=False)
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|fueltype|rating|rentertripstaken|reviewcount|city
|state_name|rate_daily|make |model |year|
+-----+-----+-----+-----+-----+-----+-----+-----+
|ELECTRIC|5 |13 |12 |Seattle |WA |135 |Tesla |Model
X|2019|
|ELECTRIC|5 |2 |1 |Tijeras |NM |190 |Tesla |Model
X|2018|
|HYBRID |5 |28 |24 |Albuquerque|NM |35 |Toyota |Prius
|2012|
|GASOLINE|5 |21 |20 |Albuquerque|NM |75 |Ford
|Mustang|2018|
|GASOLINE|5 |3 |1 |Albuquerque|NM |47
|Chrysler|Sebring|2010|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

only showing top 5 rows

```
>>> df_rental_hive.printSchema()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'df_rental_hive' is not defined
```

```
>>> df_rental_hive.printSchema
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'df_rental_hive' is not defined
```

```
>>> df_rental_hive.printSchema()
root
|-- fueltype: string (nullable = true)
|-- rating: integer (nullable = true)
|-- rentertripstaken: integer (nullable = true)
|-- reviewcount: integer (nullable = true)
|-- city: string (nullable = true)
|-- state_name: string (nullable = true)
|-- rate_daily: integer (nullable = true)
|-- make: string (nullable = true)
```

```
|-- model: string (nullable = true)
|-- year: integer (nullable = true)
```

```
>>> df_rental_hive.write.saveAsTable('car_rental_analytics', mode='overwrite')
>>>
```

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

Se arman dos DAG el padre llama al dag hijo cuando termina la ingesta

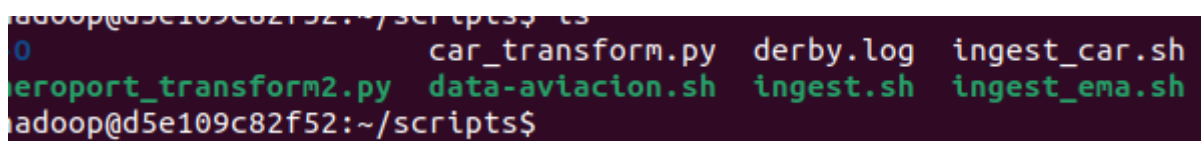
parent\_dag.py se genera en el directorio /home/hadoop/airflow/dags y ya tenemos los archivos

ingest\_car.sh

y

car\_transform.py

en el directorio



```
hadoop@d5e109c82f52:~/scripts$ ls
0                                car_transform.py  derby.log    ingest_car.sh
eroport_transform2.py  data-aviacion.sh  ingest.sh    ingest_ema.sh
hadoop@d5e109c82f52:~/scripts$
```

damos los permisos

```
chmod 555 car_transform.py
```

## Código de los dags

parent\_dag.py responsable de la ingesta

```
from airflow import DAG
from airflow.operators.dummy_operator import DummyOperator
from airflow.operators.subdag_operator import SubDagOperator
from airflow.operators.bash_operator import BashOperator
from datetime import datetime
from child_dag import process_and_load_to_hive # Importa el DAG hijo
```

```

def parent_dag(parent_dag_name, child_dag_name, args):
    dag = DAG(
        child_dag_name,
        default_args=args,
        description="DAG Padre para orquestar la ingestión y procesamiento de
archivos",
        schedule_interval=None, # No es necesario un horario aquí, lo ejecutaremos
manualmente
    )

    start = DummyOperator(
        task_id="start",
        dag=dag
    )

    # Ejecutar el archivo de ingesta (ingest_car.sh) usando el BashOperator
    ingest_files = BashOperator(
        task_id="ingest_files",
        bash_command="bash /home/hadoop/scripts/ingest_car.sh", # Ruta completa
al archivo
        dag=dag
    )

    # SubDAG (DAG hijo)
    process_data = SubDagOperator(
        task_id="process_and_load_to_hive",
        subdag=process_and_load_to_hive(parent_dag_name,
"process_and_load_to_hive", args),
        dag=dag
    )

    # Definir el flujo de trabajo
    start >> ingest_files >> process_data

    return dag

```

ejecutamos el dag

airflow dags trigger car\_parent\_dag

y el del

child\_dag.py

```
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from datetime import datetime
import subprocess

def process_and_load_to_hive(parent_dag_name, child_dag_name, args):
    dag = DAG(
        child_dag_name,
        default_args=args,
        description="DAG Hijo para procesar y cargar datos en Hive",
        schedule_interval=None, # Ejecutar solo desde el DAG Padre
    )

    def run_car_transform():
        # Ejecutar el script car_transform.py usando subprocess
        subprocess.run(['python', '/home/hadoop/scripts/car_transform.py'],
            check=True)

    # Usar PythonOperator para ejecutar el script
    process_data_task = PythonOperator(
        task_id="process_and_load_to_hive_task",
        python_callable=run_car_transform,
        dag=dag
    )

    return dag
```

y damos permiso de ejecución

```
chmod +x parent_dag.py
```

```
chmod +x child_dag.py
```

Tengo cargados los files de los dags en

/home/hadoop/airflow/dags

```
hadoop@ds109c82f52:~/airflow/dags$ ls
2dag_vuelos.py _pycache_ car_child_dag.py car_parent_dag.py child_dag.py dag_ema.py dag_vuelos.py example-DAG.py ingest-transform.py parent_dag.py
hadoop@ds109c82f52:~/airflow/dags$ pwd
```

