La organización Kiva de préstamos entre particulares

Disponible en Kaggle en:

https://www.kaggle.com/kiva/data-science-for-good-kiva-crowdfunding/version/5

El conjunto de datos elegido recoge estadísticas de Kiva entre 2014 y 2017. Kiva abre un nuevo mundo de oportunidades para los menos favorecidos y nos permite a cualquiera de nosotros convertirnos en superhéroes. Se trata de una organización sin ánimo de lucro que ofrece pequeños préstamos para ayudar a las comunidades desatendidas que no tienen acceso a los servicios bancarios normales. Proporciona una plataforma y une a personas que estén dispuestas a prestar, un mínimo de 25 dólares, y prestatarios que expongan sus necesidades, la finalidad del dinero y las condiciones de devolución, porque no es una donación, es un préstamo.

Al crear este servicio, Kiva habilita una solución que desbloquea capital para todos y mantiene un interés financiero muy bajo para los prestatarios. Por otro lado, permite que cualquiera sea parte de la solución y brinda a las personas un amplio abanico de opciones para elegir quién, dónde, cuánto y para qué sector desea ayudar.

Variables y significado

Las variables utilizadas para describir cada préstamo son:

- id: ID único para préstamo
- funding_amount: la cantidad desembolsada por Kiva al agente de campo (USD)
- loan_amount: la cantidad desembolsada por el agente de campo al prestatario (USD)
- country_code: código ISO del país en el que se desembolsó el préstamo
- activity: Categoría más granular
- sector: Categoría de alto nivel
- use: Uso exacto del monto del préstamo
- country: Nombre completo del país en el que se desembolsó el préstamo
- region: Nombre completo de la región dentro del país
- currency: La moneda en que se desembolsó el préstamo
- partner_id: ID de la organización asociada
- posted_time: Hora a la que el agente de campo (intermediario) publica el préstamo en Kiva
- disbursed_time: Hora en que el agente de campo (intermediario) entrega el préstamo al beneficiario
- funded_time: El momento en que el préstamo publicado en Kiva es financiado por los prestamistas por completo
- term_in_months: La duración por la cual el préstamo se desembolsó en meses

- lender_count: El número total de prestamistas que contribuyeron a este préstamo.
- tags: Etiquetas para describir el caso específico
- borrower_genders: letras M, F separadas por comas, donde cada instancia representa un solo hombre / mujer en el grupo
- repayment_interval: Estado de pago
- date: fecha en la base de datos de esta operación.

Nombre completo del alumno:

INSTRUCCIONES: en cada celda debes responder a la pregunta formulada, asegurándote de que el resultado queda guardado en la(s) variable(s) que por defecto vienen inicializadas a `None`. No se necesita usar variables intermedias, pero puedes hacerlo siempre que el resultado final del cálculo quede guardado exactamente en la variable que venía inicializada a None (debes reemplazar None por la secuencia de transformaciones necesarias, pero nunca cambiar el nombre de esa variable). **No olvides borrar la línea *raise NotImplementedError()* de cada celda cuando hayas completado la solución de esa celda y quieras probarla**.

Después de cada celda evaluable verás una celda con código. Ejecútala (no modifiques su código) y te dirá si tu solución es correcta o no. En caso de ser correcta, se ejecutará correctamente y no mostrará nada, pero si no lo es mostrará un error. Además de esas pruebas, se realizarán algunas más (ocultas) a la hora de puntuar el ejercicio, pero evaluar dicha celda es un indicador bastante fiable acerca de si realmente has implementado la solución correcta o no. Asegúrate de que, al menos, todas las celdas indican que el código es correcto antes de enviar el notebook terminado.

Sobre el dataset kiva_loans.csv se pide:

(1 punto) Ejercicio 1

- Leerlo **sin intentar** que Spark infiera el tipo de dato de cada columna
- Puesto que existen columnas que contienen una coma enmedio del valor, en esos casos los valores vienen entre comillas dobles. Spark ya contempla esta posibilidad y puede leerlas adecuadamente **si al leer le indicamos las siguientes opciones adicionales** además de las que ya sueles usar: `.option("quote", "\"").option("escape", "\"")`.
- Asegúrate de que las **filas que no tienen el formato correcto sean descartadas**, indicando también la opción `mode` con el valor `DROPMALFORMED` como vimos en clase.
- Crear un nuevo DF `kivaRawNoNullDF` en el que se hayan eliminado todas las filas que tengan algún valor nulo en cualquier columna **excepto en la columna tags**, que no será relevante para el análisis y por tanto podemos ignorar sus valores nulos y mantener dichas filas.

```
# LÍNEA EVALUABLE, NO RENOMBRAR LAS VARIABLES
kivaRawDF = None
# Descomentar estas líneas para calcular la lista de columnas que sí deben tenerse en c
# tendrás que utilizar dicha lista en la operación que elimina los nulos
# columnasExceptoTags = kivaRawDF.columns
# columnasExceptoTags.remove("tags")
kivaRawNoNullDF = None
# YOUR CODE HERE
kivaRawDF = spark.read.option("header","true")\
                      .option("quote", "\"")\
                      .option("escape", "\"")\
                      .option("mode", "DROPMALFORMED")\
                      .csv("gs://ucm-bucket1/notebooks/jupyter/UCM-Tarea/kiva_loans.csv

kivaRawNoNullDF = kivaRawDF.drop("tags")

kivaRawNoNullDF = kivaRawNoNullDF.dropna()
```

```
from pyspark.sql.types import DoubleType
assert(kivaRawNoNullDF.count() == 574115)
```

