

ÍNDICE

1. Introducción al ejercicio de evaluación de Spark.....	2
2. Instrucciones y recomendaciones para resolver la práctica.....	3
3. Preguntas de evaluación	8

1. INTRODUCCIÓN AL EJERCICIO DE EVALUACIÓN DE SPARK

Una compañía de telecomunicaciones quiere obtener *insights* de la muestra de 30 clientes obtenida al final de este mes y con información también del mes anterior. Los analistas de datos tendrán que agregar esta información, que se proporciona para responder a 10 cuestiones que ha planteado la directiva.

Las tablas de *input* son las mismas que se utilizaron en el ejercicio guiado de Spark:

- *df_clientes*: muestra información de cada cliente (id, nombre, edad, ciudad, país).
- *df_facturas_mes_ant*: muestra la factura del mes anterior junto con el plan que tenían contratado para cada cliente. Un mismo cliente puede tener contratadas dos ofertas con la compañía, en cuyo caso habrá dos filas (de ahora en adelante, registros) con el mismo “id_cliente” y distinto “id_oferta”.
- *df_facturas_mes_actual*: es una tabla equivalente a la anterior, pero muestra la información actualizada a último día del mes actual (agosto de 2020).
- *df_consumos_diarios*: muestra para cada cliente, y diariamente, los datos móviles consumidos, el tráfico, SMS enviados y minutos de llamadas a fijos y móviles durante el mes en curso.
- *df_ofertas*: muestra un catálogo de los distintos paquetes ofrecidos por la empresa, junto con una descripción y sus precios.

2. INSTRUCCIONES Y RECOMENDACIONES PARA RESOLVER LA PRÁCTICA

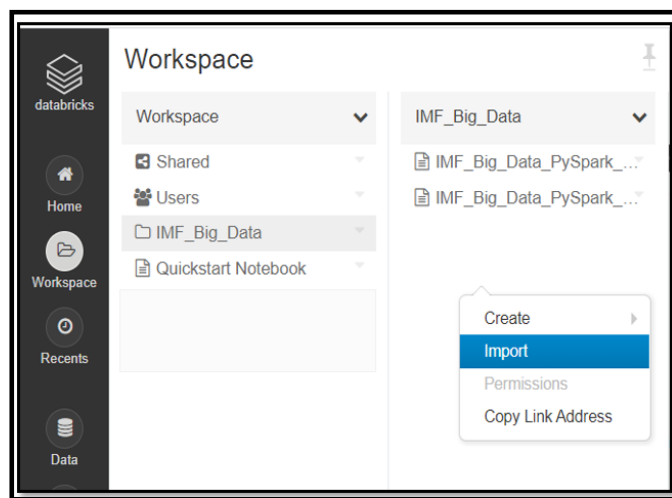
Para la resolución de la práctica, es aconsejable resolverlo en primer lugar desde la plataforma Databricks, pues se trabaja de forma mucho más visual y rápida, especialmente si se cuenta con máquinas con recursos limitados (menos de 12 GB de RAM).

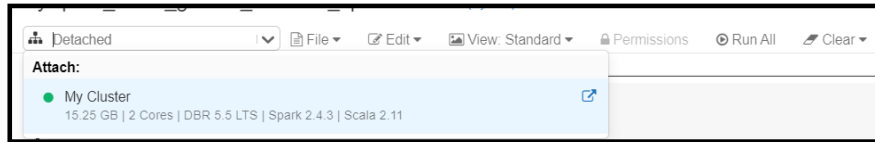
Por ello, se incluye adjunto a esta práctica un *notebook* de Databricks (“PySpark_Telco_evaluacion_Databricks.py”), que contiene las 10 preguntas de evaluación, con una celda de separación entre ellas para que podáis aplicar los comandos que se crean oportunos para resolver cada caso.

Comentado [ABN1]: Adjuntar archivo

Es muy sencillo importar el *notebook*: sobre la pestaña “Workspace”, se abre la carpeta donde se aloja el *notebook* → clic derecho en el lado donde aparece en la imagen siguiente → “Import” → arrastrar el *notebook* “.py” mencionado anteriormente.