Tanto el “airflow webserver” y el “schedule” están corriendo

```
hadoop@ds109c82f52:~/airflow/dags$ ps aux | grep "airflow webserver"
hadoop 33918 0.0 0.0 3312 1664 pts/2 S+ 12:40 0:00 grep --color=auto airflow webserver
hadoop@ds109c82f52:~/airflow/dags$ ps aux | grep "airflow scheduler"
hadoop 5745 3.0 0.6 131652 104344 ? S 05:22 17:11 /usr/bin/python3 /home/hadoop/.local/bin/airflow scheduler
hadoop 5749 0.9 0.6 130188 101844 ? S 05:22 4:11 airflow scheduler -- DagFileProcessorManager
hadoop 33940 0.0 0.0 3444 1792 pts/2 S+ 12:40 0:00 grep --color=auto airflow scheduler
```

Pero por alguna razón no me carga lidos dos DAGs,

ID parent\_dag = "ingest\_files"

ID child\_dag = car\_process\_and\_load\_to\_hive\_task"

No aparecen en el listado de los dags

```
hadoop@ds109c82f52:~/airflow/dags$ airflow dags list
dag_id | filepath | owner | paused
-----|-----|-----|-----
Aviacion | dag_ema.py | airflow | True
VUELOS-DAG | dag_vuelos.py | airflow | True
Vuelos-2 | 2dag_vuelos.py | airflow | True
example-DAG | example-DAG.py | airflow | True
example_branch_operator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_operator.py | airflow | True
example_branch_datetime_operator_2 | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_datetime_operator.py | airflow | True
example_branch_dop_operator_v3 | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_python_dop_operator_3.py | airflow | True
example_branch_labels | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_labels.py | airflow | True
example_branch_operator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_operator.py | airflow | True
example_branch_python_operator_decorator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_operator_decorator.py | airflow | True
example_complex | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_complex.py | airflow | True
example_dag_decorator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_dag_decorator.py | airflow | True
example_external_task_marker_child | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_external_task_marker_dag.py | airflow | True
example_external_task_marker_parent | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_external_task_marker_dag.py | airflow | True
example_nested_branch_dag | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_nested_branch_dag.py | airflow | True
example_passing_params_via_test_command | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_passing_params_via_test_command.py | airflow | True
example_python_operator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_python_operator.py | airflow | True
example_short_circuit_operator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_short_circuit_operator.py | airflow | True
example_skip_dag | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_skip_dag.py | airflow | True
example_sla_dag | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_sla_dag.py | airflow | True
example_subdag_operator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_subdag_operator.py | airflow | True
example_subdag_operator.section-1 | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_subdag_operator.py | airflow | True
example_subdag_operator.section-2 | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_subdag_operator.py | airflow | True
example_task_group | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_task_group.py | airflow | True
example_task_group_decorator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_task_group_decorator.py | airflow | True
example_time_delta_sensor_async | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_time_delta_sensor_async.py | airflow | True
example_trigger_controller_dag | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_trigger_controller_dag.py | airflow | True
example_trigger_target_dag | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_trigger_target_dag.py | airflow | True
example_weekday_branch_operator | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_branch_day_of_week_operator.py | airflow | True
example_xcom | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_xcom.py | airflow | True
example_xcom_args | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_xcomargs.py | airflow | True
example_xcom_args_with_operators | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_xcomargs.py | airflow | True
ingest-transform | ingest-transform.py | airflow | True
latest_only | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_latest_only.py | airflow | True
latest_only_with_trigger | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/example_latest_only_with_trigger.py | airflow | True
tutorial | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/tutorial.py | airflow | True
tutorial_etl_dag | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/tutorial_etl_dag.py | airflow | True
tutorial_taskflow_api_etl | /home/hadoop/.local/lib/python3.8/site-packages/airflow/example_dags/tutorial_taskflow_api_etl.py | airflow | True
```

y no logré hacer correr los DAG's.

Reinicie el contenedor varias veces y no logré sacarlo adelante.

5.- Datos insertado en Hive y vistos desde Dbeaver



fueltype	rating	rentertripstaken	reviewcount	city	state_name	rate_daily	make	model	year
ELECTRIC	5	13	12	Seattle	WA	135	Tesla	Model X	2.019
ELECTRIC	5	2	1	Tijeras	NM	190	Tesla	Prius X	2.018
HYBRID	5	28	24	Albuquerque	NM	35	Toyota	Prius	2.012
GASOLINE	5	21	20	Albuquerque	NM	75	Ford	Mustang	2.018
GASOLINE	5	3	1	Albuquerque	NM	47	Chrysler	Sebring	2.010
GASOLINE	5	13	12	Albuquerque	NM	58	Mercedes-Benz	GL-Class	2.012
GASOLINE	4	13	12	Albuquerque	NM	42	GMC	Yukon XL	2.005
GASOLINE	5	12	10	Albuquerque	NM	117	Ford	Expedition	2.018
HYBRID	5	1	1	Albuquerque	NM	102	Ford	Focus RS	2.016
GASOLINE	5	22	17	Albuquerque	NM	49	Ford	EcoSport	2.018

