Proyecto OpenSky

Minería de Datos

Guillermo Ovejero Sánchez

Índice general

1 Datos de Opensky

1.1. Esquema API

La API nos devuelve los datos en formato JSON, el usado como estandar en la mayoria de APIs de hoy en dia debido al gran soporte que tiene y la gran legibilidad.

El formato JSON (JavaScript Object Notation) se compone de objetos ‘ ’, campos ‘x’ : ‘10’ y listas ‘[ ]’

Ejemplo:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

Al devolvernos este formato y según las necesidades del caso de uso podríamos guardar la información en distintos tipos de bases de datos NoSQL.
Como por ejemplo MongoDB para una colección de documentos en el cual podamos realizar querys. O por ejemplo si quisiésemos saber rápidamente las trayectorias de los aviones estas podrían guardarse en una BBDD de Clave-Valor como por ejemplo Redis. Internamente los resultados históricos se almacenan en una base de datos con Hadoop a la cual dan acceso por petición

La api se encuentra en el dominio https://opensky-network.org/apidoc/ y los endpoints se añaden al final de esta ruta.

A traves de esta API se puede obtener principalmente información sobre vuelos, de la cual obtenemos todos estos datos:

StateVector

1. **icao24** - ICAO24 address of the transmitter in hex string representation.

2. **callsign** - callsign of the vehicle. Can be None if no callsign has been received.
3. **origin_country** - inferred through the ICAO24 address
4. **time_position** - seconds since epoch of last position report. Can be None if there was no position report received by OpenSky within 15s before.
5. **last_contact** - seconds since epoch of last received message from this transponder
6. **longitude** - in ellipsoidal coordinates (WGS-84) and degrees. Can be None
7. **latitude** - in ellipsoidal coordinates (WGS-84) and degrees. Can be None
8. **geo_altitude** - geometric altitude in meters. Can be None
9. **on_ground** - true if aircraft is on ground (sends ADS-B surface position reports).
10. **velocity** - over ground in m/s. Can be None if information not present
11. **heading** - in decimal degrees (0 is north). Can be None if information not present.
12. **vertical_rate** - in m/s, incline is positive, decline negative. Can be None if information not present.
13. **sensors** - serial numbers of sensors which received messages from the vehicle within the validity period of this state vector. Can be None if no filtering for sensor has been requested.
14. **baro_altitude** - barometric altitude in meters. Can be None
15. **squawk** - transponder code aka Squawk. Can be None
16. **spi** - special purpose indicator
17. **position_source** - origin of this state's position: 0 = ADS-B, 1 = ASTERIX, 2 = MLAT, 3 = FLARM

La API tiene diferentes endpoints, pero muchos no se ha conseguido que devolviesen información para vuelos actuales (algunas ni siquiera los ejemplos de la documentación funcionaban):

1. states/all
2. tracks
3. routes
4. airports
5. flights

1.2. Impala Shell - Hadoop

Para acceder aqui se tuvo que rellenar un formulario en https://opensky-network.org/data/apply.

Esta Base de Datos tiene diferentes tablas, algunas particionadas por día u hora para hacer queries mas eficientes. la lista de tablas que hay actualmente es la siguiente:

```
[hadoop-29:21000] > show tables;
+-----------------------------+
| name                        |
+-----------------------------+
| acas_data4                  |
| allcall_replies_data4       |
| flarm_raw                   |
| flights                     |
| flights_data4               |
| identification_data4        |
| operational_status_data4    |
| position_data4              |
| rollcall_replies_data4      |
| sensor_visibility           |
| sensor_visibility_data3     |
| state_vectors               |
| state_vectors_data3         |
| state_vectors_data4         |
| velocity_data4              |
+-----------------------------+
```

De las cuales las mas interesantes y que mas información útil se puede sacar son las siguientes:

1. flights & flights_data4
2. position_data4
3. state_vectors_data_3 & state_vectors_data_4 (particionada por hora)

En esta tabla actualmente no se puede acceder al campo 'track'

```
[hadoop-29:21000] > describe flights_data4
+-----------------------------------+-----------------------------+
| name                              | type                        |
+-----------------------------------+-----------------------------+
| icao24                            | string                      |
| firstseen                         | int                         |
| estdepartureairport               | string                      |
```

```
| lastseen                          | int                       |
| estarrivalairport                 | string                    |
| callsign                          | string                    |
| track                             | array<struct<             |
|                                   |   time:int,               |
|                                   |   latitude:double,        |
|                                   |   longitude:double,       |
|                                   |   altitude:double,        |
|                                   |   heading:float,          |
|                                   |   onground:boolean        |
|                                   | >>                        |
| serials                           | array<int>                |
| estdepartureairporthorizdistance  | int                       |
| estdepartureairportvertdistance   | int                       |
| estarrivalairporthorizdistance    | int                       |
| estarrivalairportvertdistance     | int                       |
| departureairportcandidatescount   | int                       |
| arrivalairportcandidatescount     | int                       |
| otherdepartureairportcandidates   | array<struct<             |
|                                   |   icao:string,            |
|                                   |   horizdistance:int,      |
|                                   |   vertdistance:int        |
|                                   | >>                        |
| otherarrivalairportcandidates     | array<struct<             |
|                                   |   icao:string,            |
|                                   |   horizdistance:int,      |
|                                   |   vertdistance:int        |
|                                   | >>                        |
| day                               | int                       |
+-----------------------------------+---------------------------+
```

```
[hadoop-29:21000] > describe position_data4;
+-------------+-------------------------+---------+
| name        | type                    | comment |
+-------------+-------------------------+---------+
| sensors     | array<struct<           |         |
|             |   serial:int,           |         |
|             |   mintime:double,       |         |
|             |   maxtime:double        |         |
|             | >>                      |         |
| rawmsg      | string                  |         |
| mintime     | double                  |         |
| maxtime     | double                  |         |
| msgcount    | bigint                  |         |
| icao24      | string                  |         |
```

