TAREA 1 - APRENDIENDO SPARK - BIG DATA

Estudiante: Yoksan Varela Cambronero

20 de Junio, 2024

1. Contenido del archivo comprimido

Dentro del archivo comprimido se encuentran, además de este documento, los siguientes elementos:

1. **Folder datasets:** Contiene los tres archivos solicitados en la sección "Datos de Entrada"del enunciado de la tarea. Estos fueron creados usando ChatGPT, y los mismos cuentan con algunos espacios vacíos seleccionados de forma aleatoria por parte de la AI a manera de emular posibles problemas a la hora de la manipulación de datos.

   El archivo *atletas.csv* cuenta con 50 diferentes atletas. Para poder ejercitar los códigos usados en esta tarea, el archivo *correr.csv* cuenta con 212 registros, y *nadar.csv* contiene 180 registros.

2. **Folder library:** Este folder contiene 5 archivos de Python que contienen todas funciones utilizadas a lo largos del código MAIN. En la sección *Funciones y sus respectivas Prueba Unitarias* de este documento se explica con más detalle las funciones y las pruebas unitarias asociadas a las mismas.

3. **Folder unitary_tests:** De firma análoga al folder anterior, en este se almacenan todas las pruebas unitarias para las funciones mencionadas anteriormente. Para más información detallada sobre estas funciones, referirse a la sección *Funciones y sus respectivas Prueba Unitarias.*

4. **Archivo Big Data - Tarea 1.pdf:** Enunciado de la tarea provisto por el profesor.

5. **Archivo build_run_docker.bat:** Este documento fue creado para facilitar la creación y la ejecución del contenedor de Docker en mi computadora personal, la cual cuenta con el OS Windows 11. Internamente, este archivo tiene las siguentes líneas: *docker build –tag tarea1_yoksanvarela .* y *docker run -i -t tarea1_yoksanvarela /bin/bash.*

6. **Archivo command_list.txt:** Es un archivo plano de texto que contiene las líneas necesarias para poder correr el programa principal y las pruebas unitarias. Cabe la pena rescatar que estos comandos hay que correrlos de forma manual en la consola del contenedor de Docker, y son los mismos comandos que se citan en la sección *Ejecución del MAIN y las Pruebas Unitarias.*

7. **Archivo Dockerfile:** Es la imagen usada en Docker, misma imagen provista por el profesor.

8. **Archivo program_YoksanVarela.py:** Es el programa principal donde se resuelve el problema planteado en la tarea 1.

2. Decisiones de diseño de la solución

A continuación se listan las decisiones de diseño tomadas para el desarrollo de esta solución:

1. **Las funciones para lectura de los CSV (read_csv.py) y manejos de los argumentos a la hora de llamar el código principal (args_parser.py) NO tienen pruebas unitarias:** Esto se debe a que la función de estas funciones es vital para que el código principal se pueda ejecutar o no, además de la dificultad de probarlas con elementos de memoria en lugar de archivos reales. Es por eso que el hecho que programa principal pueda ejecutarse es, en sí, la prueba a esta funciones.

2. **Líneas con NaN or NULL en los archivos CSV de entrada fueron descartados:** Esto se realizar en la mayoría de los CSV de entrada en algún punto de la ejecución del programa principal, en especial en el archivo atletas.csv (dado que no hay forma de recuperar la informacíon de los otros archivos). No obstante, en muchas funciones se incluyó un manejo de NaN o Null, y estos casos son ejercitados en las pruebas unitarias.

3. **El uso del inglés en todos los códigos:** Aunque parezca trivial, considero importante mencionar esta decisión. De forma personal, me siento más cómodo programando en inglés (incluyendo comentarios, markdowns, etc.) dada mi experiencia laboral, además de poder hacer códigos legibles de manera universal.

   Esta práctica podría dar pie a pensar que esta solución podría ser un plagio o que fue desarrollada por otra persona, pero hago constar que el total de esta entrega fue desarrollado por mi persona.

3. Ejecución del MAIN y las Pruebas Unitarias

Estos son los pasos para poder ejecutar tanto el código principal como las pruebas unitarias:

1. Construir el contenedor:

```
docker build --tag tarea1\_yoksanvarela .
```

2. Ejecutar el contenedor:

```
docker run -i -t tarea1_yoksanvarela /bin/bash
```

3. Correr las pruebas unitarias: En la consola del contenedor, ejecutar los siguientes comandos:

```
pytest unitary_tests/test_data_integrity.py
pytest unitary_tests/test_data_transformation.py
pytest unitary_tests/test_data_analysis.py
```

4. Correr el programa principal: En la consola del contenedor, ejecutar el siguiente comando (es una sola línea:

```
spark-submit program_YoksanVarela.py ./datasets/atletas.csv
./datasets/nadar.csv ./datasets/correr.csv
```

4. Funciones y sus respectivas Prueba Unitarias

Las funciones usadas a lo largo del programa principal están contenidas en 5 archivos dentro de la carpeta *library*. Las pruebas unitarias están contenidas en 3 archivos dentro de la carpeta *unitary_tests*. A continuación se explican todas las funciones:

- **Función en args_parser.py:**
  - **Función:** ArgParser()
  - **Descripción:** Se encarga de procesar los argumentos que se pasan a la hora de ejecutar el código principal, los cuales son los tres archivos CSV, el retornar el nombre de los archivos.
  - **Pruebas Unitarias:** Ninguna.
- **Función en read_csv.py:**
  - **Función:** read_csv(csv_file_name)
  - **Descripción:** Crea un dataframe usando el archivo que se pasa como parámetro. Se escoje el schema correcto dependiendo del nombre del archivo, de manera que es una función única para la carga de los tres archivos solicitados.
  - **Pruebas Unitarias:** Ninguna.
- **Funciones en data_integrity.py:**
  - **Función:** clean_nan(dataframe)