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

```
SELECT COUNT(*) AS total_rentals
FROM car_rental_analytics
WHERE fueltype IN ('HYBRID', 'ELECTRIC')
AND rating >= 4
AND fueltype IS NOT NULL
AND rating IS NOT NULL;
```

total_rentals
771

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

```
SELECT state_name, SUM(renterTripsTaken) AS rental_count
FROM car_rental_analytics
WHERE state_name IS NOT NULL AND state_name != ''
GROUP BY state_name
ORDER BY rental_count ASC
LIMIT 10;
```

```

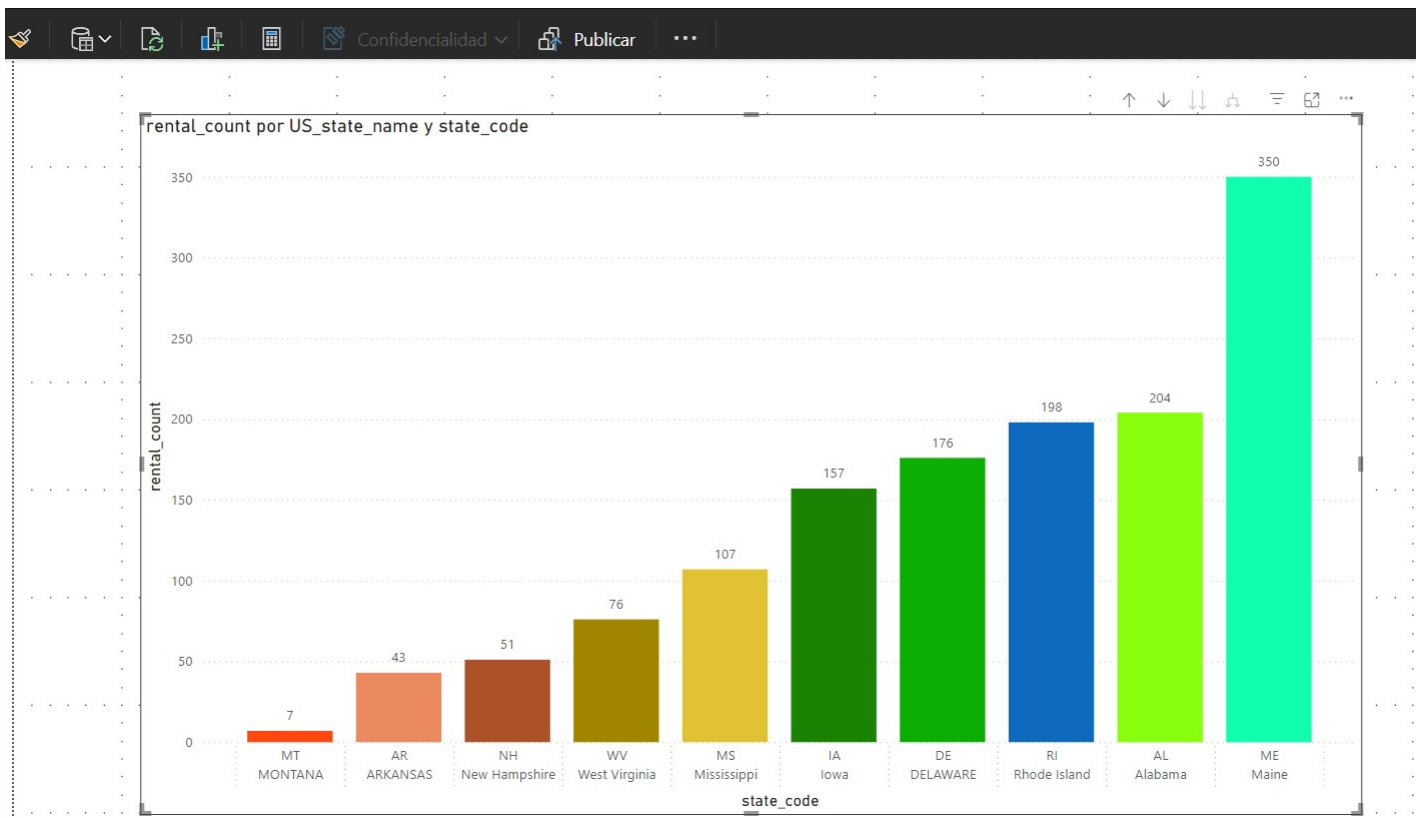
SELECT state_name, SUM(renterTripsTaken) AS rental_count
FROM car_rental_analytics
WHERE state_name IS NOT NULL AND state_name != ''
GROUP BY state_name
ORDER BY rental_count ASC
LIMIT 10;

```

Resultados 1 X

SELECT state\_name, SUM(renterTripsTaken) AS rental\_count

	A-Z state_name	123 rental_count
1	MT	7
2	AR	43
3	NH	51
4	WV	76
5	MS	107
6	IA	157
7	DE	176
8	RI	198
9	AL	204
10	ME	350



Este gráfico lo realicé en PowerBI y muestra la gráfica de la consulta realizada donde se puede ver que los estados de:

- 1.- Montana (MT) con 7 alquileres
- 2.- Arkansas(AR) con 43,
- 3.- New Hampshire con 51 y
- 4.- West Virginia(WV) con 76 tiene menos de 100 alquileres.

Si la estrategia del negocio necesita saber cuales son este tipo de estada para tomar acciones de promoción o reducción de servicios, este gráfico da una imagen clara de la cantidad de de cada estado.