nicsuppla	boolean	
hcr	double	
nic	smallint	
survstatus	smallint	
nicsupplb	boolean	
odd	boolean	
baroalt	boolean	
lat	double	
lon	double	
alt	double	
nicsupplc	boolean	
groundspeed	double	
gsresolution	double	
heading	double	
timeflag	boolean	
surface	boolean	
hour	int	

```
[hadoop-29:21000] > describe state_vectors_data4;
```

name	type	comment
time	int	Inferred from Parquet
icao24	string	Inferred from Parquet
lat	double	Inferred from Parquet
lon	double	Inferred from Parquet
velocity	double	Inferred from Parquet
heading	double	Inferred from Parquet
vertrate	double	Inferred from Parquet
callsign	string	Inferred from Parquet
onground	boolean	Inferred from Parquet
alert	boolean	Inferred from Parquet
spi	boolean	Inferred from Parquet
squawk	string	Inferred from Parquet
baroaltitude	double	Inferred from Parquet
geoaltitude	double	Inferred from Parquet
lastposupdate	double	Inferred from Parquet
lastcontact	double	Inferred from Parquet
serials	array<int>	Inferred from Parquet
hour	int	

2 Obtención de Datos

Los datos para los siguientes análisis se han obtenido de ambas fuentes de datos (API, Hadoop).

2.1. API

Para los datos de la API se realizo un proyecto en Python para envolver las llamadas de la API en comandos sencillos. Ejemplo de la autenticación de la API y una llamada al endpoint de 'states'

```python
class OpenSkyApi:
  @staticmethod
  def get_auth(username, password):
      return base64.b64encode(
              bytes(":".join([username, password]), "utf-8"))
              .decode("utf-8")

  API = "https://opensky-network.org/api/"

  def __init__(self, username=None, password=None):
      self._auth = OpenSkyApi.get_auth(username, password)
      self._headers = {
          "Authorization": "Basic {}".format(self._auth),
      }
      self.last_request = None

  def get_states(self):
      url = OpenSkyApi.API + "states/all"
      response = requests.request("GET", url,
          headers=self._headers, data={})
      if not response.ok:
          return None
      return json.loads(response.text)
```

2.2. Hadoop

Para obtener los datos de esta base de datos distribuida se tiene que acceder al *Impala Shell*, el cual se accede con un usuario y contraseña a través de SSH y para la obtención se utiliza una lenguaje SQL-Like, probablemente Hive, para consultar sobre el sistema distribuido, algunas querys realizadas de prueba superaban el TeraByte de datos. Para acceder aquí y comprender como funciona el entorno *OpenSky* tiene una guía en su pagina para familiarizarte con el entorno: https://opensky-network.org/data/impala

Esta interfaz es limitada pues solo nos permite obtener los datos que capturemos a través de la terminal. Algunos ejemplos de las Querys realizadas para la obtencion de datos son estas:

Obtener un vector de estados para un vuelo aleatorio en concreto.

```
SELECT time, icao24, lat, lon, velocity, heading, vertrate, callsign,
onground, baroaltitude, geoaltitude, lastposupdate, lastcontact, hour
FROM state_vectors_data4
WHERE hour=unix_timestamp('2021-05-16 12:00:00')
ORDER BY rand() LIMIT 1;
```

Vuelos desde al 10 hasta el 20 de Mayo

```
SELECT *
FROM state_vectors_data4 v
JOIN (SELECT QUOTIENT(time, 60) AS minute, MAX(time) AS recent, icao2
FROM state_vectors_data4
WHERE hour>=unix_timestamp('2021-05-14 01:00:00')
GROUP BY icao24, minute) AS m ON v.icao24=m.icao24 AND v.time=m.recen
WHERE v.lat<=43.74 AND v.lat>=35.94
AND v.lon<=3.03 AND v.lon>=-9.39
AND v.hour>=unix_timestamp('2021-04-30 01:00:00')
AND v.hour<=unix_timestamp('2021-05-21 23:00:00');
```