Es importante haya un clúster ya activo porque, una vez importado, habrá que asignar el *notebook* a un clúster, por ejemplo, el que se creó durante el ejercicio guiado:





Se incluyen también los datos de las cinco tablas de *input* para que se creen *dataframes* y se almacenen en Databricks. De la misma forma, se incluyen también los *imports* necesarios para ejecutar todos los comandos que se buscan con la práctica sin problemas.

Es **muy importante** que se parta de la lectura de las tablas de *input* que sean necesarias para resolver cada una de las preguntas. Por ejemplo, si para resolver la cuestión uno son necesarias las tablas “df_clientes” y “df_ofertas”, la resolución de la cuestión debe comenzar con las dos líneas siguientes:

```
df_clientes = spark.read.table("df_clientes")
```

```
df_ofertas = spark.read.table("df_ofertas")
```

De esta manera, se podrá llegar a la misma solución partiendo de los mismos datos de entrada. Si en la cuestión dos se parte de un *dataframe* modificado en la cuestión anterior, se distorsionarán los resultados.

A partir de ahí, siempre será posible llegar al *dataframe* que se pide como solución mediante la utilización de diferentes comandos, y se valorará aquellas formas que sean óptimas. No es lo mismo llegar a una solución aplicando un *join* de dos tablas más una transformación “withColumn”, que realizar dos *joins* y una unión para obtener el mismo *dataframe* resultante. Lógicamente, será mejor puntuada la primera de las maneras, a modo de ejemplo.

Una vez se ha comprobado que se obtienen los *dataframes* esperados para cada una de las cuestiones desde Databricks, se ejecutan de manera separada desde la PySpark *Shell*, como se ha visto en el ejercicio de repaso.

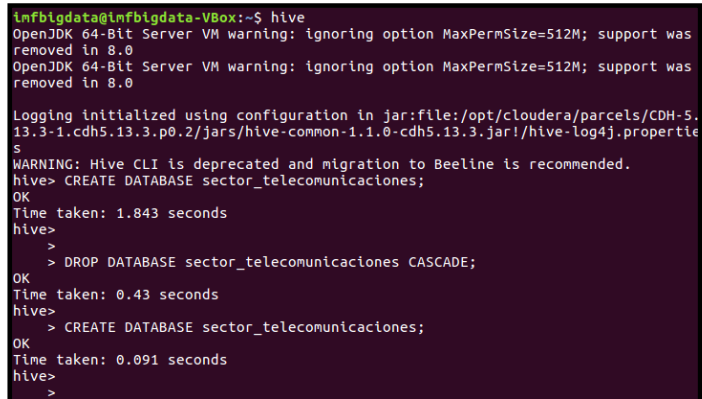
Es muy importante que esté vacía la BBDD “sector_telecomunicaciones” antes de proceder a resolver el caso de evaluación. Si estuvieran las tablas cargadas del ejercicio de repaso, habría que borrarla y volver a crearla de la siguiente forma:

Desde la terminal de Linux, se ejecuta el siguiente comando:

```
DROP DATABASE sector_telecomunicaciones CASCADE;
```

Ahora, desde la misma terminal de Linux, se ejecuta el comando:

```
CREATE DATABASE sector_telecomunicaciones;
```



```
lmfbigdata@lmfbigdata-VBox:~$ hive
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=512M; support was
removed in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=512M; support was
removed in 8.0

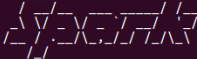
Logging initialized using configuration in jar:file:/opt/cloudera/parcels/CDH-5.
13.3-1.cdh5.13.3.p0.2/jars/hive-common-1.1.0-cdh5.13.3.jar!/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> CREATE DATABASE sector_telecomunicaciones;
OK
Time taken: 1.843 seconds
hive>
>
> DROP DATABASE sector_telecomunicaciones CASCADE;
OK
Time taken: 0.43 seconds
hive>
> CREATE DATABASE sector_telecomunicaciones;
OK
Time taken: 0.091 seconds
hive>
>
```

Es importante que estos comandos se realicen por terminal y no desde Hue, porque ocasiona problemas.

