**Proyecto Final**

***Dashboard COVID***

***Integrantes del Grupo:***

Sergio José Barrios Martínez - Carné:19012765

Alvaro Andres Esquivel Gómez - Carné:11002822

Jonathan André De León Monzón - Carné:09001843

Marlon Samuel Gonzales Flores - Carné:20007175

***Parte 1: Estructura completa de Folder y archivos con solución del proyecto***

Se detallará la estructura completa de la solución subida en el repositorio de github, solo se dará una breve explicación del contenido de cada folder, además de remarcar las funciones de algunos archivos en específico.

*Nota: El funcionamiento de los contenedores y el dashboard se detallará posteriormente.*

* Carpeta *airflow*: Archivos para la creación del Contenedor de Airflow, con todas sus configuraciones
  + *config*
    - airflow.cfg
    - setup\_connections.py
  + *script*
    - *entrypoint.sh*
  + *Dockerfile*
  + *requirements.txt*
* Carpeta *dags*: Archivos para la creación de los DAGS de Airflow, con ellos se cargará la información de los archivos CSV a la base de datos MySQL.
  + *covid\_dag.py*
  + *covid\_deaths.py*
  + *covid\_recover.py*
* Carpeta *dashboard\_shiny*: Archivos para la creación del Contenedor de Airflow, con el código del dashboard a ejecutar.
  + *code*
    - *server.R*
    - *ui.R*
  + *Dockerfile*
* Carpeta *data*: Archivos CSV de la data que se procesará para cargar los datos del dashboard. Esta carpeta no es donde los DAGS leen la información, simplemente se dejan aquí para su almacenamiento y referencia.
  + *time\_series\_covid19\_confirmed\_global.csv*
  + *time\_series\_covid19\_deaths\_global.csv*
  + *time\_series\_covid19\_recovered\_global.csv*
* Carpeta *logs*: Carpeta donde el contenedor de shiny depositará los logs.
* Carpeta *monitor*: Carpeta donde los DAGS de airflow tomarán los archivos CSV para cargar a la base de datos.
* Carpeta *script\_dbs*: Archivos para la creación de las tablas de la Base de datos del Dashboard de COVID
  + *schema.sql*
* Archivo *Docker-compose.yml* : Docker-Compose del proyecto
* Archivo *Proyecto Final- Especificación Técnica.docx*: Documento con especificación Técnica del Proyecto.

***Parte 2: Contenedor Airflow***

Para el contenedor de airflow, se uso de base el contenedor creado para el Proyecto puckel/airflow (<https://github.com/puckel/docker-airflow>). Tomar en consideración:

* El archivo de configuración de airflow (airflow.cfg) es el mismo del repositorio.
* El Dockerfile tiene una pequeña modificación para la instalación de algunos paquetes de Python específicos para el proyecto (*requirements.txt*).
* Se crearon de forma automática las conexiones (Connections) que utilizaremos en los dags.
  + El archivo *setup\_connections.py* contiene el código en Python para la creación de las conexiones “airflow\_db” y “fs\_default”; además borra las demás conexiones que no utilizaremos.
  + Se modifica el archivo *entrypoint.sh,* colocando la ejecución del archivo con código de Python para la creación de las conexiones.

python3 ${AIRFLOW\_HOME}/setup\_connections.py

* + Se modifica el Dockerfile para copiar el archivo *setup\_connections.py* a la carpeta raíz de airflow.

COPY config/setup\_connections.py ${*AIRFLOW\_USER\_HOME*}/setup\_connections.py

* Para la base de datos que usará el contenedor, se usará postgreSQL, que se creará y configurará desde el *Docker-compose.yml*.

***Parte 3: Contenedor Shiny Server (Dashboard)***

Para el contenedor de shiny Server, se usará de base el contenedor creado para el Proyecto rocker/shiny (<https://hub.docker.com/r/rocker/shiny>). Tomar en consideración:

* Se creo un *Dockerfile* específico para este proyecto, ya que se necesitaban colocar algunos requisitos de librerías para nuestro dashboard.

FROM rocker/shiny:latest  
  
RUN apt-get update -qq && apt-get -y --no-install-recommends install \  
 libmysqlclient-dev \  
 libxml2  
  
RUN R -e "install.packages(c('RMySQL','dplyr','shinydashboard','leaflet','RColorBrewer','tidyverse','ggplot2','plotly','lubridate','shinyWidgets'))"

* El código fuente del dashboard (*server.R, Ui.R*) se copiará a la carpeta que necesita Shiny server desde el *Docker-compose.yml*.
* Para la base de datos que usará el contenedor, se usará MySQL, que se creará y configurará desde el *Docker-compose.yml*.

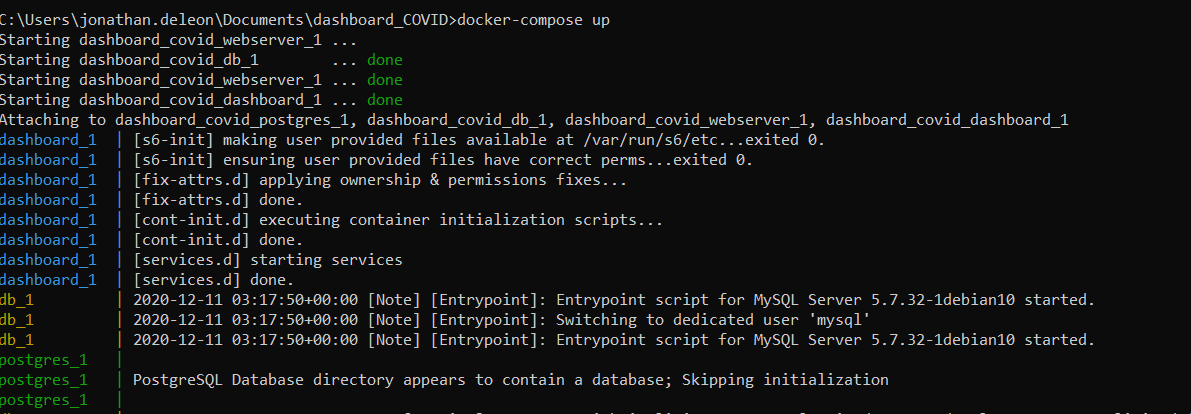
***Parte 4: Docker-compose***

El Docker-compose orquestará el contenedor de airflow con su base de datos (postgreSQL), shiny Server y la base de datos que utilizará el dashboard (MySQL), de la siguiente forma:

version: '3.7'  
services:  
 postgres:  
 image: postgres:9.6  
 environment:  
 - POSTGRES\_USER=airflow  
 - POSTGRES\_PASSWORD=airflow  
 - POSTGRES\_DB=airflow  
 logging:  
 options:  
 max-size: 10m  
 max-file: "3"  
  
 webserver:  
 build: ./airflow  
 restart: always  
 depends\_on:  
 - postgres  
 environment:  
 - LOAD\_EX=n  
 - EXECUTOR=Local  
 logging:  
 options:  
 max-size: 10m  
 max-file: "3"  
 volumes:  
 - ./dags:/usr/local/airflow/dags  
 - ./monitor:/home/airflow/monitor  
 ports:  
 - "8080:8080"  
 command: webserver  
 healthcheck:  
 test: ["CMD-SHELL", "[ -f /usr/local/airflow/airflow-webserver.pid ]"]  
 interval: 30s  
 timeout: 30s  
 retries: 3  
  
 db:  
 image: mysql:5.7  
 volumes:  
 - ./script\_dbs/schema.sql:/docker-entrypoint-initdb.d/1.sql  
 restart: always  
 ports:  
 - 3306:3306  
 environment:  
 MYSQL\_ROOT\_PASSWORD: root123  
 MYSQL\_DATABASE: covid  
 MYSQL\_USER: covid  
 MYSQL\_PASSWORD: covid123  
 dashboard:  
 build:  
 ./dashboard\_shiny  
 depends\_on:  
 - db  
 ports:  
 - 3838:3838  
 volumes:  
 - ./dashboard\_shiny/code:/srv/shiny-server/  
 - ./logs:/var/log/shiny-server/

*\*\* Nota: Se dejará archive yml en el repositorio de las fuentes del programa (Github).*

* Se levanta el Docker-compose por medio del comando “*docker-compose up*”.