Datos de tabla de flights_data4

```
SELECT *
FROM flights_data4
WHERE day>=unix_timestamp('2021-04-29 00:00:00')
AND day<=unix_timestamp('2021-05-22 00:00:00');
```

3 Filtrado de Datos

Para el filtrado de datos al estar los datos en formato CSV se han realizado varias operaciones sobre ellos, con diferentes herramientas dependiendo del caso necesario. Para algunos se ha cargado un DataFrame en python y con este se han hecho operación para eliminar algunos campos que estaban vacíos o filtrar datos que no correspondían a lo que se quería mostrar.
También se ha filtrado mediante las consultas de SQL directamente al obtener los datos (a través del Impala Shell). Y por ultimo a través de la interfaz de DataBricks con la cual se podía cargar la tabla con Spark y filtrarla o bien mediante operaciones en el DataFrame o al visualizarlos con SQL.

Un ejemplo de filtrado y modificacion sobre un dataset:

```
flights = spark.read.table("flights_csv")
flights = flights.withColumn("day",col("day")
            .cast("timestamp"))

flights_departure = flights
    .groupBy(flights.estdepartureairport,
    window(flights.day, "1 hour"))
    .count()

flights_spain = flights_departure
    .join(airports,
    flights_departure.estdepartureairport == airports.icao,
    "full")

flights_spain = flights_spain
    .select("estdepartureairport","window","name","count")
    .where(col("icao").isNotNull())

flights_spain = flights_spain
    .select("*")
    .where(col("window").isNotNull())
```

4 Análisis con Python & DataBricks

4.1. Análisis de Airports Data

Para realizar el analisis de los aeropuertos se obtuvo a traves de la API los datos de todos los aeropuertos y el cual luego se filtro para coger solamente los que estaban en territorio español filtrando por las coordenas (latitud y longitud) de España, este analisis se realizo en un Notebook de Python puramente, para ello se usaron varias librerias como Pandas o Geopandas y Matplotlib. El mapa obtenido de https://gadm.org/download_country_v3.html fue usado por Geopandas para poder mostrar la posición de estos aeropuertos encima de un mapa.

```
df = df.loc[~((df["longitude"] < -7) & (df["latitude"] < 42))
     & ~((df["longitude"] > -1.5)
     & (df["latitude"] >42.5))]

 df_large = df.loc
     [df["type"] == "large_airport"]
 df_large.drop(
     columns=["position", "altitude", "geometry"], axis=1)
 df_large.to_csv("test.csv")
```

index	icao	iata	name	type	latitude	longitude
8	LEAL	ALC	Alicante International Airport	large_airport	38.28	-0.55
16	LEBL	BCN	Barcelona International Airport	large_airport	41.297	2.078
57	LEMD	MAD	Adolfo Suárez MadridBarajas Airport	large_airport	40.471	-3.562
59	LEMG	AGP	Málaga Airport	large_airport	36.674	-4.499
71	LEPA	PMI	Palma De Mallorca Airport	large_airport	39.551	2.738
88	LEST	SCQ	Santiago de Compostela Airport	large_airport	42.896	-8.415