Se muestra un ejemplo de cómo cargar una de las tablas (habrá que proceder de la misma manera con las cuatro restantes):

```

lnfbigdata@lnfbigdata-VBox:~$ pyspark2
Python 2.7.12 (default, Jul 21 2020, 15:19:50)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license()" for more information.
20/09/17 21:31:23 WARN lineage.LineageWriter: Lineage directory /var/log/spark2/lineage doesn't exist or is not w
ritable. Lineage for this application will be disabled.
20/09/17 21:31:25 WARN lineage.LineageWriter: Lineage directory /var/log/spark2/lineage doesn't exist or is not w
ritable. Lineage for this application will be disabled.
Welcome to

 version 2.4.0.cloudera2

Using Python version 2.7.12 (default, Jul 21 2020 15:19:50)
SparkSession available as 'spark'.
>>>
>>> columns = ['id_oferta', 'descripcion', 'importe']
>>> data = [(1, 'Fibra Optica 600MB + Fijo ilimitado + Television', 70.00),
... (2, 'Fibra Optica 600MB + Fijo ilimitado + Television', 70.50),
... (3, 'Fibra Optica 300MB + Movil llamadas y datos ilimitados + Television', 85.00),
... (4, 'Fibra Optica 600MB + Movil llamadas y datos ilimitados + Television', 95.00),
... (5, 'Fibra Optica 300MB + Fijo ilimitado + Movil llamadas y datos ilimitados', 102.50),
... (6, 'Fibra Optica 600MB + Fijo ilimitado + Movil llamadas y datos ilimitados + Television', 124.50),
... (7, 'Fibra Optica 300MB + Movil llamadas ilimitadas + Television', 80.50),
... (8, 'Fibra Optica 300MB + Movil datos ilimitados + Television', 84.00),
... (9, 'Fijo llamadas ilimitadas + Movil llamadas y datos ilimitados', 42.50),
... (10, 'Fijo llamadas ilimitadas + Movil llamadas ilimitadas', 31.99)]
>>>
>>> df_ofertas = spark.createDataFrame(data, columns)
>>> df_ofertas.write.mode("overwrite").saveAsTable('sector_telecomunicaciones.df_ofertas')
[Stage 0:]>

```

Para cada una de las cuestiones, se tomarán dos capturas de pantalla: una que incluye los comandos que se lanzan por la `pyspark-shell` y otra con el `dataframe` resultante.

Para poder pegar varias líneas de una vez en la consola de PySpark, es necesario añadir al final de la línea el símbolo “\n”, como ya se ha comentado durante el módulo.

Por último, se debe crear un *script* que agrupe todos los comandos necesarios para completar cada una de las cuestiones. El *script* debe comenzar con la línea **# encoding: utf-8**.

Un ejemplo de cómo se pide el *script* final sería el siguiente:

```

1 # encoding: utf-8
2
3 from pyspark.sql.functions import *
4 from pyspark.sql.types import *
5 from pyspark.sql.window import Window
6 from pyspark.sql import SparkSession
7
8 spark = SparkSession.builder.appName('Spark_IMF_Big_Data').getOrCreate()
9
10 # CASO 1
11
12 df_facturas_mes_ant = spark.read.table("sector_telecomunicaciones.df_facturas_mes_ant")
13 df_facturas_mes_actual = spark.read.table("sector_telecomunicaciones.df_facturas_mes_actual")
14
15 Comandos de Spark para resolver el Caso 1
16
17 # CASO 2
18
19 df_ofertas = spark.read.table("sector_telecomunicaciones.df_ofertas")
20
21 Comandos de Spark para resolver el Caso 2
22
23
24
25 # CASO 3
26
27
28
29
30

```

Al final del documento que se entregue con las capturas de la resolución de las 10 cuestiones, habrá que añadir un anexo en el que se ejecute el comando `spark2-submit` pasándole el *script* que se construya.