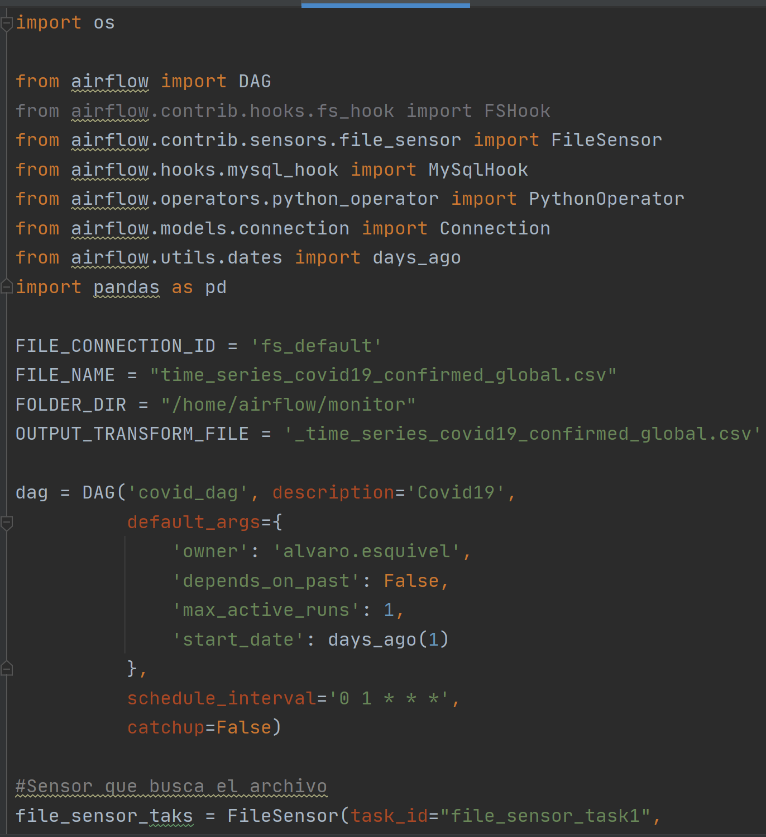


*\*\* Nota: No se coloca todo el texto generado al ejecutar el comando.*

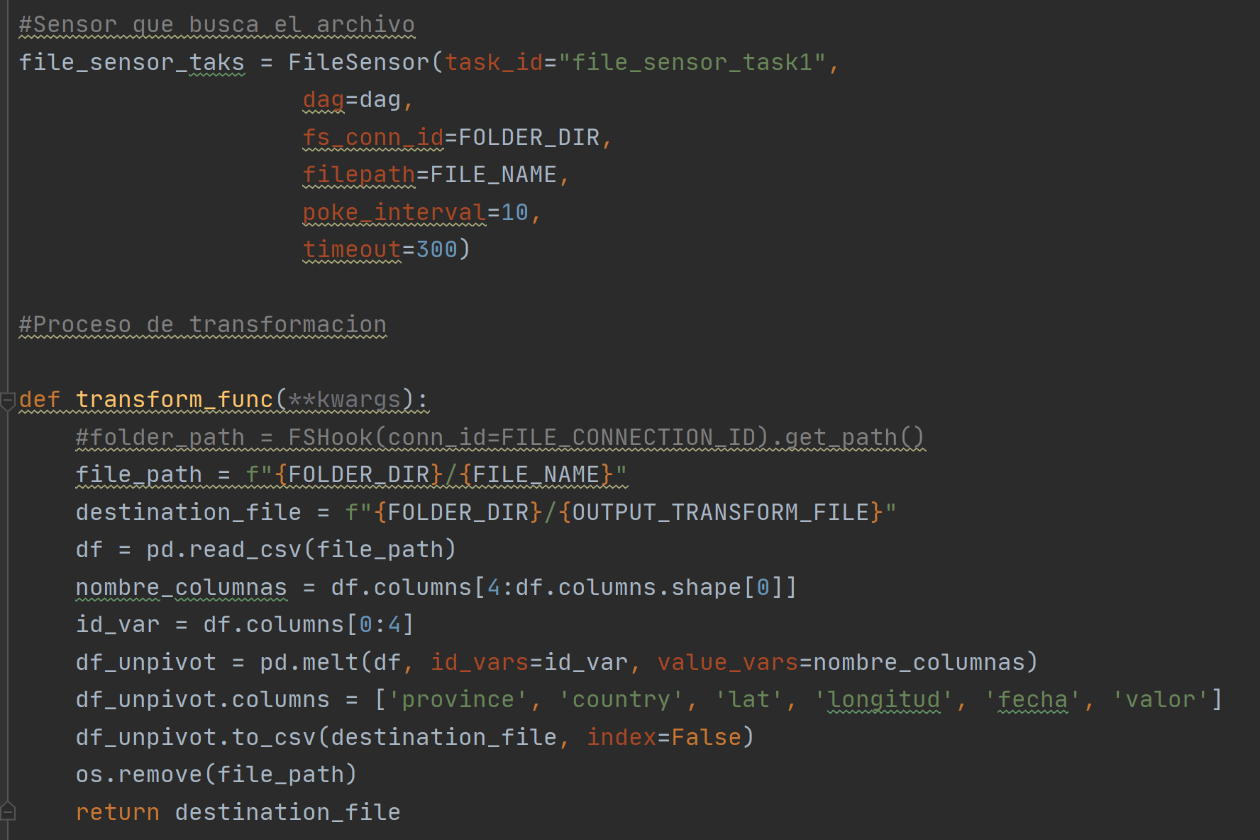
***Parte 5: DAGs en Airflow***

Con la finalidad de tener la posibilidad de crear diferentes programaciones para los archivos, se decido crear 3 DAGs diferentes para la extracción, transformación y carga de los archivos a la base de datos.