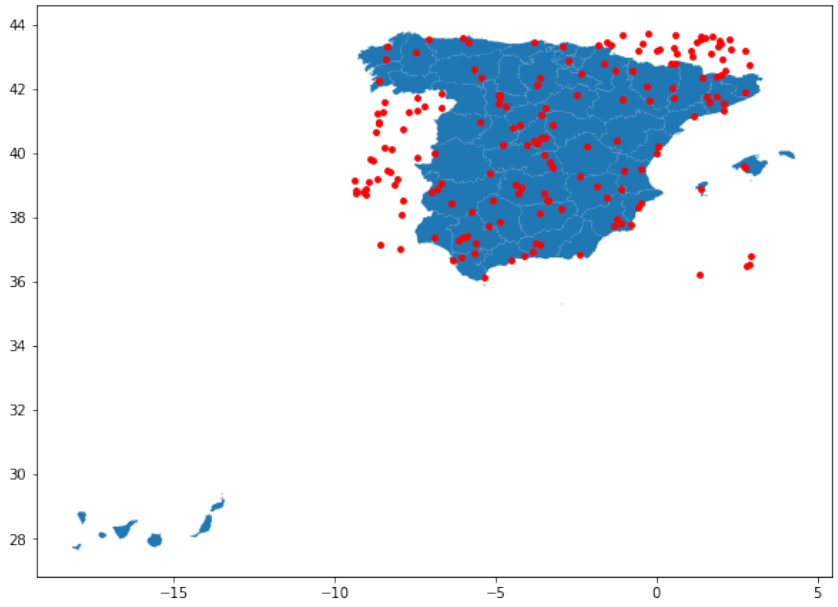

Figura 4.1: Aeropuertos sin procesar

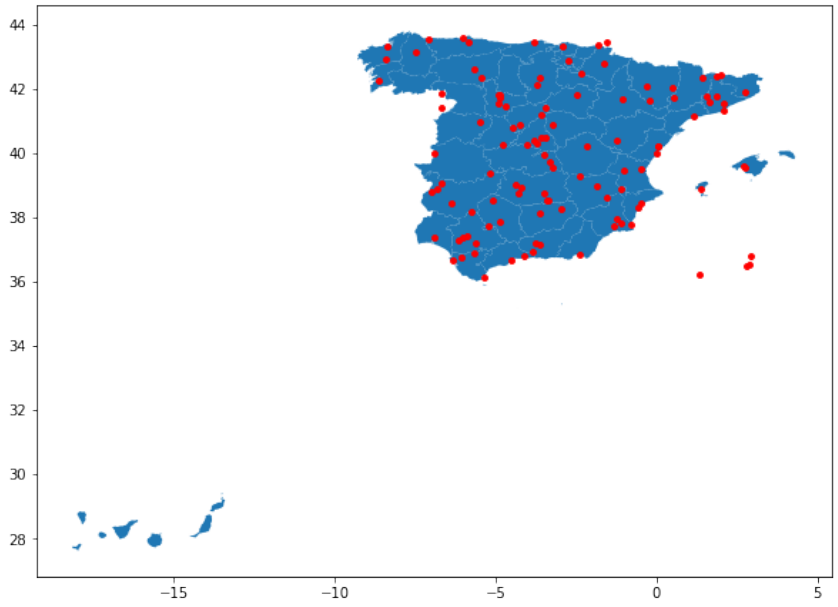

Figura 4.2: Eliminación de outliers

Posteriormente se analizo y mostró impreso en el mapa los aeropuertos en sus posición según el tamaño de estos, en amarillo los pequeños y en rojo los medianos/grandes