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

```
SELECT make, model, COUNT(*) AS rental_count
FROM car_rental_analytics
WHERE make IS NOT NULL
AND model IS NOT NULL
GROUP BY make, model
ORDER BY rental_count DESC
LIMIT 10;
```

-


```

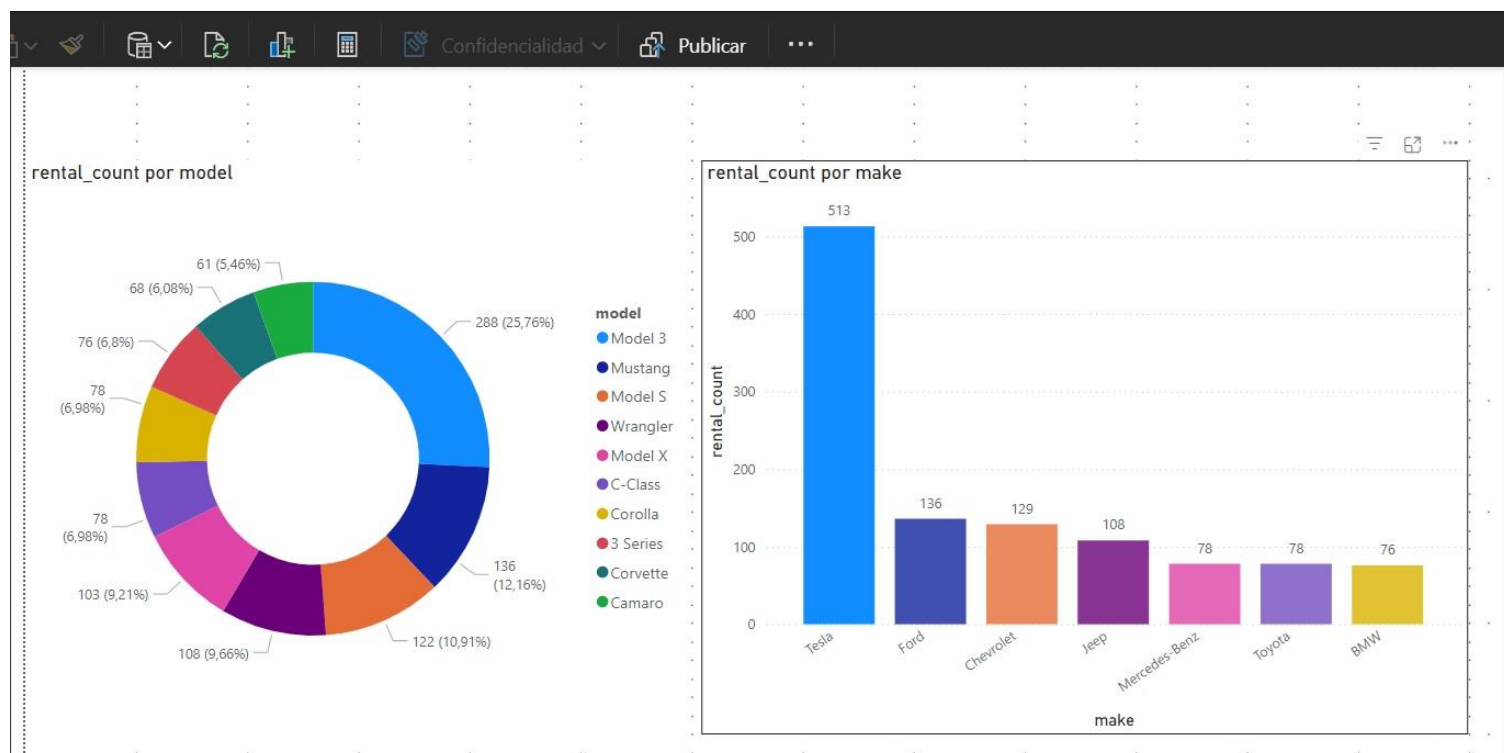
SELECT make, model, COUNT(*) AS rental_count
FROM car_rental_analytics
WHERE make IS NOT NULL
      AND model IS NOT NULL
GROUP BY make, model
ORDER BY rental_count DESC
LIMIT 10;

```

Resultados 1 X

SELECT make, model, COUNT(\*) AS rental\_count F   Enter a SQ

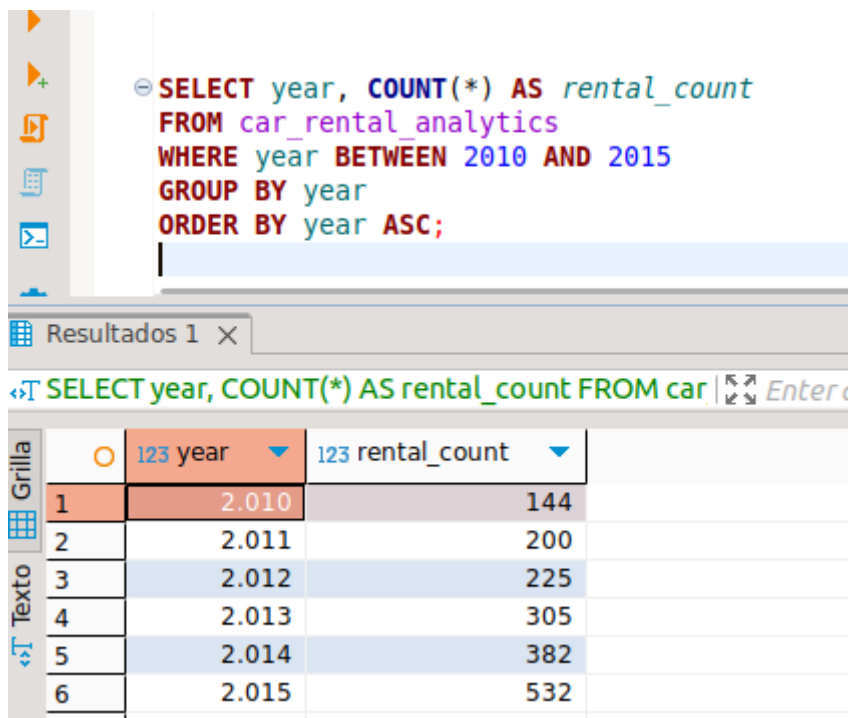
	 A-Z make	A-Z model	123 rental_count
1	Tesla	Model 3	288
2	Ford	Mustang	136
3	Tesla	Model S	122
4	Jeep	Wrangler	108
5	Tesla	Model X	103
6	Toyota	Corolla	78
7	Mercedes-Benz	C-Class	78
8	BMW	3 Series	76
9	Chevrolet	Corvette	68
10	Chevrolet	Camaro	61