- **Descripción:** Elimina las líneas que contengan NaN or NULL del dataframe que se pase por parámetro. El dataframe resultante es retornado.
- **Pruebas Unitarias:** En el archivo test_data_integrity.py:
  - **test_remove_rows_nan(spark_session):** Verifica que las líneas con NULL sean removidas.
- **Función:** date_format(date,formats)
- **Descripción:** Transforma una fecha provista en el parámetro date en YYY-MM-DD. Soporta los formatos de fechas definidos en la lista formats que es recibida como parámetro también. El dataframe resultante es retornado.
- **Pruebas Unitarias:** En el archivo test_data_integrity.py:
  - **test_correct_date(spark_session):** Verifica en un datraframe con diferentes formatos de fechas se retorne un dataframe con el formato de fecha deseado.

- **Funciones en data_transformation.py:**
  - **Función:** dataframe_joiner_byEmail(dataframe1, dataframe2)
  - **Descripción:** Hace un Join de los dos diferentes dataframes que entran como parámetros usando dos columnas en específico para el join: Correo_Electronico y Correo_Electronico_Atleta. El dataframe resultante es retornado, o bien, un estado de FAIL si no es capaz de realizar el join.
  - **Pruebas Unitarias:** En el archivo test_data_transformation.py:
    - **test_dataframe_joiner_byEmail(spark_session):** Verifica la unión exitosa de dos dataframes con las mismas columnas.
    - **test_dataframe_joiner_notSameColumns(spark_session):** Prueba la prevención del caso donde alguna de las dos columnas con la que se hace el join no está presente.
  - **Función:** keep_columns(dataframe)
  - **Descripción:** Elimina todas las columnas del dataframe por parámetro que no son de interés, manteniendo solamente Correo_Electronico, Distancia_Total_(m) y Fecha. El dataframe resultante es retornado, o bien, un estado de FAIL si no es capaz de realizar este proceso debido a que alguna de las columnas importantes no está presente.
  - **Pruebas Unitarias:** En el archivo test_data_transformation.py:
    - **test_keep_important_columns(spark_session):** Verifica que se mantengan las columnas de interés.
    - **test_keep_missing_columns(spark_session):** Prueba la prevención del caso donde falta alguna de las columnas importantes.
  - **Función:** dataframe_union(dataframe1, dataframe2)
  - **Descripción:** Aplica un Union entre los dos dataframes en los parámetros, siempre y cuando ambos dataframes tengan las mismas columnas. El dataframe resultante es retornado, o bien, un estado de FAIL si no es capaz de realizar este proceso debido a que alguna de las columnas importantes no está presente.
  - **Pruebas Unitarias:** En el archivo test_data_transformation.py:
    - **test_concatenate_same_columns(spark_session):** Verifica una unión exitosa de dos dataframes con las mismas columnas.
    - **test_concatenate_different_columns(spark_session):** Prueba la prevención del caso donde las columnas no son las mismas.
  - **Función:** aggregate_by_email_date(dataframe)

- **Descripción:** Hace groupBy en el dataframe con base a las columnas Correo_Electronico_Atleta y Fecha. Durante el agrupamiento, el código suma todas las distancias totales y cuenta la cantidad de veces que aparece el mismo correo electrónico. Finalmente, crea la columna Distancia_promedio_dia(m) de manera manual al dividir la distancia total entre el conteo de correos, para luego eliminar las líneas con NaN y NULL. Retorna el dataframe resultante o el un estado de FAIL en el caso de que el proceso no se pueda realizar.
- **Pruebas Unitarias:** En el archivo test_data_transformation.py:
  - **test_aggr_by_email_date(spark_session):** Verifica que el proceso de agregación se complete de forma correcta y el cálculo del promedio.
  - **test_aggr_NaN_value(spark_session):** Comprueba el manejo de NaN o NULL en columnas que no se usan para durante el proceso de groupBy.
  - **test_aggr_missing_date (spark_session):** Comprueba el manejo del fallo si la columna Fecha no existe para el proceso de groupBy, y cómo afecta esto al cálculo del promedio por día.
  - **test_aggr_using_Null(spark_session):** Comprueba el manejo de NaN o NULL en la columna Fecha que se usa durante el proceso de groupBy, y cómo afecta esto al cálculo del promedio por día.
  - **test_aggr_average(spark_session):** Comprueba el cálculo del promedio por día de manera más rigurosa.

- **Función en data_analysis.py:**
  - **Función:** top_athletes_totalDistance_perCountry(dataframe)
  - **Descripción:** Hace un groupBy por Nombre y Pais, el cual suma la distancia total y los promedios por día durante el agrupamiento. Retorna el dataframe resultante o el un estado de FAIL en el caso de que el proceso no se pueda realizar.
  - **Pruebas Unitarias:** En el archivo test_data_analysis.py:
    - **test_top_athletes_total_distance_oneCountry(spark_session):** Verifica la agrupaciíon pero para el caso donde solo hay un país.
    - **test_top_athletes_total_distance_moreCountries(spark_session):** Verifica la agrupaciíon con más de un país.
    - **test_top_athletes_total_distance_null(spark_session):** Para probar el comportamiento del código cuando hay NULL en los datos que se suman.
    - **test_top_athletes_group_null(spark_session):** Para probar el comportamiento del código cuando hay NULL en los datos que se usan durante el groupBy.
    - **test_top_athletes_missingColumn(spark_session):** Para comprobar el modo de fallo del código cuando falta una columna que se usa durante el groupBy.