Nota: la letra “ñ” se sustituirá por el símbolo “Ã±”. No influye ni penaliza en la resolución del ejercicio.

En resumen, las entregas de este caso de negocio serán:

1. Documento con capturas de la PySpark *shell* con los comandos y *dataframes* resultantes que resuelven cada una de las 10 cuestiones. Como anexo, añadir captura del lanzamiento por terminal de Linux del `spark2-submit` con el *script* “.py” que se genere.
2. *Script* “.py”, sobre el que se ejecuta el comando “`spark2-submit`”.
3. HTML exportado de Databricks con las cuestiones resueltas.

3. PREGUNTAS DE EVALUACIÓN

Se presentan a continuación las 10 preguntas prácticas de evaluación sobre diferentes aspectos acerca del negocio y facturación en los dos últimos meses:

CASO 1: mostrar por pantalla el número *total* de clientes del mes anterior con más de un contrato con la compañía.

CASO 2: generar un nuevo *dataframe* de facturas del mes actual que asigne un 7 % de descuento a todos los contratos de clientes que ya existían en el mes anterior (los contratos de los clientes nuevos seguirán con el mismo importe). Mantener la columna "importe" y crear una nueva columna "importe_dto" con el nuevo importe, casteada a dos decimales.

Mostrar, además, el resultado ordenado por los campos “id_cliente” ascendente e importe descendente.

CASO 3: por problemas de saturación en la red debido al incremento en el uso de datos por el confinamiento debido a la COVID-19, se decide limitar el uso de datos este mes subiendo la tarifa de todas las ofertas que incluyan "datos ilimitados" en un 15 %. Obtener un DF con las mismas columnas que el DF de facturas del mes actual y, a mayores, la columna "importe_dto".

CASO 4: crear una nueva variable "grupo_edad" que agrupe a los clientes, tanto del “mes_actual” como del “mes_anterior” en cuatro rangos según su edad asignando valores del uno al cuatro según el rango al que pertenezcan (18-25 (1), 26-40 (2), 41-65 (3), >65 (4)).

Obtener una tabla resumen que extraiga para cada uno de los cuatro grupos identificados la media de consumo de datos, SMS enviados, “minutos_movil”, “minutos_fijo”, con todos los campos casteados a dos decimales y ordenar el DF por “grupo_edad” ascendente. Extraer conclusiones.

CASO 5: se quiere realizar un estudio por sexo para analizar si son las mujeres o los hombres quienes consumen más datos durante el fin de semana y hacen más llamadas desde el móvil. Para ello, se deberá, sin ayuda de un calendario, extraer el día de la semana al que corresponde cada una de las fechas del mes de agosto para saber cuáles son fin de semana (se consideran días de fin de semana el viernes, sábado y domingo).

El DF a obtener deberá tener dos registros con las siguientes columnas: sexo, “total_mins_movil_finde”, “total_datos_moviles_finde”. Extraer conclusiones tras presentar el DF resultante.

CASO 6: obtener un DF que contenga cuatro registros, que serán el cliente de cada grupo edad que más datos móviles ha consumido durante los 15 primeros días del mes de agosto (día 15 incluido en el cálculo).

El DF deberá contener las columnas: nombre, edad, "grupo_edad", "datos_moviles_total_15", "max_sms_enviados_15" ("max_sms_enviados_15" contiene el máximo de sms enviados en un día por el cliente con más "datos_moviles" consumidos de cada "grupo_edad" durante esos 15 primeros días del mes).

Extraer conclusiones en cuanto a datos consumidos y SMS enviados por cada "grupo_edad".

CASO 7: interesa averiguar los minutos de llamadas de los clientes que son nuevos este mes para realizar un estudio del impacto que tendrán en caso de producirse un pico de volumen de llamadas en la red.