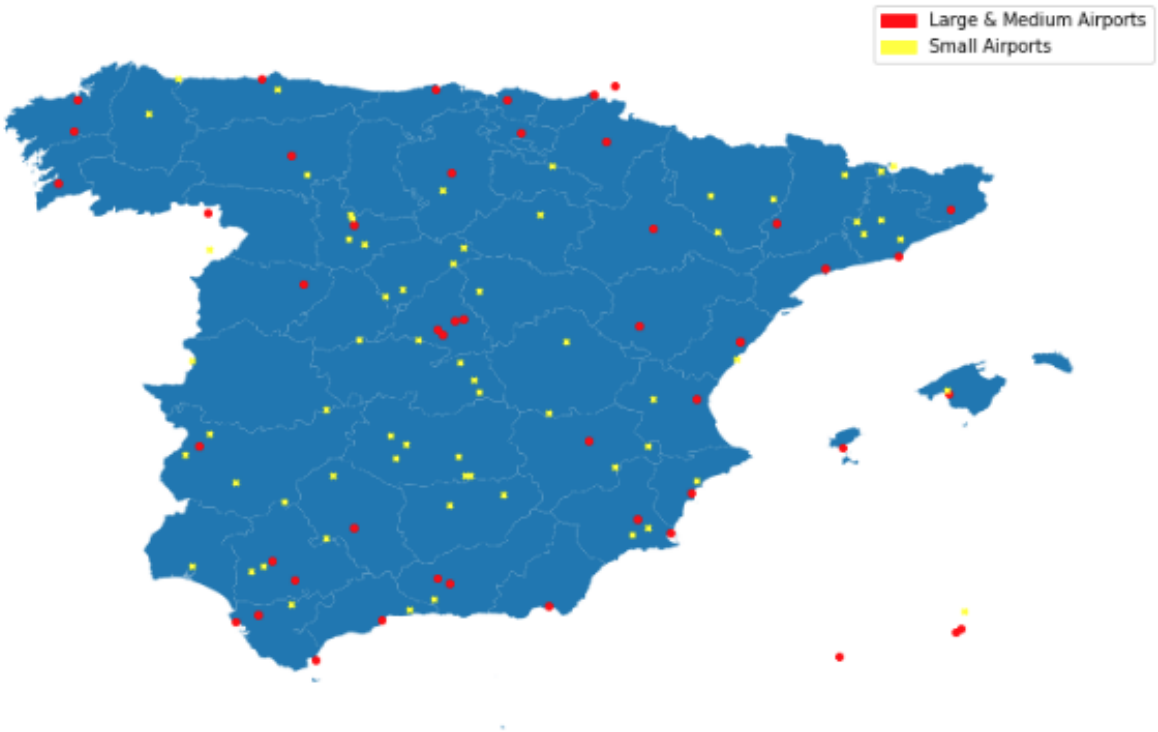

Figura 4.3: Aeropuertos diferenciados por tipo

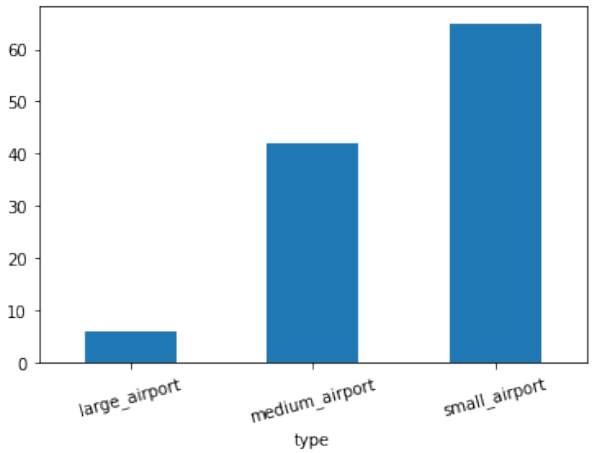

Figura 4.4: Aeropuertos diferenciados por tipo

Como se puede observar la gran mayoría de aeropuertos grandes donde un avión de pasajeros podría aterrizar se encuentran en las ciudades mas grandes y donde mas turismo llega, dejando así a muchas comunidades sin un aeropuerto de estas dimensiones, lo que hace que el movimiento hasta estos sitios deba hacerse por otro medio de transporte (Tren,Coche,Bus, etc).

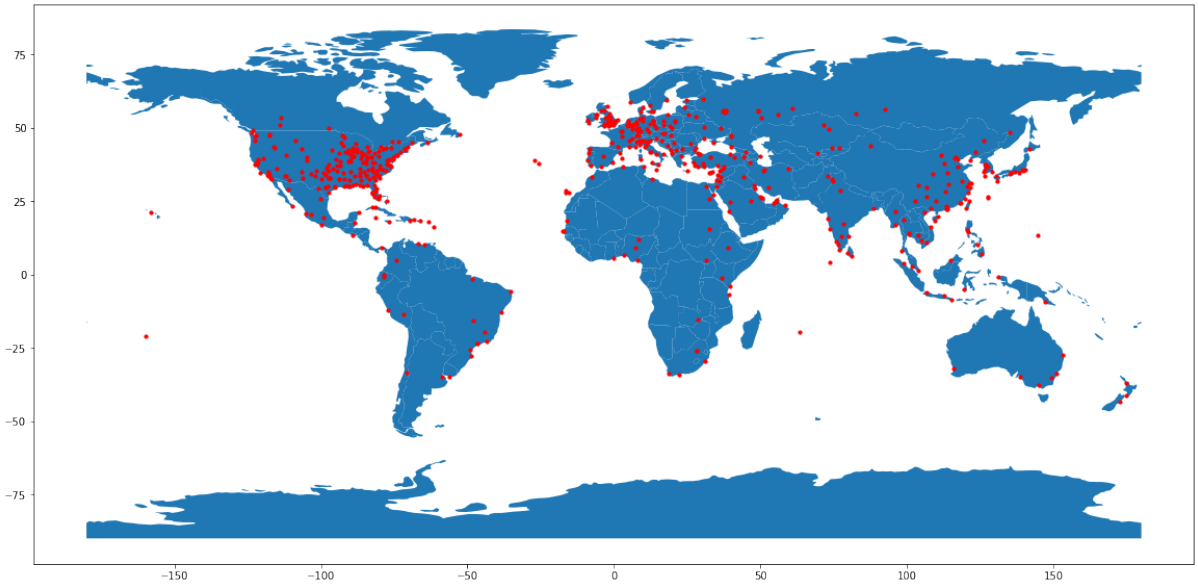

Figura 4.5: Mapa-mundi aeropuertos

De nuevo se ve como las zonas con mas riqueza son las que mayor cantidad de aeropuertos tienen, vemos como en la costa este de Estados Unidos es donde mayor densidad de aeropuertos hay, mientras que en zonas como América del Sur o África apenas hay un aeropuerto (de gran tamaño) por país.

