Ejercicio 5

- Objetivo: Analizar la evolución de casos de COVID-19 por país utilizando un conjunto de datos real descargado de internet.
- Planteamiento: Una organización de salud global necesita analizar la propagación del COVID-19 en diferentes países para informar sus políticas y recomendaciones. Utilizarán el conjunto de datos de casos confirmados de COVID-19 proporcionado por el Centro de Ciencia e Ingeniería de Sistemas de la Universidad Johns Hopkins. El dataset contiene información diaria sobre los casos confirmados de COVID-19 para diferentes países y regiones.
- Se requiere:
  - Descargar y procesar el conjunto de datos más reciente.
  - Transformar los datos de formato ancho a largo para facilitar el análisis temporal.
  - Calcular los nuevos casos diarios por país.
  - Identificar los 10 países con más casos acumulados.
  - Calcular la media móvil de 7 días de nuevos casos para los top 10 países.
  - Almacenar los resultados en un formato eficiente para futuras consultas.
- Ayuda:
  - Utiliza las funciones de Spark SQL para realizar transformaciones y agregaciones.
  - Considera usar Window functions para cálculos como la media móvil.
  - Aprovecha las capacidades de particionamiento de Spark al escribir los resultados.
- Dataset: "<https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv>

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
import org.apache.spark.sql.expressions.Window
import java.net.URL
import java.nio.file.{Files, Paths, StandardCopyOption}

object Ejercicio5 {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("Ejercicio5")
      .master("local[*]")
      .getOrCreate()

    import spark.implicits._

    // URL del dataset
    val url = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv"

    // Descargar el archivo
    val file = Paths.get("covid_data.csv")
    if (!Files.exists(file)) {
      println("Descargando archivo...")
      val connection = new URL(url).openConnection()
```

```
      val in = connection.getInputStream
      Files.copy(in, file, StandardCopyOption.REPLACE_EXISTING)
      in.close()
      println("Archivo descargado.")
    }

    // Leer el CSV
    val covidData = spark.read
      .option("header", "true")
      .option("inferSchema", "true")
      .csv(file.toString)

    // Transformar los datos de formato ancho a largo
    val dateCols = covidData.columns.drop(4)
    val longFormatData = covidData
      .select(
        col("Country/Region"),
        explode(array(dateCols.map(c => struct(lit(c).as("date"),
col(c).as("confirmed_cases"))): _*)).as("tmp")
      )
      .select(
        col("Country/Region").as("country"),
        to_date(col("tmp.date"), "M/d/yy").as("date"),
        col("tmp.confirmed_cases").cast("int").as("confirmed_cases")
      )

    // Calcular nuevos casos diarios
    val dailyCases = longFormatData
      .withColumn("previous_day_cases",
        lag("confirmed_cases",
1).over(Window.partitionBy("country").orderBy("date")))
      .withColumn("new_cases",
        when(col("confirmed_cases") - col("previous_day_cases") < 0, 0)
          .otherwise(col("confirmed_cases") - col("previous_day_cases")))

    // Analizar los 10 países con más casos acumulados
    val topCountries = dailyCases
      .groupBy("country")
      .agg(max("confirmed_cases").as("total_cases"))
      .orderBy(col("total_cases").desc)
      .limit(10)

    println("Top 10 países con más casos acumulados:")
    topCountries.show()

    // Calcular la media móvil de 7 días de nuevos casos para los top 10 países
    val movingAverage = dailyCases
      .join(topCountries, Seq("country"))
      .withColumn("7_day_average",

avg("new_cases").over(Window.partitionBy("country").orderBy("date").rowsBetween(-
6, 0)))
      .select("country", "date", "new_cases", "7_day_average")
      .orderBy("country", "date")

    println("Media móvil de 7 días de nuevos casos para los top 10 países:")
    movingAverage.show()