Obtener un DF que contenga solo a los clientes nuevos de este mes con cuatro columnas: "nombre_cliente_nuevo", edad, "importe_total_mes_actual", "total_minutos".

Se ordenará el DF resultante por la columna "total_minutos", que contiene el total de minutos de llamadas fijas y móviles durante este mes para cada cliente, ordenándolos de manera que el primer cliente nuevo que aparezca sea el que menos minutos de llamadas ha realizado en el mes de agosto.

Notas: si hay solo dos clientes nuevos, el DF a obtener tendrá solo dos registros. Ha de tenerse presente que a la hora de hacer un *join*, la clave de unión debe ser única en la tabla de la derecha o, de otra manera, se multiplicarán los registros resultantes de manera descontrolada.

Tener en cuenta que, si un cliente nuevo tiene dos ofertas contratadas, la columna "importe_mes_actual" deberá contener la suma de los importes de ambas ofertas.

CASO 8: obtener un DF que contenga, para los clientes que ya existían en el mes anterior y siguen dados de alta este mes, tres columnas: nombre, edad, "n_dias_sin_sms". La

última columna se refiere a obtener el número de días en los que el cliente no ha enviado ningún SMS durante el mes de agosto. Forzar al comando *show* a que muestre 30 valores.

Si en todo el mes el cliente "A" no envió SMS tres días, esta columna deberá contener el valor "3". Si hubiera algún cliente que hubiese enviado al menos un SMS todos los días del mes, esta columna tendrá valor "0").

CASO 9: se desea obtener un coeficiente de ponderación que permita evaluar a cada cliente en función de su consumo para identificar los clientes más atractivos que forman parte de la compañía.

Este cálculo solo se realizará para los clientes que tengan una sola oferta contratada con la compañía.

Este coeficiente se obtendrá en base a los consumos diarios y, por tanto, solo se tendrán en cuenta los clientes que existen en el mes de agosto, ya que no existen datos de consumo del mes de julio. Las ponderaciones que se darán a cada uno de los consumos son las siguientes:

- 0,4 → llamadas desde teléfono móvil.
- 0,3 → "datos_móviles_MB".
- 0,2 → llamadas desde teléfono fijo.
- 0,1 → SMS enviados.

Los pasos a seguir son los siguientes:

1. Obtener la suma de todos los días de cada uno de los cuatro consumos para cada cliente.
2. Obtener el máximo del cálculo anterior de todos los clientes para poder obtener un valor entre cero y uno para cada uno de los cuatro consumos (recordar que solo aplica para los clientes con **una sola oferta contratada** en el mes de agosto. El cliente con mayor consumo de datos tendrá un valor de "1" en la columna "datos_móviles_0_1").

3. Multiplicar estas columnas obtenidas con valor entre 0 y 1 por su ponderación correspondiente (por ejemplo, la columna "datos_moviles_0_1" se multiplicará por la ponderación de datos móviles 0,4).
4. Por último, se calculará la suma de las cuatro ponderaciones obtenidas para calcular el coeficiente de ponderación buscado ("coeficiente_cliente"), casteado a tres decimales.
5. Mostrar el DF resultante ordenado por este coeficiente en sentido descendente, de manera que el primer registro corresponderá al cliente más atractivo.

El DF resultante tendrá tres columnas: nombre, edad, "coeficiente_cliente".

CASO 10: se desea averiguar la fecha en la que entre los tres clientes que más datos consumen de cada "grupo_edad" llegan a un consumo de 20 GB de datos móviles. En caso de que algún "grupo_edad" no llegue, entre los tres clientes, a 20 GB en todo el mes, se asignará *null* como valor de esta columna (1 GB = 1024 MB).

Obtener un DF que contiene cuatro registros, uno para cada "grupo_edad" y tres columnas: "grupo_edad", "fecha_20_GB", "datos_moviles_total_grupo_3_clientes".

"datos_moviles_total_grupo_3_clientes" representa el total de datos consumidos en MB por los tres clientes del grupo hasta final de mes).