from pyspark.sql import types as T **(1 punto)** Ejercicio 2

- Las columnas `posted_time` y `disbursed_time` son en realidad instantes de tiempo que Spark debería procesar como timestamp. Partiendo de `kivaRawNoNullDF`, reemplaza **ambas columnas** por su versión convertida a timestamp, utilizando `withColumn` con el mismo nombre de cada columna, y donde el nuevo valor de la columna viene dado por el siguiente código:

```
F.from_unixtime(F.unix_timestamp('nombreColumna', 'yyyy-MM-dd HH:mm:ssXXX')).c
```

- Además, convierte a `DoubleType` la columna `loan_amount` y a `IntegerType` la columna `term_in_months`.
- El DF resultante de todas estas operaciones debe quedar almacenado en la variable `kivaDF`, **cacheado**.

```
from pyspark.sql import functions as F
from pyspark.sql.functions import col
from pyspark.sql.types import IntegerType
from pyspark.sql.types import DoubleType

kivaDF = kivaRawNoNullDF

# YOUR CODE HERE
```

```
kivaDF = kivaDF.withColumn('posted_time', F.from_unixtime(F.unix_timestamp('posted_time

kivaDF = kivaDF.withColumn('disbursed_time', F.from_unixtime(F.unix_timestamp('disburse

kivaDF = kivaDF.withColumn("loan_amount", kivaDF["loan_amount"].cast(DoubleType()))

kivaDF = kivaDF.withColumn("term_in_months", kivaDF["term_in_months"].cast(IntegerType(

kivaDF.cache()
```

```
DataFrame[id: string, funded_amount: string, loan_amount: double, activity: string,
sector: string, use: string, country_code: string, country: string, region: string,
currency: string, partner_id: string, posted_time: timestamp, disbursed_time:
timestamp, funded_time: string, term_in_months: int, lender_count: string,
borrower_genders: string, repayment_interval: string, date: string]
```

```
typesDict = dict(kivaDF.dtypes)
assert(typesDict["posted_time"] == "timestamp")
assert(typesDict["disbursed_time"] == "timestamp")
assert(typesDict["loan_amount"] == "double")
assert(typesDict["term_in_months"] == "int")
cnt_cond = lambda cond: F.sum(F.when(cond, 1).otherwise(0))
nullsRow = kivaDF.select(cnt_cond(F.col("posted_time").isNull()).alias("posted_nulls"),
              cnt_cond(F.col("disbursed_time").isNull()).alias("disbursed_nulls")).head
assert(nullsRow.posted_nulls == 0)
assert(nullsRow.disbursed_nulls == 0)
```

kivaDF**(2 puntos)** Ejercicio 3

Partiendo de `kivaDF`:

- Primero, añade una nueva columna `dias_desembolso` que contenga el número de días que han pasado entre la fecha en que los prestamistas aceptaron financiar un proyecto, y la fecha en que el agente de campo entregó los fondos al beneficiario. Para ello, utiliza `withColumn` en combinación con la función `F.datediff("colFuturo", "colPasado")`
- De manera análoga, añade otra nueva columna `dias_aceptacion` que contenga el número de días entre el anuncio de la necesidad de préstamo y la aceptación de financiarlo por parte de algún prestamista.
- Reemplazar la columna `sector` por otra en la que se hayan traducido las categorías "Education" por "Educacion" (sin tilde para evitar posibles problemas) y "Agriculture" por "Agricultura", dejando como están el resto de categorías. **La sustitución no debe tener más que tres casos**: uno para cada categoría que vamos a reemplazar, y un tercero para el resto de categorías, que deben quedarse como estaban.
- El resultado debe quedar guardado en la variable `kivaTiemposDF`.

```
# LÍNEA EVALUABLE, NO RENOMBRAR VARIABLES
from pyspark.sql import functions as F
```

```
from pyspark.sql.functions import datediff

kivaTiemposDF = kivaDF
# YOUR CODE HERE

kivaTiemposDF = kivaTiemposDF.withColumn("dias_desembolso", F.datediff("funded_time", "
kivaTiemposDF = kivaTiemposDF.withColumn("dias_aceptacion", F.datediff("funded_time","p


kivaTiemposDF = kivaTiemposDF.withColumn("sector",
                            F.when(kivaTiemposDF.sector == "Education", "Educacion")
                            .when(kivaTiemposDF.sector == "Agriculture", "Agricultura")
                            .otherwise(kivaTiemposDF.sector))
```

```
assert(kivaTiemposDF.where("sector == 'Agricultura'").count() == 157003)
assert(kivaTiemposDF.where("sector == 'Educacion'").count() == 28417)
# Comprobamos que las 13 restantes se mantienen sin cambios
assert(kivaTiemposDF.groupBy("sector").count().join(kivaDF.groupBy("sector").count(), [
```