    // Guardar resultados en formato Parquet
```

```
movingAverage.write.mode("overwrite").partitionBy("country").parquet("covid_analy
sis_results")

    spark.stop()
  }
}
```

Explicación detallada del código:

1. **Importaciones:**
   - `org.apache.spark.sql.SparkSession`: Clase principal para interactuar con Spark SQL. Permite crear DataFrames y ejecutar consultas SQL.
   - `org.apache.spark.sql.functions._`: Importa todas las funciones predefinidas de Spark SQL (como `col`, `explode`, `struct`, `lit`, `to_date`, `lag`, `when`, `max`, `avg`). El _ es un comodín para importar todos los miembros de la clase `functions`.
   - `org.apache.spark.sql.expressions.Window`: Proporciona funcionalidades para definir ventanas de datos sobre las cuales se pueden aplicar funciones analíticas (como `lag` y `avg` con `over`).
   - `java.net.URL`: Clase para trabajar con URLs (para descargar el dataset).
   - `java.nio.file.{Files, Paths, StandardCopyOption}`: Clases para operaciones de archivos y directorios (para descargar y guardar el dataset localmente).

2. **Objeto `Ejercicio5`:**
   - Define un objeto Scala llamado `Ejercicio5`, que contendrá la lógica principal de la aplicación.

3. **Método `main`:**
   - Es el punto de entrada de la aplicación. Recibe un array de strings (`args`) como argumentos de línea de comandos (aunque en este caso no se utilizan).

4. **Inicialización de `SparkSession`:**
   - `val spark = SparkSession.builder().appName("Ejercicio5").master("local[*]").getOrCreate()`:
     - `SparkSession.builder()`: Crea un constructor para una sesión de Spark.
     - `.appName("Ejercicio5")`: Establece un nombre para la aplicación Spark (útil para monitorización).
     - `.master("local[*]")`: Especifica el modo de ejecución de Spark. `"local[*]"` indica que Spark se ejecutará en modo local utilizando todos los núcleos disponibles de la máquina.
     - `.getOrCreate()`: Intenta obtener una sesión de Spark existente. Si no existe ninguna, crea una nueva.

5. **Importación de `spark.implicits._`:**
   - `import spark.implicits._`: Importa implicits que permiten realizar conversiones automáticas entre objetos Scala estándar (como `List`, `Seq`) y DataFrames de Spark, así como proporcionar métodos de extensión convenientes (como `.toDF()`).

6. **Definición de la URL del Dataset:**
   - `val url = "..."`: Define una variable `url` que contiene la dirección web del archivo CSV con los datos de casos confirmados de COVID-19.

7. **Descarga del Archivo:**
   - Se define la ruta local del archivo (`covid_data.csv`).
   - Se verifica si el archivo ya existe localmente usando `Files.exists(file)`.
   - Si el archivo no existe:
     - Se imprime un mensaje indicando la descarga.
     - Se abre una conexión a la URL utilizando `new URL(url).openConnection()`.
     - Se obtiene un flujo de entrada (`InputStream`) desde la conexión.
     - Se copia el contenido del flujo de entrada al archivo local utilizando `Files.copy()`, reemplazando el archivo si ya existiera (`StandardCopyOption.REPLACE_EXISTING`).
     - Se cierra el flujo de entrada.
     - Se imprime un mensaje indicando que la descarga se completó.

8. **Lectura del Archivo CSV:**
   - `val covidData = spark.read.option("header", "true").option("inferSchema", "true").csv(file.toString)`:
     - `spark.read`: Obtiene un objeto `DataFrameReader` para leer datos.
     - `.option("header", "true")`: Especifica que la primera línea del archivo CSV contiene los nombres de las columnas.
     - `.option("inferSchema", "true")`: Indica a Spark que intente inferir automáticamente los tipos de datos de cada columna basándose en su contenido.
     - `.csv(file.toString)`: Especifica la ruta del archivo CSV que se va a leer. El resultado es un DataFrame llamado `covidData`.

9. **Transformación de Formato Ancho a Largo:**
   - `val dateCols = covidData.columns.drop(4)`: Obtiene los nombres de todas las columnas del DataFrame `covidData` y luego elimina las primeras 4 columnas (que suelen ser información geográfica como "Province/State", "Country/Region", "Lat",

"Long"). Las columnas restantes se asumen que son las fechas con los conteos de casos.

- `val longFormatData = ...`: Realiza la transformación de formato ancho a largo (unpivot).
  - `.select(col("Country/Region"), explode(array(dateCols.map(c => struct(lit(c).as("date"), col(c).as("confirmed_cases"))): _*)).as("tmp"))`:
    - `col("Country/Region")`: Selecciona la columna del país/región.