Esta consulta y estos gráficos muestran que los autos marca Tesla, con su modelo Model 3 es el auto más alquilado, y su modelo Model S es el tercero y el model X en quinto lugar en la lista, siendo también la marca Tesla la marca más alquilada. El segundo modelo más alquilado es el Mustang de Ford. El tercer lugar se lo lleva Chevrolet con sus modelos Corvette y Camaro y el cuarto modelo es la Jeep Wrangler. Telsla representa el 25,76% del total de los alquileres, seguido por Ford con un 12,16% y Chevrolet con un 11,54 % sobre el total de los alquileres. Lo que queda claramente expreso en este gráfico es que los Clientes quieren manejar un TESLA

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

```
SELECT year, COUNT(*) AS rental_count
FROM car_rental_analytics
WHERE year BETWEEN 2010 AND 2015
GROUP BY year
ORDER BY year ASC;
```



The screenshot shows a SQL IDE interface. At the top, a query is entered in a text editor:

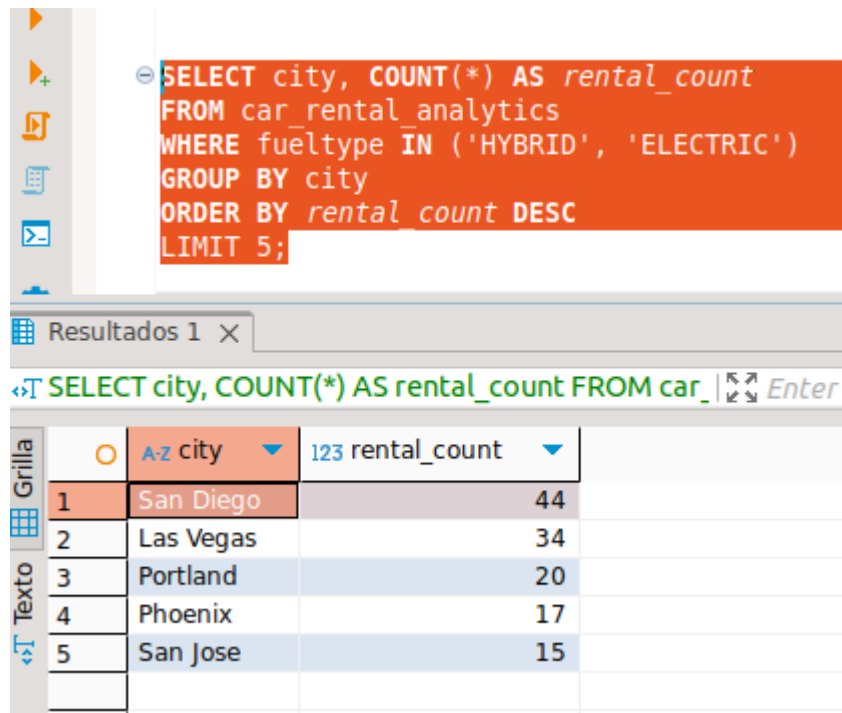
```
SELECT year, COUNT(*) AS rental_count
FROM car_rental_analytics
WHERE year BETWEEN 2010 AND 2015
GROUP BY year
ORDER BY year ASC;
```

Below the editor, a tab labeled "Resultados 1" is active. It shows the same query and a table of results. The table has two columns: "year" and "rental\_count". The results are ordered by year from 2010 to 2015.

	year	rental_count
1	2.010	144
2	2.011	200
3	2.012	225
4	2.013	305
5	2.014	382
6	2.015	532

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

```
SELECT city, COUNT(*) AS rental_count
FROM car_rental_analytics
WHERE fueltype IN ('HYBRID', 'ELECTRIC')
GROUP BY city
ORDER BY rental_count DESC
LIMIT 5;
```



```
SELECT city, COUNT(*) AS rental_count
FROM car_rental_analytics
WHERE fueltype IN ('HYBRID', 'ELECTRIC')
GROUP BY city
ORDER BY rental_count DESC
LIMIT 5;
```

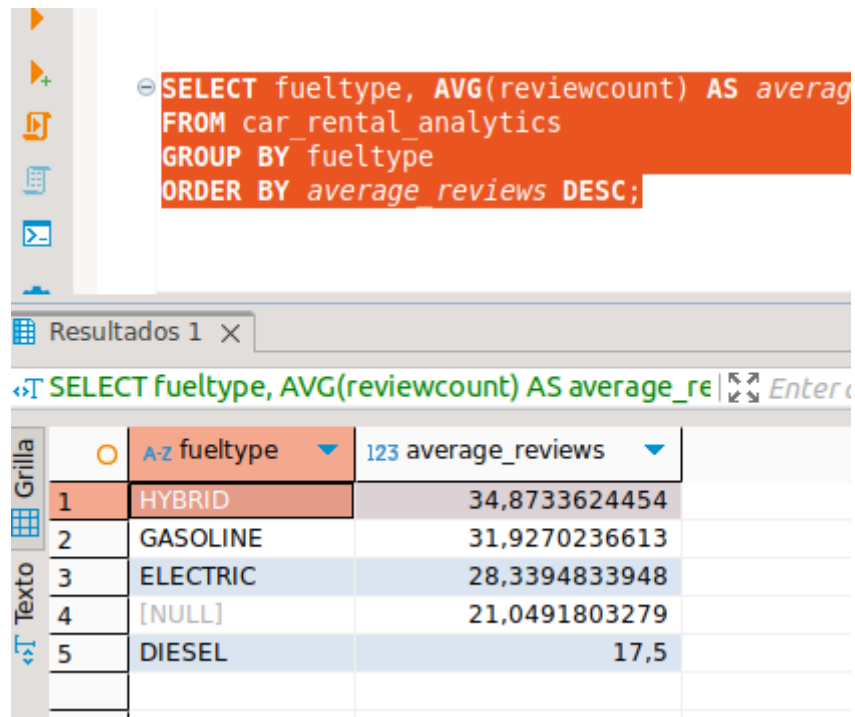
Resultados 1

SELECT city, COUNT(\*) AS rental\_count FROM car\_ | Enter

	A-z city	rental_count
1	San Diego	44
2	Las Vegas	34
3	Portland	20
4	Phoenix	17
5	San Jose	15