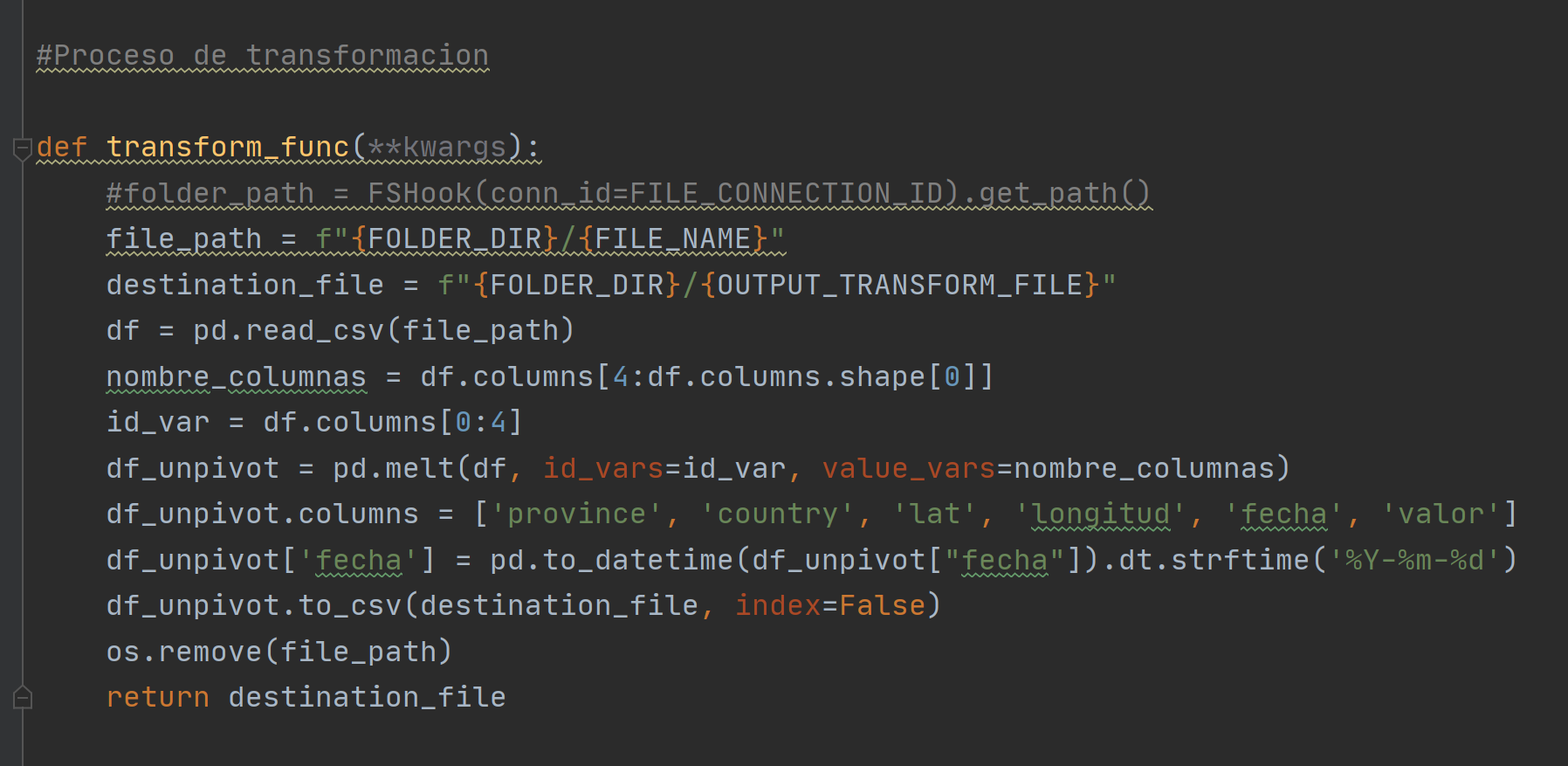
* Primero se definieron como constantes globales el nombre la configuración de conexión a la carpeta del archivo, el nombre de los archivos, la ubicación dentro del servidor de Airflow y el nombre de salida del archivo.



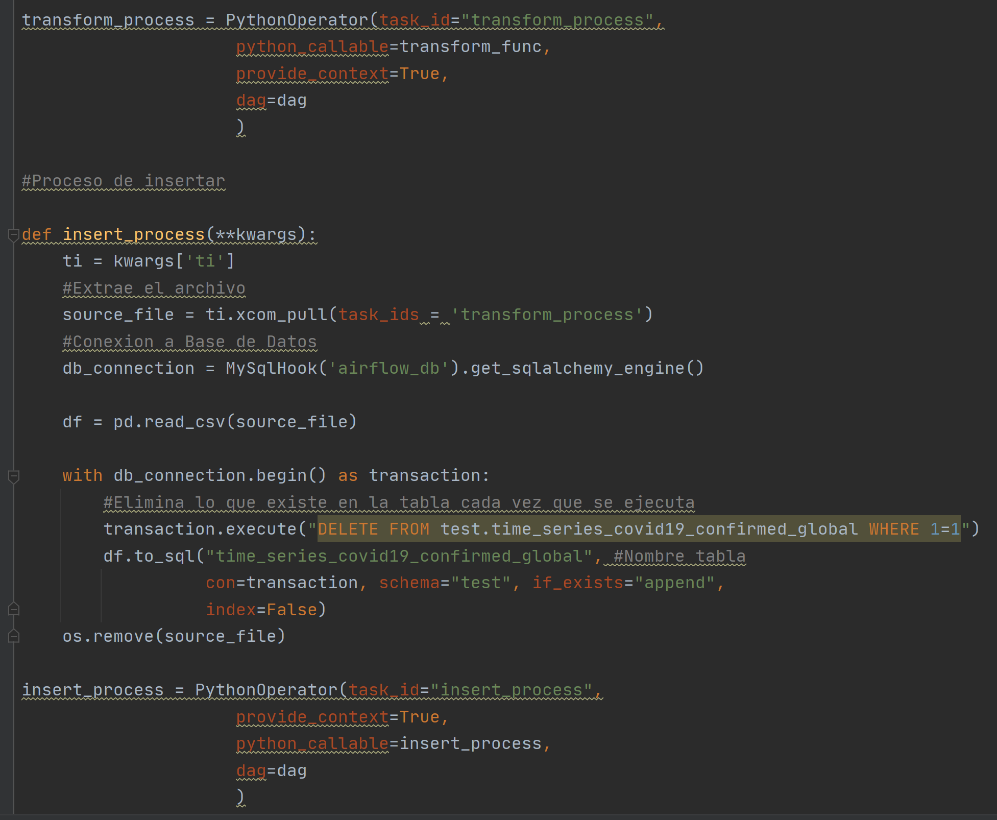
* Luego de las anteriores configuraciones, se configura la función de DAG con los valores utilizados para identificar el DAG y la programación de va a tener (ver imagen anterior), en este ejercicio como será ejecutado manualmente estos valores son únicamente ilustrativos.
* La configuración siguiente es la del sensor del archivo, el cual estará buscando la llegada del archivo definido cada 10 segundos en la ruta y con el nombre configurado.