    - `dateCols.map(c => struct(lit(c).as("date"), col(c).as("confirmed_cases")))`: Para cada nombre de columna de fecha en `dateCols`, crea una estructura (`struct`) con dos campos: "date" (con el valor literal del nombre de la columna) y "confirmed_cases" (con el valor de esa columna).
    - `array(...)`: Crea un array de estas estructuras.
    - `explode(...)`: Toma el array de estructuras y crea una nueva fila para cada elemento del array. La columna resultante se llama "tmp" (por el `.as("tmp")`).
  - `.select(col("Country/Region").as("country"), to_date(col("tmp.date"), "M/d/yy").as("date"), col("tmp.confirmed_cases").cast("int").as("confirmed_cases"))`:
    - Selecciona las columnas necesarias del DataFrame intermedio "tmp".
    - Renombra "Country/Region" a "country".
    - Convierte la columna de fecha (que es un string) al tipo `Date` utilizando `to_date`, especificando el formato original de la fecha ("M/d/yy").
    - Convierte la columna de casos confirmados al tipo `Integer` utilizando `.cast("int")` y la renombra a "confirmed_cases".

10. **Cálculo de Nuevos Casos Diarios:**
    - `val dailyCases = ...`: Crea un nuevo DataFrame `dailyCases` añadiendo una columna con los nuevos casos diarios.
      - `.withColumn("previous_day_cases", lag("confirmed_cases", 1).over(Window.partitionBy("country").orderBy("date")))`:
        - Añade una nueva columna llamada "previous_day_cases".
        - Utiliza la función `lag()` para obtener el valor de la columna "confirmed_cases" de la fila anterior.
        - `over(Window.partitionBy("country").orderBy("date"))`: Define una ventana de datos. Los datos se particionan por país (`partitionBy("country")`) y se ordenan por fecha dentro de cada partición (`orderBy("date")`). El `lag(..., 1)` mira una fila hacia atrás dentro de esta ventana.
      - `.withColumn("new_cases", when(col("confirmed_cases") - col("previous_day_cases") < 0, 0).otherwise(col("confirmed_cases") - col("previous_day_cases")))`:
        - Añade una nueva columna llamada "new_cases".
        - Calcula la diferencia entre los casos confirmados del día actual y los del día anterior.

        - Utiliza la función `when()` para manejar posibles inconsistencias en los datos (donde los casos confirmados podrían disminuir). Si la diferencia es negativa, establece los nuevos casos en 0; de lo contrario, utiliza la diferencia calculada.

11. **Análisis de los 10 Países con Más Casos Acumulados:**
    - `val topCountries = ...`: Crea un DataFrame `topCountries` con los 10 países con el mayor número total de casos confirmados.
        - `.groupBy("country")`: Agrupa el DataFrame `dailyCases` por la columna "country".
        - `.agg(max("confirmed_cases").as("total_cases"))`: Para cada grupo (país), calcula el valor máximo de la columna "confirmed_cases" (que representa el total acumulado hasta esa fecha) y lo nombra "total_cases".
        - `.orderBy(col("total_cases").desc)`: Ordena el DataFrame resultante por la columna "total_cases" en orden descendente.
        - `.limit(10)`: Limita el resultado a las 10 primeras filas (los 10 países con más casos).
    - `println("Top 10 países con más casos acumulados:")` y `topCountries.show()`: Imprime un encabezado y muestra el DataFrame `topCountries` en la consola.

12. **Cálculo de la Media Móvil de 7 Días:**
    - `val movingAverage = ...`: Crea un DataFrame `movingAverage` con la media móvil de 7 días de nuevos casos para los 10 principales países.
        - `.join(topCountries, Seq("country"))`: Realiza una unión (inner join por defecto) entre el DataFrame `dailyCases` y `topCountries` utilizando la columna "country" como clave de unión. Esto filtra `dailyCases` para incluir solo los datos de los 10 principales países.
        - `.withColumn("7_day_average", avg("new_cases").over(Window.partitionBy("country").orderBy("date").rowsBetween(-6, 0)))`:
            - Añade una nueva columna llamada "7_day_average".
            - Utiliza la función `avg()` para calcular la media de la columna "new_cases".
            - `over(Window.partitionBy("country").orderBy("date").rowsBetween(-6, 0))`: Define una ventana de datos. Los datos se particionan por país y se ordenan por fecha. `rowsBetween(-6, 0)` especifica que la ventana incluye la fila actual y las 6 filas anteriores (un total de 7 días).
        - `.select("country", "date", "new_cases", "7_day_average")`: Selecciona las columnas relevantes para el resultado.
        - `.orderBy("country", "date")`: Ordena el DataFrame resultante por país y fecha.
    - `println("Media móvil de 7 días de nuevos casos para los top 10 países:")` y `movingAverage.show()`: Imprime un encabezado y muestra el DataFrame `movingAverage` en la consola.