4.2. Análisis de Flights Data

4.2.1. Salidas y llegadas de España

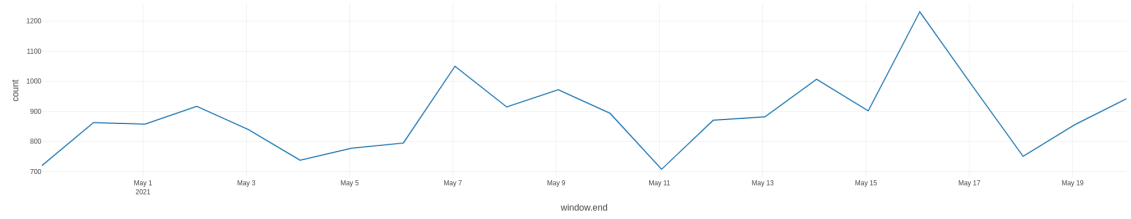

Figura 4.6: Salidas

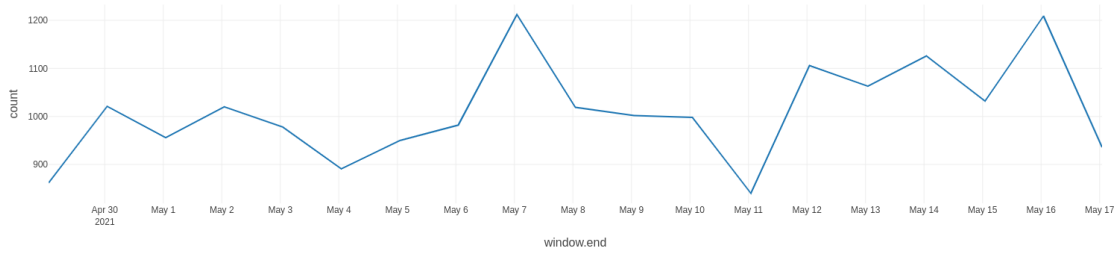

Figura 4.7: Llegadas

Se puede observar como durante los días de antes del fin del estado de alarma (1 al 8 de Mayo) el numero de salidas y llegadas es algo menor en comparación con los de después, esto es debido al fin de las restricciones de movilidad en distintas comunidades autónomas, principalmente Madrid, el cual es uno de los aeropuertos españoles con mas trafico. También se observa que los dos fines de semana después de que finalizase el estado de alarma hay grandes picos en ambas salidas y llegadas, especialmente el día 7 y el 16 (Viernes y Domingo respectivamente)

Vuelos totales por día

Analizando los días que mas trafico hay se ve claramente como los fines de semana, desde el viernes hasta el domingo es donde mas vuelos hay, siendo después entre semana una zona valle donde hay una cantidad menor de vuelos, aunque con tendencia al alza desde los primeros días hasta ahora. Posiblemente si se siguiese monitorizando estos datos (a través del streaming de estos datos) se podría ver como la media de vuelos seguirá subiendo (con posibles picos en verano por las vacaciones) hasta que llegue a estabilizarse.

4.2.2. Aeropuertos con mas trafico

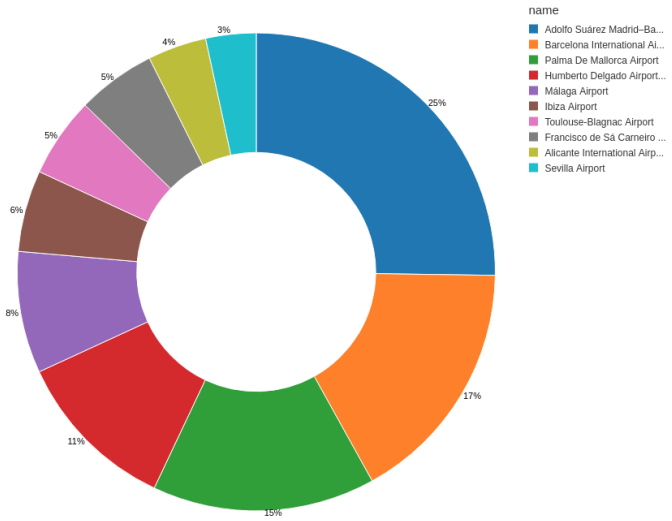

Figura 4.8: Salidas

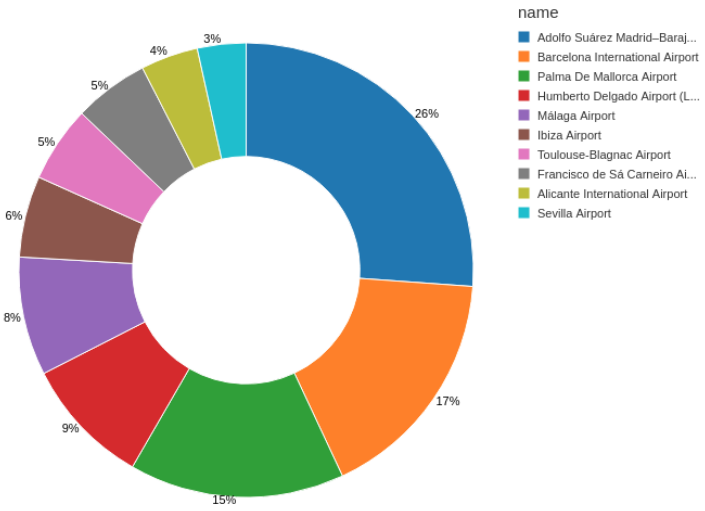

Figura 4.9: Llegadas

Vemos como ambos porcentajes de salidas/llegadas son muy parecidos teniendo en el top 3 los aeropuertos de Madrid,Barcelona y Palma.
El resto de destinos (un par de fuera de España, Lisboa y Toulouse) son en su mayoría destinos turísticos (además de ser destinos con playa), para estas visualizaciones se cogieron el top 10 de Aeropuertos con mas trafico, puesto que a partir de ahí el resto contaban con porcentajes insignificantes, menores incluso del 1 %

Aquí podemos ver también como los aeropuertos que mas trafico tienen (sin contar Madrid) donde mas trafico reciben son desde el aeropuerto de Madrid, esto se puede

num_flights	name	departure	arrival
234	Palma De Mallorca Airport	LEMD	LEPA
232	Barcelona International Airport	LEMD	LEBL
97	Humberto Delgado Airport (Lisbon Portela Airport)	LEMD	LPPT
94	Ibiza Airport	LEMD	LEIB
75	Málaga Airport	LEMD	LEMG
67	Adolfo Suárez MadridBarajas Airport	LEMD	LEMD
62	Alicante International Airport	LEMD	LEAL
57	Sevilla Airport	LEMD	LEZL

deber a escalas que hace las compañías (Iberia mayoritariamente en España) para dar un servicio desde otras partes del mundo.