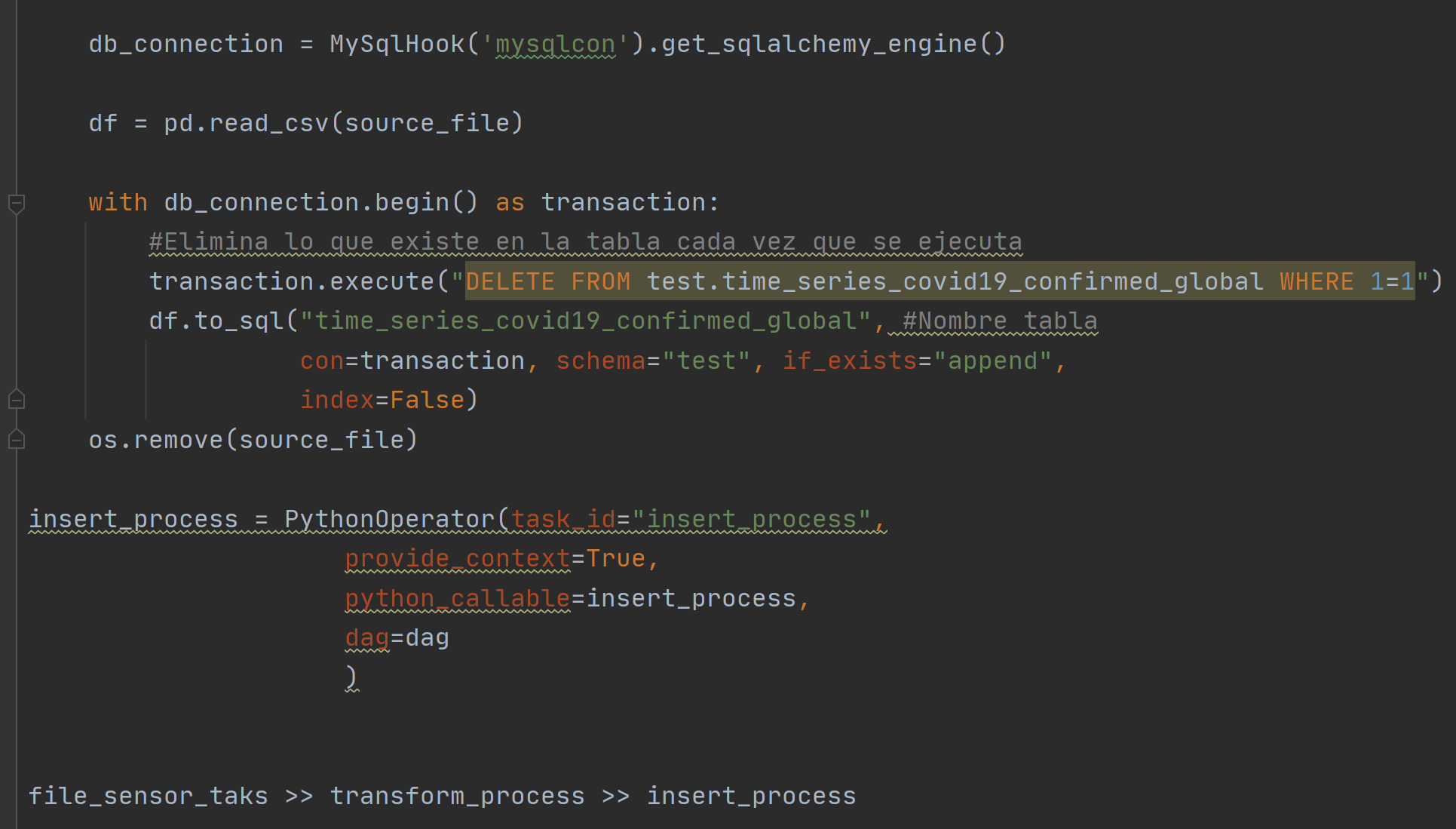
* Luego está la función de transformación, que toma todos los valores el csv, y transforma su formato para que nuevas observaciones se almacenen con filas nuevas en lugar de columnas como originalmente estaba (unpivot). También cambia el nombre de las columnas resultantes y por último cambia el formato de fecha de tipo string a tipo date. Al finalizar los cambios guarda el dataframe en un csv para que sea cargado a la base de datos.



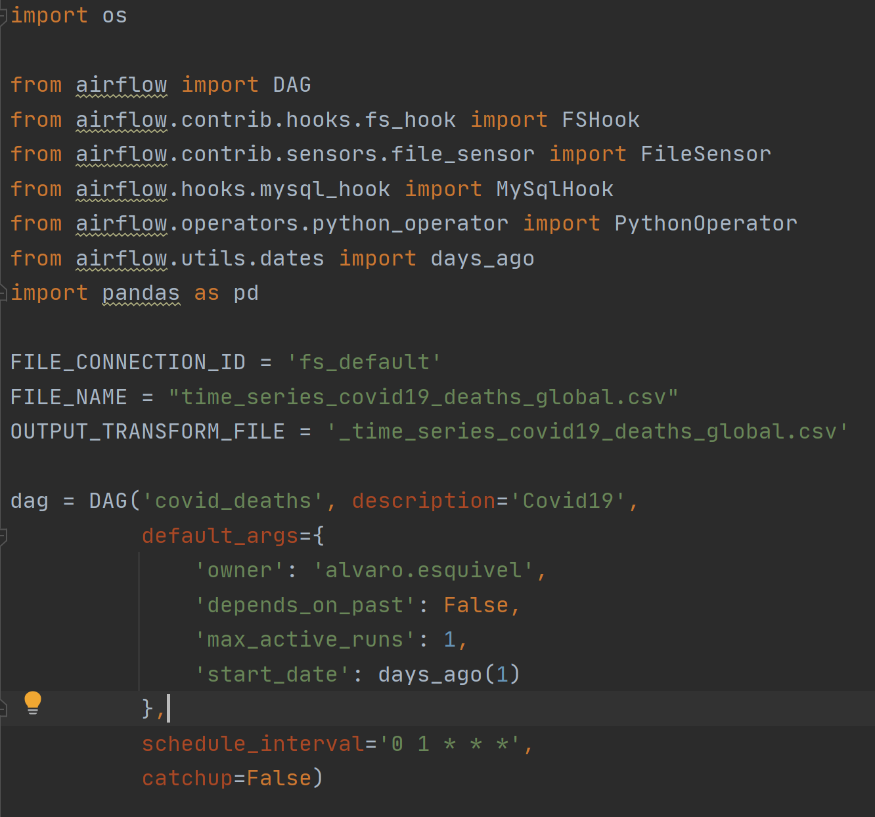
* La configuración para insertar el csv en la base de datos se describe a continuación, básicamente se utiliza la conexión creada al memento de levantar Airflow llamada “airflow\_db”, y con la función to\_sql se carga el archivo especificado en la tabla de MySQL, en el esquema también indicado. Luego, mediante la función remove del sistema, se elimina el archivo de la ruta original.
* Tanto la transformación como la inserción, se realiza mediante PythonOperators, lo cual define procesos en el DAG para posteriormente configurar el orden de ejecución es estos.