13. **Guardado de Resultados en Formato Parquet:**
    - `movingAverage.write.mode("overwrite").partitionBy("country").parquet("covid_analysis_results")`:

guarda el DataFrame `movingAverage` en el sistema de archivos (que puede ser local, HDFS, S3, etc.) en formato Parquet. Vamos a desglosarlo parte por parte:

- **`movingAverage.write`**: Esto inicia la operación de escritura del DataFrame `movingAverage`. El objeto `DataFrame` en Spark tiene un atributo `.write` que proporciona métodos para guardar los datos.
- **`.mode("overwrite")`**: Esto especifica el modo de escritura. En este caso, `"overwrite"` significa que si ya existe un directorio o conjunto de archivos en la ruta de destino (`"covid_analysis_results"`), Spark lo eliminará y escribirá los nuevos datos. Otros modos comunes incluyen:
    - `"append"`: Añade los nuevos datos a los archivos existentes (si los hay).
    - `"ignore"`: Si la ruta de destino ya existe, la operación de escritura no hará nada y no generará ningún error.
    - `"error"` o `"errorifexists"`: Si la ruta de destino ya existe, la operación de escritura fallará y lanzará una excepción.
- **`.partitionBy("country")`**: Esto indica que los datos se deben particionar por la columna `"country"`. La partición es una forma de organizar los datos en el sistema de archivos basada en los valores de una o más columnas. En este caso, Spark creará subdirectorios dentro de `"covid_analysis_results"` para cada valor único en la columna `"country"`. Dentro de cada subdirectorio, se guardarán los datos correspondientes a ese país. La partición es útil para mejorar el rendimiento de las consultas posteriores que filtran por la columna de partición, ya que Spark solo necesitará leer los directorios relevantes.
- **`.parquet("covid_analysis_results")`**: Esto especifica el formato en el que se guardarán los datos y la ruta de destino.
    - **`.parquet`**: Indica que los datos se deben guardar en formato Parquet. Parquet es un formato de archivo columnar binario optimizado para el procesamiento de datos a gran escala. Ofrece ventajas como compresión eficiente y lectura selectiva de columnas, lo que puede mejorar significativamente el rendimiento de las consultas analíticas.
    - **`"covid_analysis_results"`**: Es la ruta de directorio donde se guardarán los archivos Parquet. Si el sistema de archivos es HDFS, esta será una ruta en HDFS. Si se ejecuta en modo local, será una ruta en el sistema de archivos local.

En resumen, esta línea de código toma el DataFrame `movingAverage` y realiza las siguientes acciones:

1. **Verifica si existe un directorio llamado `covid_analysis_results` en la ubicación especificada.**
2. **Si existe, lo elimina debido al modo `"overwrite"` (¡cuidado con esto, los datos existentes se perderán!).**
3. **Organiza los datos del `movingAverage` en subdirectorios dentro de `covid_analysis_results`, donde cada subdirectorio corresponde a un valor único de la columna `"country"` (por ejemplo, `covid_analysis_results/country=Spain/`, `covid_analysis_results/country=Italy/`, etc.).**

4. **Guarda los datos reales dentro de cada subdirectorio en formato de archivo Parquet.**

Beneficios de guardar en Parquet con partición:

- **Rendimiento de lectura mejorado:** Cuando se realizan consultas que filtran por la columna `"country"`, Spark puede utilizar la información de la partición para leer solo los archivos relevantes, en lugar de escanear todo el conjunto de datos. Esto puede reducir significativamente el tiempo de procesamiento y los recursos utilizados.
- **Compresión eficiente:** Parquet utiliza técnicas de compresión columnar que suelen resultar en tamaños de archivo más pequeños en comparación con otros formatos como CSV.
- **Esquema evolutivo:** Parquet permite la evolución del esquema, lo que significa que se pueden agregar nuevas columnas al esquema de los datos guardados sin necesidad de volver a escribir todos los datos.

En el contexto del programa completo, esta línea guarda los resultados del cálculo de la media móvil de 7 días de nuevos casos para los principales países en un formato eficiente y organizado para su posterior análisis o uso.

Posible Resultado en la Consola:

Primero, si el archivo `covid_data.csv` no existe en el directorio local, verás un mensaje indicando que se está descargando el archivo y luego que se ha descargado.

A continuación, se mostrarán los **10 países con más casos acumulados** de COVID-19 hasta la última fecha disponible en el dataset. La salida tendrá la siguiente estructura (los países y el número de casos serán los reales del dataset):