4.2.3. Análisis vuelo IBE2031 (BCN-MAD)

Por ultimo se quiso visualizar la trayectoria de un vuelo, tanto a nivel de altitud como a nivel de posición. Para ello se escogió el vuelo del que obtuvimos mas posiciones y datos, el Iberia 2031, con ruta Barcelona-Madrid del día 2 de Mayo de 2021. Con salida a las 8:30PM y llegada sobre las 10:40PM.

Lo primero que tenemos es la altitud del vuelo respecto al tiempo, y vemos como la mayor parte del tiempo (mas de la mitad) esta en situaciones de ascenso o descenso, pues es un vuelo corto el cual dura apenas 2h, la altitud máxima que alcanza el vuelo es: 8846.82 metros. Este altitud puede no ser la altitud real del vuelo, pues es la baroaltitud, la cual su medida depende del tiempo que haga y de como el sensor capte la información del avión. En cambio con la geoaltitud tenemos un máximo de 9006.84 metros, esta posiblemente mas precisa debido a que su información la retransmite directamente el avión hacia tierra para que el resto pueden captar esta información.

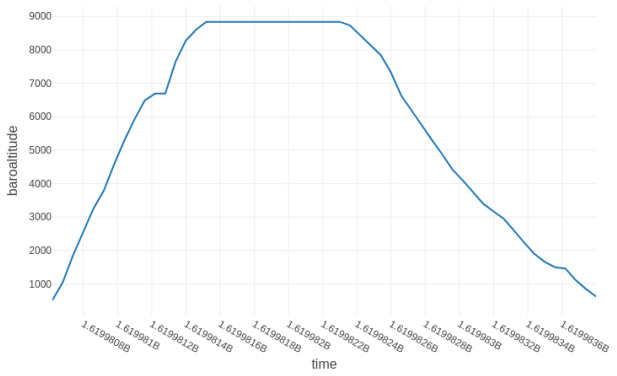

Figura 4.10: Altitud Barométrica

En esta figura tenemos la trayectoria del vuelo desde su despegue en BCN (Arriba derecha) hasta su llegada a MAD (Abajo izquierda) y vemos que durante ambas aproximaciones al aeropuerto la trayectoria da un giro para poder encarar la pista de aterrizaje de frente.

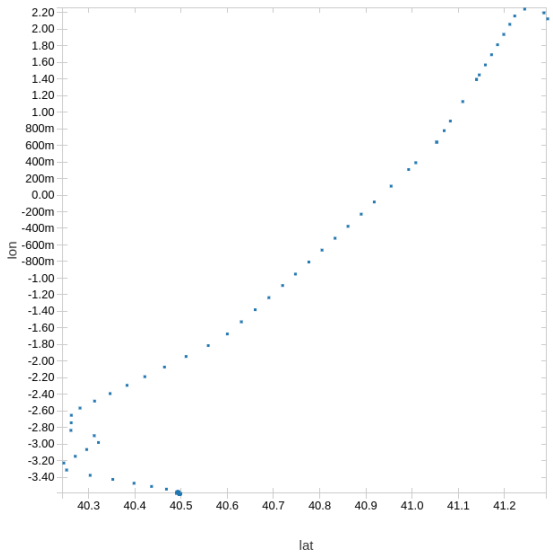

Figura 4.11: Trayectoria del vuelo

Por ultimo comparamos el trayecto que realizo a máxima altitud. Y vemos como durante este corto espacio de tiempo que estaba en su máxima altitud es también donde mas espacio recorrió, puesto que cuando no se varia la velocidad vertical (*vertical rate*), es donde mas velocidad punta alcanza una aeronave y por lo tanto cuando mas espacio cubre por segundo. El máximo de velocidad fue 215 m/s alcanzada a las 19:08, unos 30 minutos después de despegar.