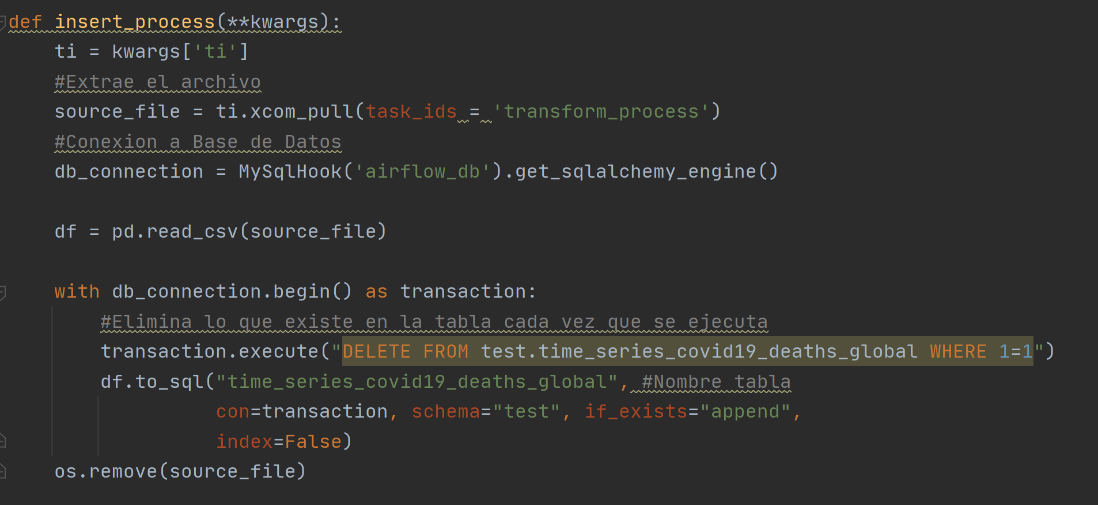


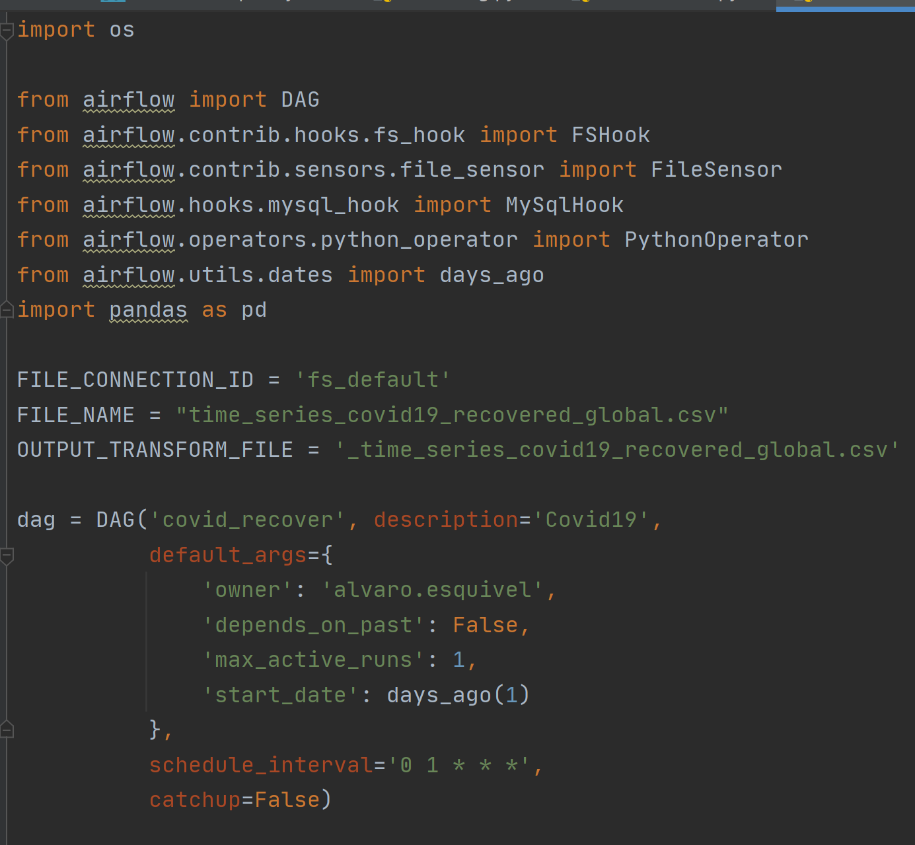
* Por último, como mencionamos anteriormente, se define el orden de ejecución, primero el proceso de extracción de archivo, luego el proceso de transformación y por último, el proceso de inserción a la base de datos.



* Los siguientes DAGs utilizan básicamente la misma configuración, por lo que no se explicaran a detalle. El único cambio, es el nombre del archivo esperado en el DAG y el nombre de la tabla en la que se almacena la información. Los cambios son los siguientes:

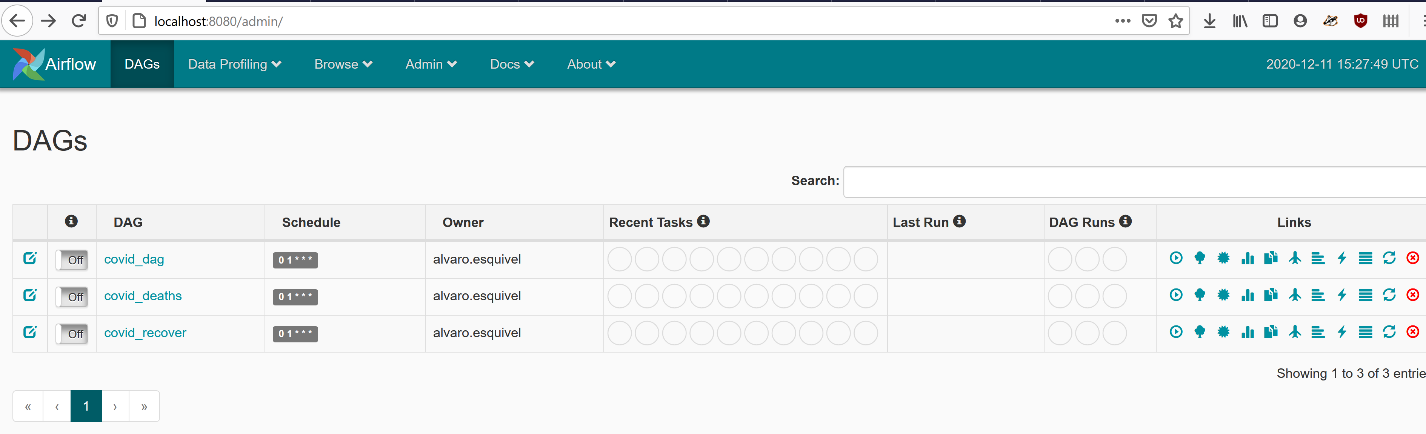




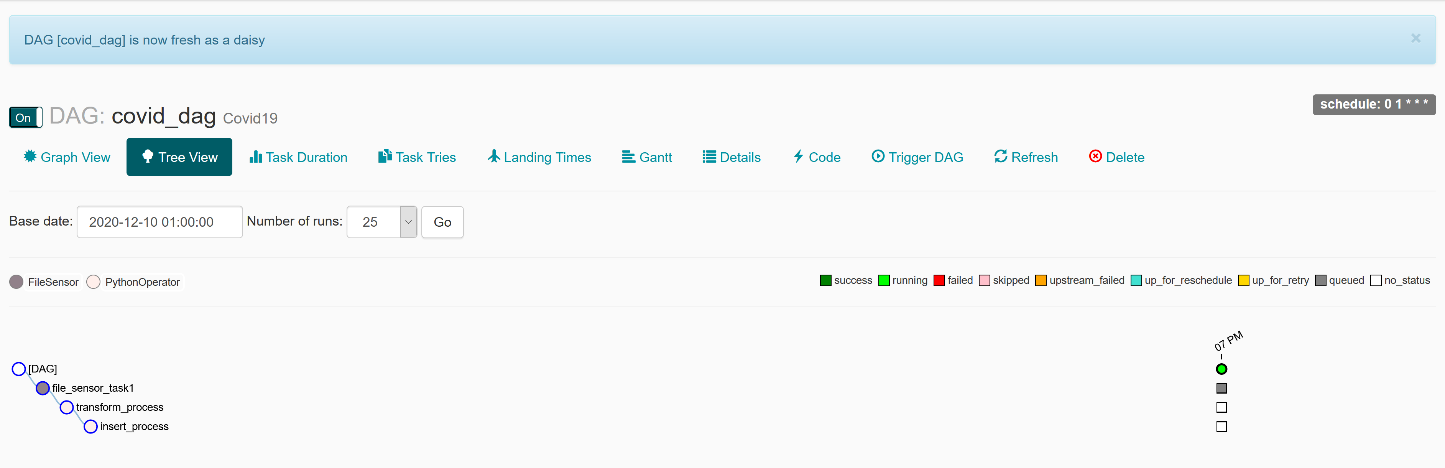




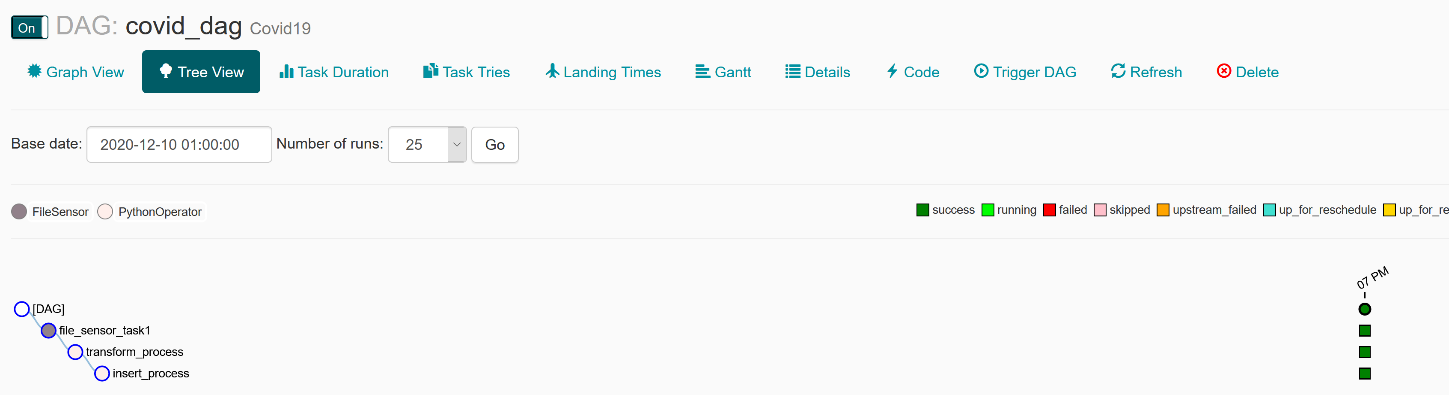
* En la UI de Airflow los DAGs son cargados automáticamente, como se puede ver en la imagen están apagados al inicio.



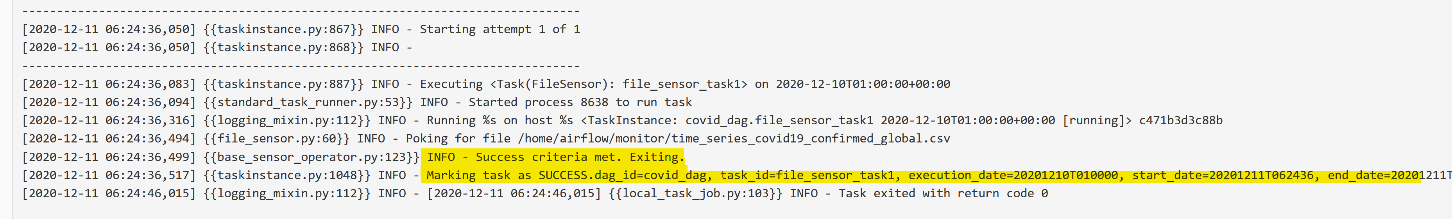
* Es necesario encender el DAG para que empiece el proceso de ejecución. Los colores indican el estado del proceso.



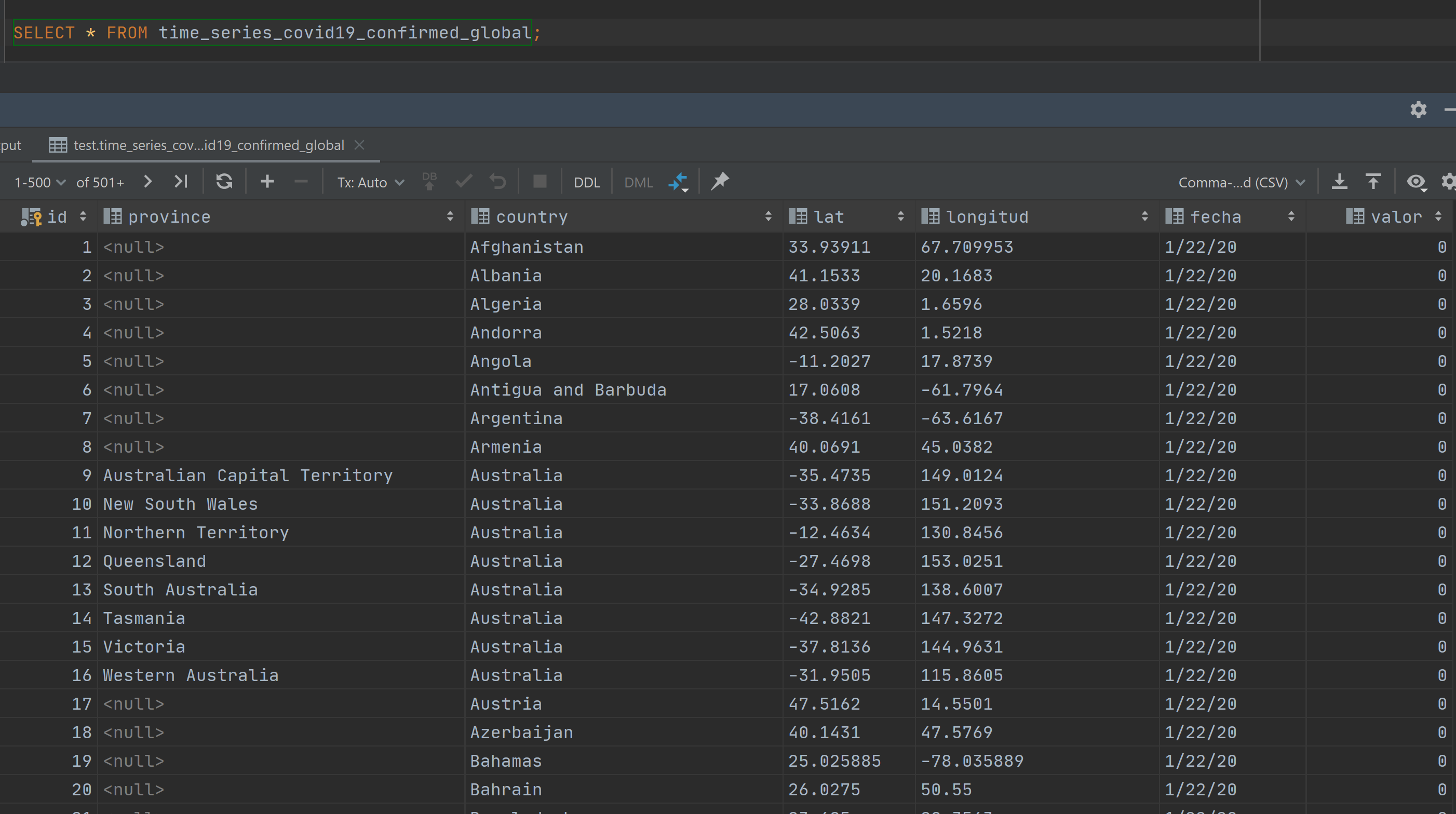
* Al finalizar la ejecución, se muestran en color verde oscuro, como en la siguiente imagen:



* En el log del DAG, se puede visualizar cualquier error ocurrido en la ejecución del DAG, en la siguiente imagen se muestra que el archivo cumplió con el criterio de nombre, por lo que fue procesado y subido a la base de datos con éxito.



* En la imagen siguiente, podemos ver la información transformada y carada en la base de datos de MySQL.

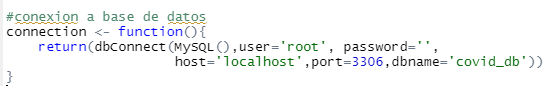


***Parte 6: Shiny Dashboard***

Se describirán las secciones importantes del dashboard shiny construido para el proyecto.

*Conexión a base de datos desde RStudio*

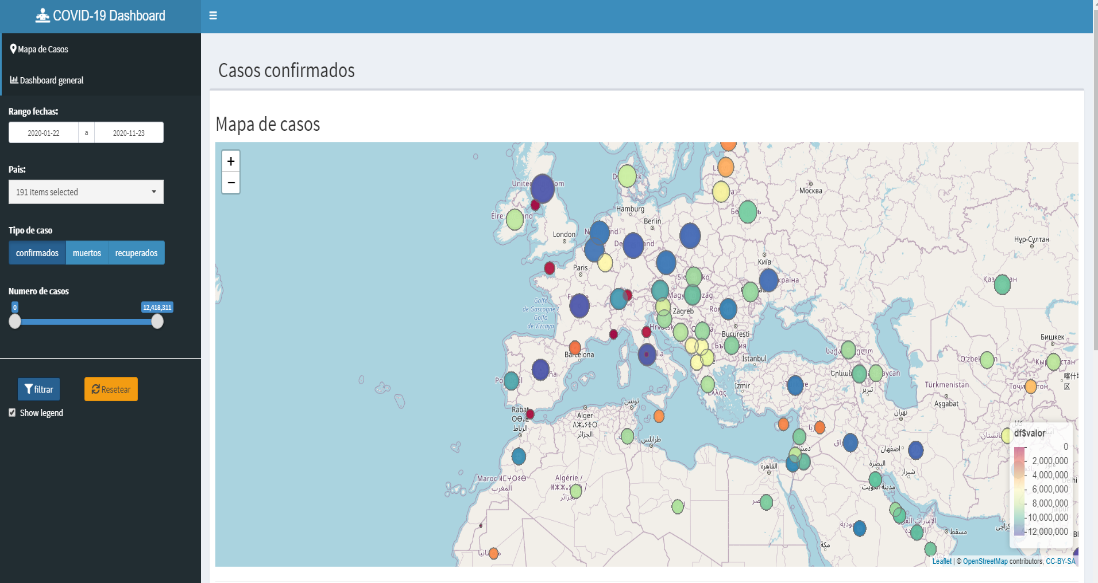
**RMySQL:** es una interfaz de base de datos y un controlador MySQL para R. E



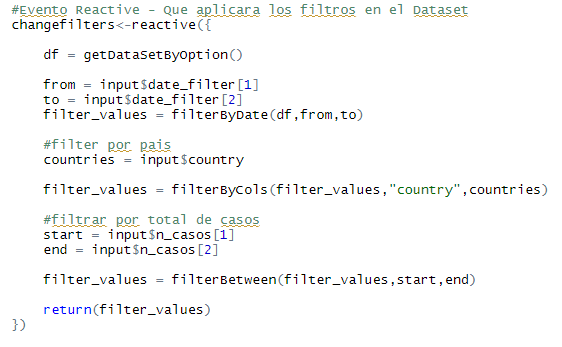
*Paquetes principales*

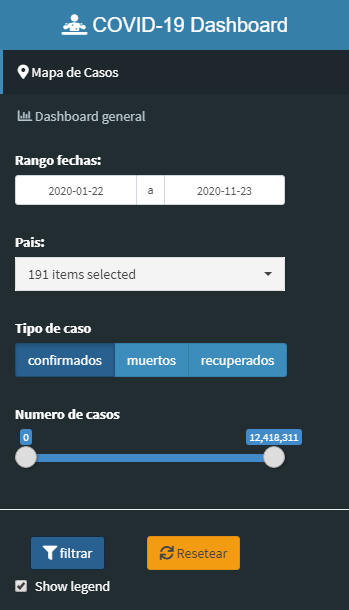
* **Tidyverse:**  es un conjunto de paquetes en R diseñados para ciencia de datos.
* **Plotly:** La biblioteca de gráficos R de Plotly crea gráficos interactivos con calidad de publicación.
* **ShinyDashboard:** facilita el uso de [Shiny](http://shiny.rstudio.com/) para crear cuadros de mando como lo siguiente.