```
Top 10 países con más casos acumulados:
+---------------+-----------+
|country        |total_cases|
+---------------+-----------+
|US             |XXXXXXXXX  |
|India          |YYYYYYY    |
|Brazil         |ZZZZZZZ    |
|France         |AAAAAAA    |
|Germany        |BBBBBBB    |
|United Kingdom |CCCCCCC    |
|Russia         |DDDDDDD    |
|Turkey         |EEEEEEE    |
|Italy          |FFFFFFF    |
|Spain          |GGGGGGG    |
+---------------+-----------+
```

Nota: Los valores `XXXXXXXXX`, `YYYYYYY`, etc., serán los números reales de casos acumulados para cada país en el momento en que se actualizó el dataset. Dado que la ubicación actual es España (según la conversación anterior), es muy probable que "Spain" aparezca en esta lista.

Después de mostrar los 10 principales países, se mostrará la **media móvil de 7 días de nuevos casos** para estos 10 países. Esta salida será más extensa, mostrando la fecha, los nuevos casos diarios y la media móvil de 7 días para cada uno de los 10 países principales, ordenados por país y fecha:

```
Media móvil de 7 días de nuevos casos para los top 10 países:
+---------------+----------+---------+-------------------+
|country        |date      |new_cases|7_day_average      |
+---------------+----------+---------+-------------------+
|Brazil         |2020-01-22|0        |0.0                |
|Brazil         |2020-01-23|0        |0.0                |
|Brazil         |2020-01-24|0        |0.0                |
|Brazil         |2020-01-25|0        |0.0                |
|Brazil         |2020-01-26|0        |0.0                |
|Brazil         |2020-01-27|0        |0.0                |
|Brazil         |2020-01-28|0        |0.0                |
|Brazil         |2020-01-29|0        |0.0                |
|Brazil         |2020-01-30|0        |0.0                |
|Brazil         |2020-01-31|0        |0.0                |
|...            |...       |...      |...                |
|Germany        |2020-01-22|0        |0.0                |
|Germany        |2020-01-23|0        |0.0                |
|Germany        |2020-01-24|0        |0.0                |
|Germany        |2020-01-25|0        |0.0                |
|Germany        |2020-01-26|0        |0.0                |
|Germany        |2020-01-27|1        |0.14285714285714285|
|Germany        |2020-01-28|0        |0.14285714285714285|
|...            |...       |...      |...                |
|Spain          |2020-01-22|0        |0.0                |
|Spain          |2020-01-23|0        |0.0                |
|Spain          |2020-01-24|0        |0.0                |
|Spain          |2020-01-25|0        |0.0                |
|Spain          |2020-01-26|0        |0.0                |
|Spain          |2020-01-27|0        |0.0                |
|Spain          |2020-01-28|0        |0.0                |
|Spain          |2020-01-29|0        |0.0                |
|Spain          |2020-01-30|0        |0.0                |
|Spain          |2020-01-31|0        |0.0                |
|Spain          |2020-02-01|1        |0.14285714285714285|
|...            |...       |...      |...                |
+---------------+----------+---------+-------------------+
```

Nota: La salida de la media móvil comenzará con valores `null` o `0.0` para los primeros 6 días de cada país, ya que no hay suficientes datos para calcular una media móvil de 7 días completa.

Finalmente, el programa guardará los resultados de la media móvil en formato Parquet, particionados por país, en un directorio llamado `covid_analysis_results` en tu sistema de archivos local. No se imprimirá ningún mensaje específico sobre esta operación en la consola, pero podrás verificar la creación del directorio y los archivos Parquet dentro.

En resumen, el resultado principal en la consola será la tabla de los 10 países con más casos acumulados y la tabla de la media móvil de 7 días de nuevos casos para esos países, mostrando la evolución de la pandemia en los lugares más afectados.