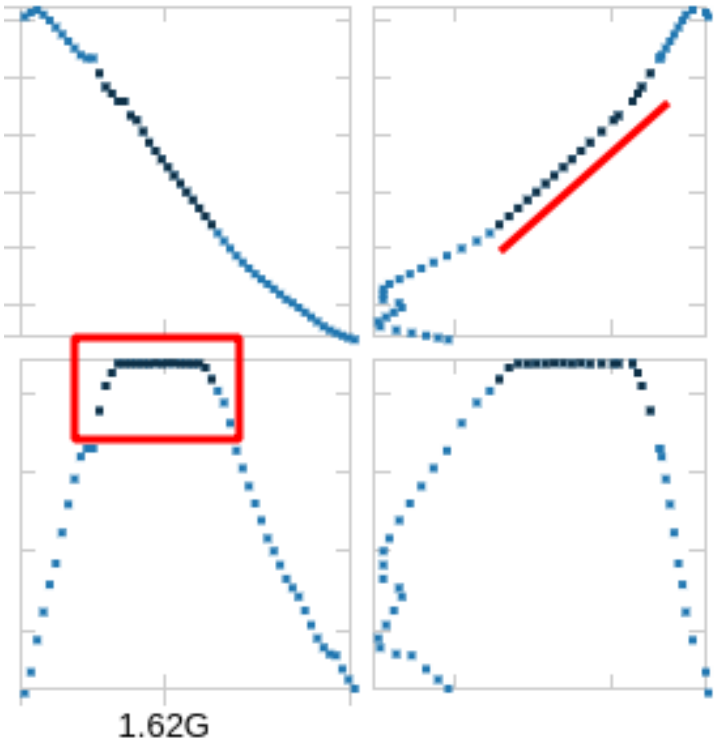

Figura 4.12: Altitud Barométrica

4.2.4. Streaming de Spark con StateVectors

Para obtener el streaming de datos de posición con spark, se simulo este streaming con ficheros, estos datos estaban agrupados por la ultima posición conocida y separados por un índice del 0 al 126 (total de posiciones que teníamos). Una vez obtenidos y procesados estos datos se procesaban con Spark para obtener datos de la posición agrupados por minuto. Así podíamos observar como Spark los iba procesando uno a uno y podíamos mostrar interactivamente la query, donde al principio se ve el trayecto realizado hasta el momento el cual era el despegue y al final observamos al repetir la query unos minutos después como el vuelo ya estaba encarando la pista de aterrizaje.

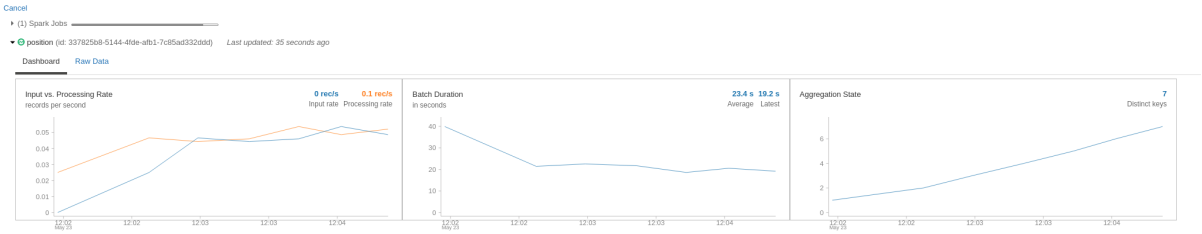

Figura 4.13: Spark Streaming

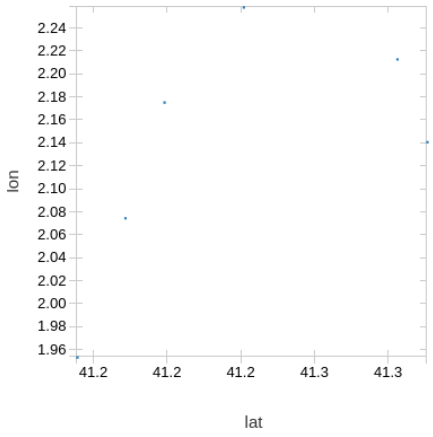

Figura 4.14: Streaming inicial

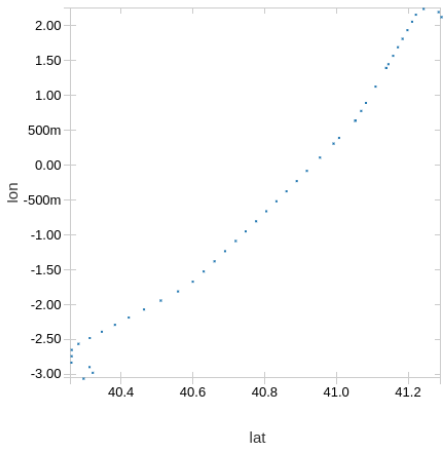

Figura 4.15: Streaming n minutos después

Esto seria interesante hacerlos obteniendo los datos en tiempo real a través de llamadas a la API y enviarlos a través de Kafka para que Spark los procesase según estos datos fuesen llegando, y así poder visualizar la trayectoria del vuelo (o cualquier otro dato, como altitud, velocidad, etc) en tiempo real según nos llegan estos datos.

5 Referencias

- https://github.com/GuillermoOvejeroSanchez/OpenSky_Min
- https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/8835461225836997/828574431330626/5451825131739270/latest.html