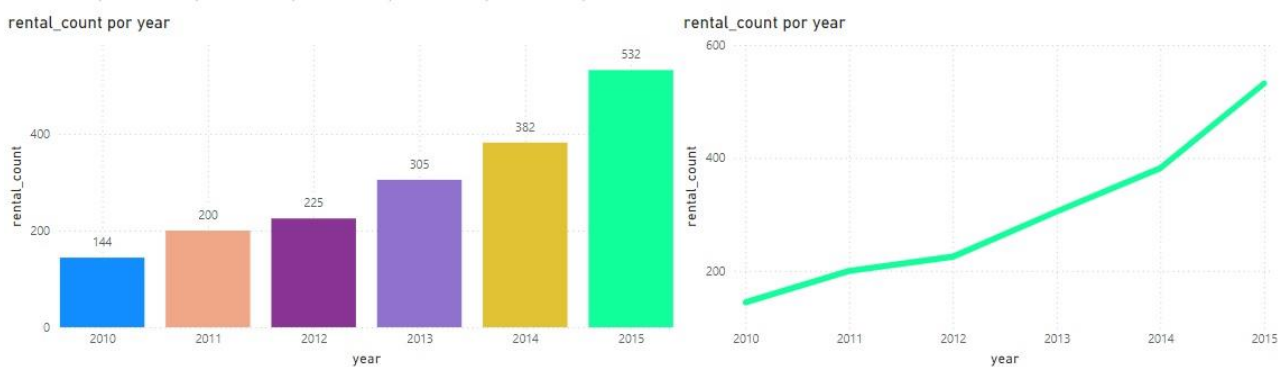
f.- el promedio de reviews, segmentando por tipo de combustible

```
SELECT fueltype, AVG(reviewcount) AS average_reviews
FROM car_rental_analytics
GROUP BY fueltype
ORDER BY average_reviews DESC;
```



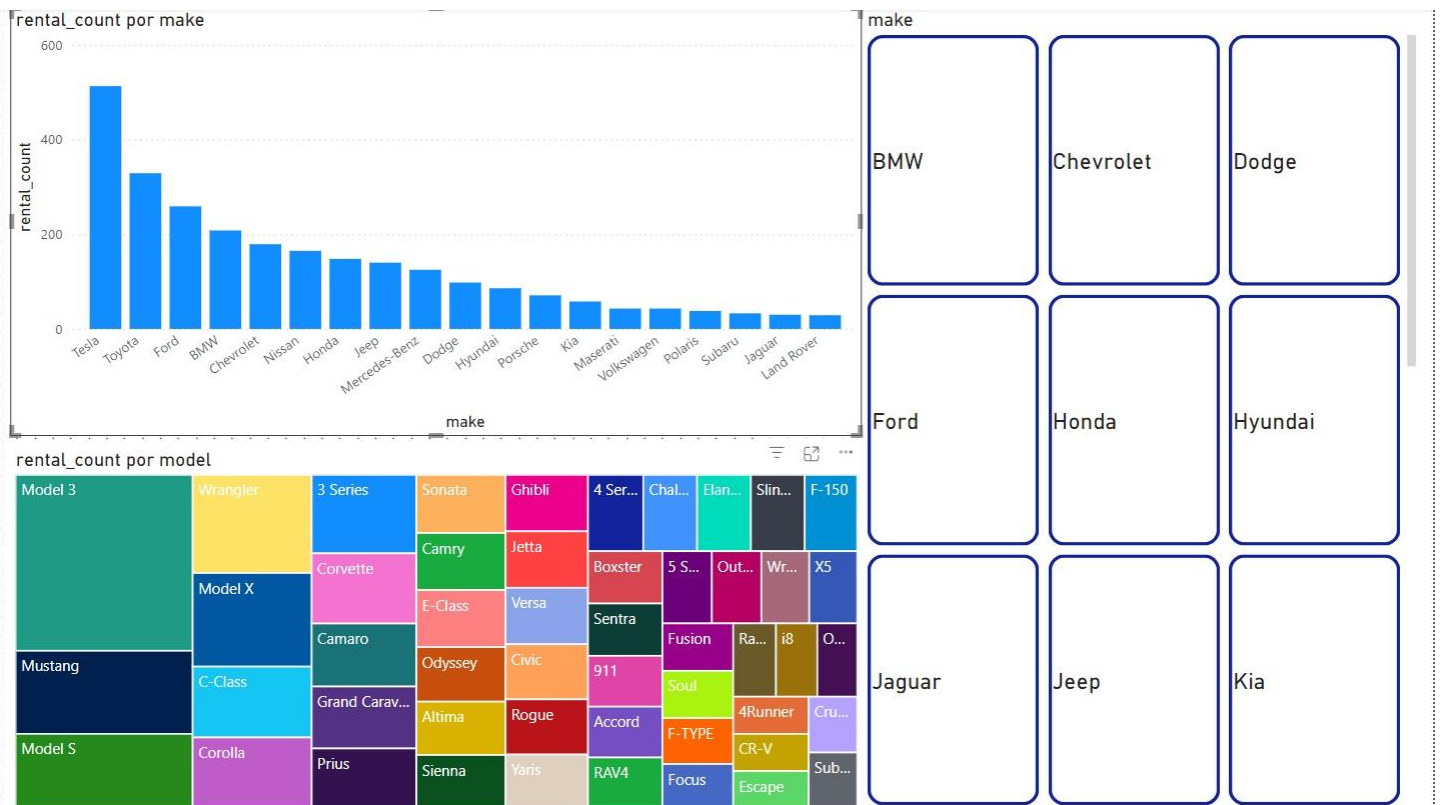
## 6.- Algunas conclusiones sobre este análisis:

A continuación podrán observar algunas gráficos adicionales para este análisis.