*Gráficas y controles principales.*

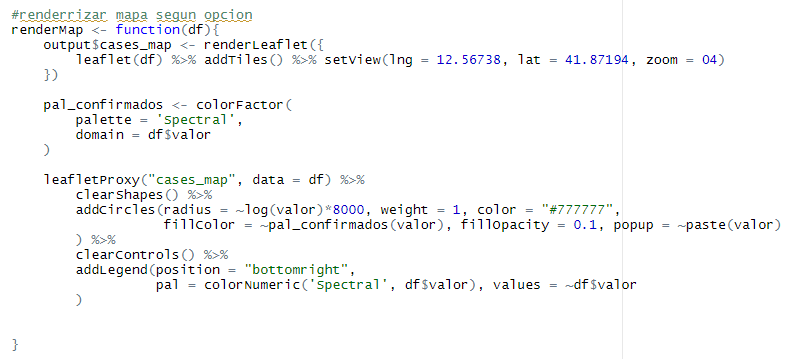


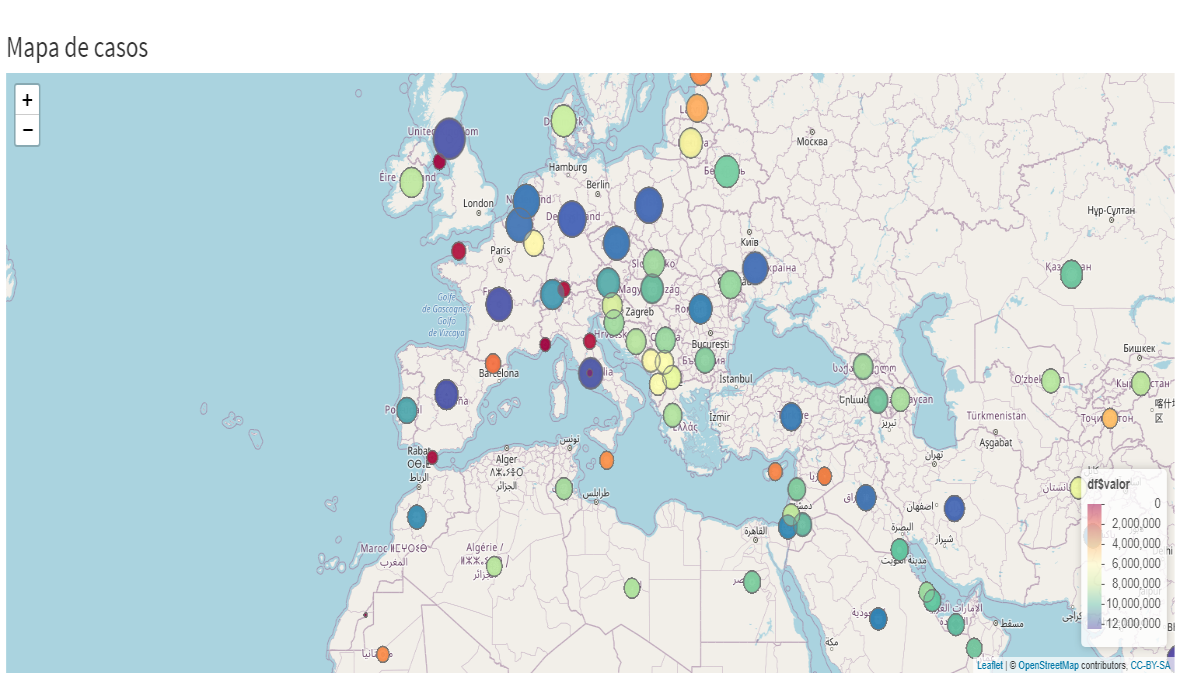
*Filtros*





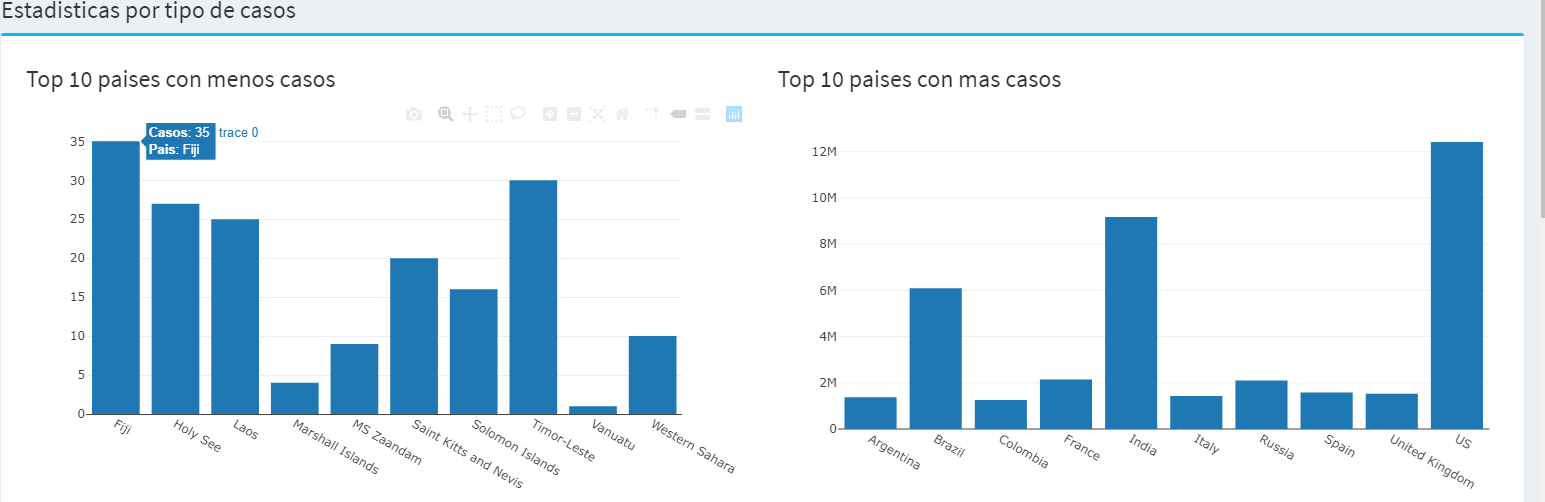
*Mapa interactivo de casos a nivel mundial*





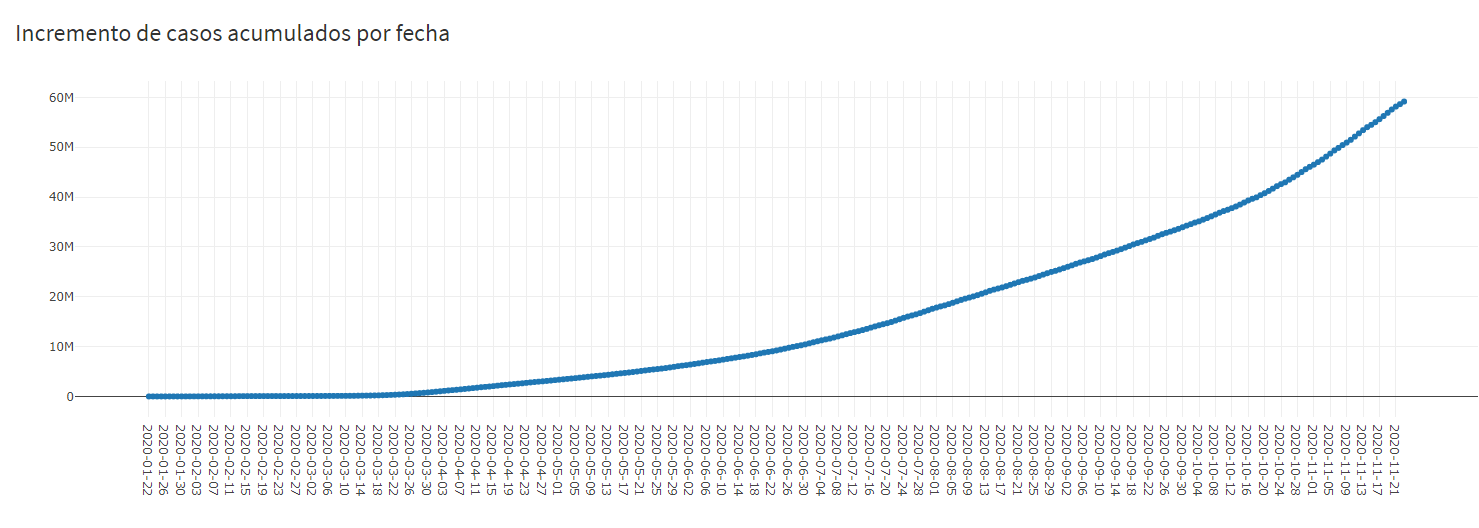
*Top 10 países con más y menos casos*





*Tendencia Acumulada por fecha a nivel mundial*





*Mapa interactivo acumulado por país y fecha.*