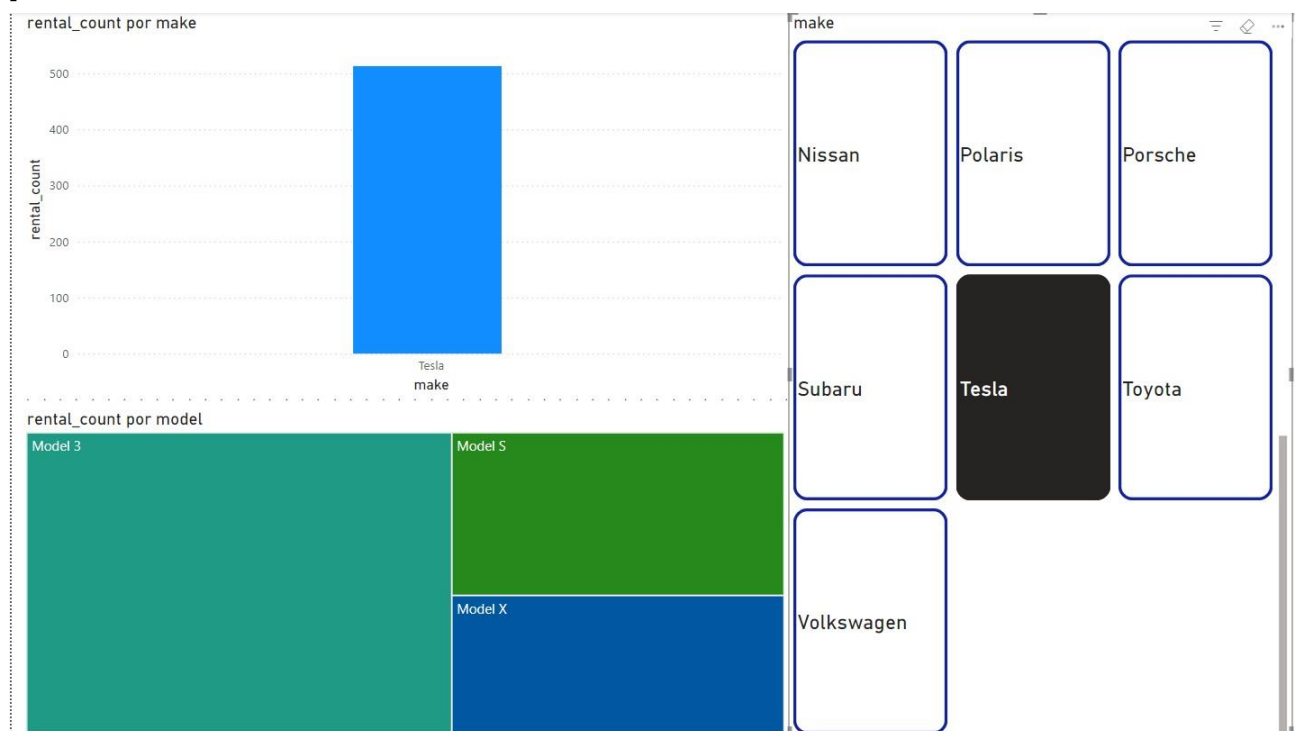


En el mismo vemos que a través de los años el número de alquileres fue relativamente constante en aumento.

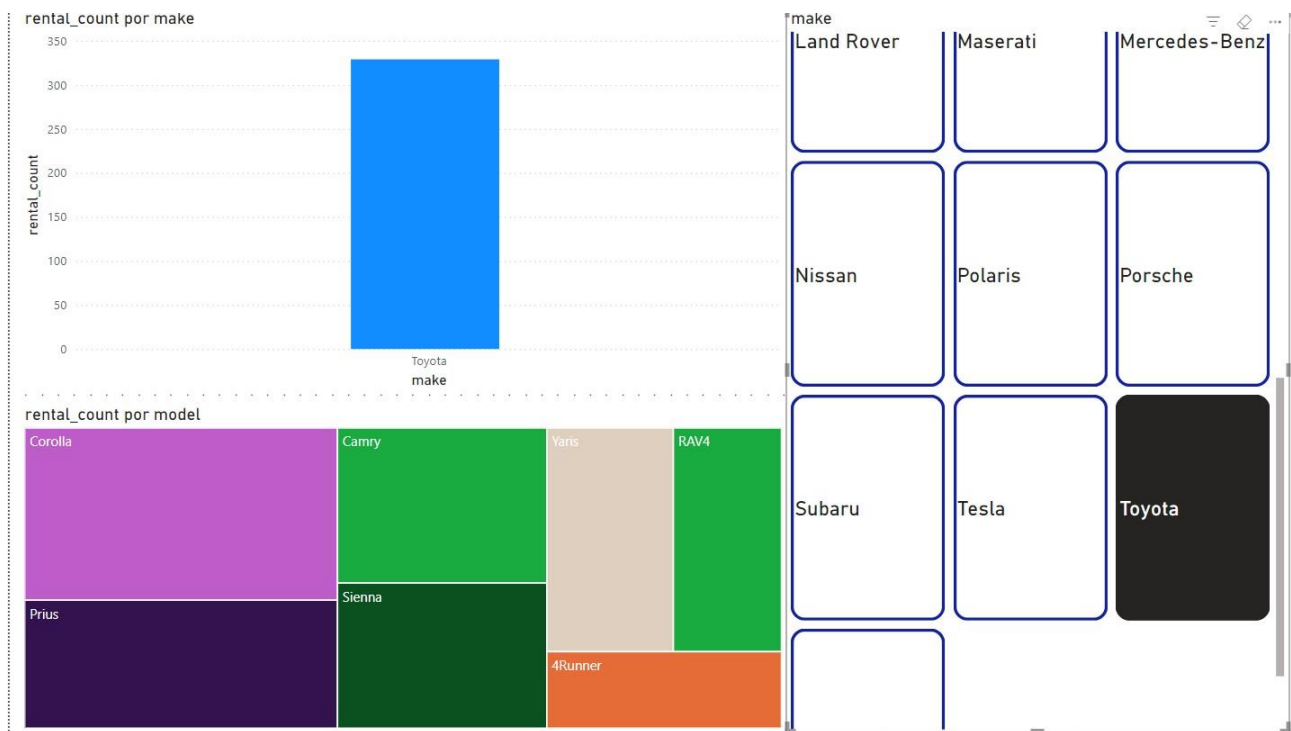




Este gráfico es el total de alquileres por marca y modelo, como ya hemos visto TESLA lleva la delantera de alquiler. -



TESLA son un autos eléctricos, tiene 3 tipos de modelo, lleva a pensar que son para viajes no demasiados largos, máximo 600 km, ya que luego debe cargar las baterías, máximo tiempo de carga 1 hs. Otra ventaja que pueden tener los TESLA es su equipamiento tecnológico entre ellos, la posibilidad de conducción automática. Pero vemos otras marcas.



En este caso TOYOTA, que es la segunda en el ranking, tiene 7 autos para elegir y hasta el 2015, horizonte de estudio de los datos, las opciones eran:

Híbridos:

Toyota Prius (Siempre híbrido).

Toyota Camry Hybrid (Disponible hasta 2015).

Toyota RAV4 Hybrid (Disponible desde 2015).

Gasolina:

Toyota Corolla.

Toyota Sienna.

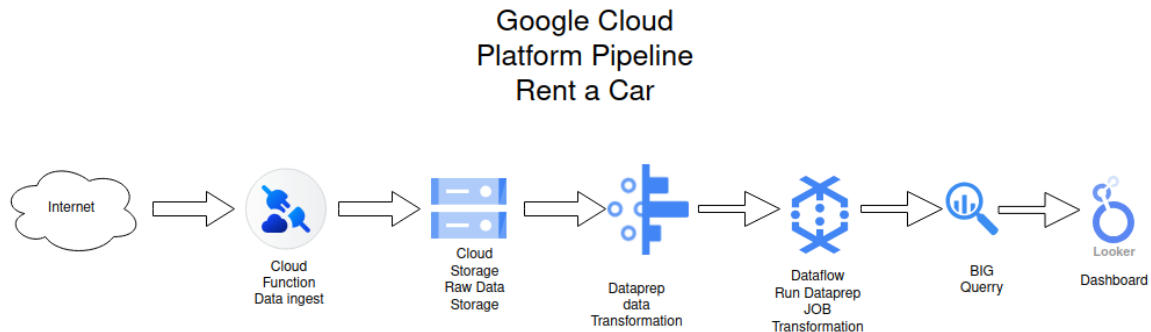
Toyota Yaris.

Toyota 4Runner.

Lleva a pensar que es el segundo por oferta de tipos de vehículos y las ventajas de los autos híbridos, que permiten una mayor autonomía. El Corolla, auto insignia de ella marca, lleva la delantera en los alquileres y es un auto a combustible. Le sigue el Prius, el primer híbrido de la marca y el Camry, todo un clásico y la familia Sienna, muy popular en el país del Norte.

Con estos breves análisis podemos llegar a la conclusión que la empresa es una empresa en crecimiento, adaptándose al mercado con la incorporación de nuevos productos como los híbridos pero también incorporando tecnología a la empresa, con 19 marcas de vehículos en su oferta desde auto de alta tecnología hasta un clásico como el Mustang. Con presencia en la mayoría de estados y principales ciudades de Estados Unidos.

## Arquitectura Alternativa



Para realizar el análisis de de estos dataset podemos utilizar una infraestructura Cloud para dotar al dicho trabajo y futuros análisis de escalabilidad, gobernanza de datos y seguridad.

La arquitectura propuesta es sobre Google Cloud Platform (GCP). Los pasos a reaizar, son los siguientes:

- 1.- Cloud Function: en esta etapa a través de una función que ingesta, por ejemplo a través de un scrip PYSPARK, los datos desde intenet.
- 2.- Cloud Storage: los datos ingestados coo raw data son guardados en Cloud Storage, el almacenamiento masivo de GCP, donde se realiza el storage de los datos para luego ser utilizados en las etapas de ETL.
- 3.- Dataprep: para realizar el ETL de los datos de forma dinámica, permitiendo ganar tiempo en la transformación y análisis de los datos. Una vez realizada la transformación según la estrategia del negocio, los datos se guardan en una tabla de BigQuery.
- 4.- Big Querry, ya con los datos transformados en BQ podemos realizar las consultas necesarias para poder obtener información relevante del negocio. Se pueden generar distintas vistas y ser graficadas a posteriari.
- 5.- Looker u PowerBI: son herramientas de visualización que se conectan con Big Query y nos permiten realizar Dashboard sobre los diferentes análisis de datos que realizamos en BigQuery. Estos gráficos son interactivos y permiten tener una imagen relacional del negocio.