(3 puntos) Ejercicio 4

Partiendo de `kivaTiemposDF`, crear un nuevo DataFrame llamado `kivaAgrupadoDF` que tenga:

- Tantas filas como **países (`country`; no usar el código de país)**, y tantas columnas como **sectores** (cada una llamada como el sector) más una (la columna del país, que debe aparecer en primer lugar). En cada celda deberá ir el número **medio (redondeado a 2 cifras decimales)** de días transcurridos en ese país y sector *entre la fecha en que se anuncia la necesidad de préstamo y la fecha en la que un prestamista acepta financiarlo*. Esta columna ha sido calculada en la celda precedente.
- Después de esto, añadir una columna adicional `transcurrido_global` con el número **medio (redondeado a 2 cifras decimales) de días transcurridos en cada país** entre ambas fechas *sin tener en cuenta el sector*. Cada fila tendrá la media de las 15 columnas del apartado anterior.
- Por último, ordenar el DF resultante **descendentemente** en base al tiempo medio global, `transcurrido_global`. El DF resultado de la ordenación debe ser almacenado en la variable `kivaAgrupadoDF`.

PISTA: utiliza el método `pivot` con el sector para el primer apartado, envolviendo a la función de agregación con la función `F.round`, es decir, `F.round(F.funcionAgregacion(....), 2)`, y `withColumn` con una operación aritmética entre columnas en el segundo. **No debe utilizarse la función `when`** ya que Spark es capaz de hacer directamente aritmética entre objetos columna. La operación aritmética también debe estar envuelta por round: `F.round(op. aritmética entre objetos columna, 2)`.

```
from pyspark.sql.functions import mean, round
from functools import reduce
from operator import add
from pyspark.sql.functions import col, lit

kivaAgrupadoDF = None
# YOUR CODE HERE

kivaAgrupadoDF = kivaTiemposDF.groupBy("country").pivot("sector").agg(round(mean("dias_

n = lit(len(kivaAgrupadoDF.columns) -1)
rowMean = (reduce(add, (col(x) for x in kivaAgrupadoDF.columns[1:])) / n)
kivaAgrupadoDF = kivaAgrupadoDF.withColumn("transcurrido_global", rowMean)

kivaAgrupadoDF = kivaAgrupadoDF.sort(col("transcurrido_global").desc())
```

```
r1 = kivaAgrupadoDF.head()
assert(r1.country == "United States")
assert((r1.Agricultura - 12.0 < 0.01) | (r1.Agricultura - 12.17 < 0.01))
assert((r1.Educacion - 15.21 < 0.01) | (r1.Educacion - 15.33 < 0.01))
assert(r1.Wholesale - 27.5 < 0.01)
assert((r1.transcurrido_global - 20.94 < 0.01) | (r1.transcurrido_global - 21.04 < 0.01
```

(3 puntos) Ejercicio 5

Partiendo de nuevo de `kivaTiemposDF`, añadir las siguientes columnas:

- Primero, tres columnas adicionales llamadas `transc_medio`, `transc_min`, `transc_max` que contengan, respectivamente, **el número de días medio, mínimo y máximo transcurrido para proyectos de ese mismo país y ese mismo sector** entre la fecha en que se postea la necesidad de préstamo y la fecha en la que alguien acepta financiarlo (es decir, la columna `dias_aceptacion` calculada antes y utilizada también en la celda anterior). Es decir, queremos una columna extra para que podamos tener, junto a cada préstamo, información agregada de los préstamos similares, entendidos como aquellos del mismo país y del mismo sector. **No se debe utilizar JOIN sino solo funciones de ventana**.
- Finalmente, crear otra columna adicional `diff_dias` que contenga la **diferencia en días entre los días que transcurrieron en este proyecto y la media de días de los proyectos similares** (calculada en el apartado anterior). Debería ser lo primero menos lo segundo, de forma que un número positivo indica que este préstamo tardó más días en ser aceptado que la media de días de este país y sector, y un número negativo indica lo contrario. El resultado debe obtenerse aplicando operaciones aritméticas con columnas existentes, **sin utilizar `when`**.
- El DF resultante con las 4 columnas nuevas que hemos añadido debe quedar almacenado en la variable `kivaExtraInfoDF`.

```
# LÍNEA EVALUABLE, NO RENOMBRAR VARIABLES
from pyspark.sql.functions import avg, col
from pyspark.sql.functions import min, max, col
from pyspark.sql.window import Window

windowPaisSector = Window.partitionBy("country", "sector")
kivaExtraInfoDF = kivaTiemposDF

# YOUR CODE HERE

kivaExtraInfoDF = kivaExtraInfoDF.withColumn("transc_medio", avg(col("dias_aceptacion")
kivaExtraInfoDF = kivaExtraInfoDF.withColumn("transc_min", min(col("dias_aceptacion")).
kivaExtraInfoDF = kivaExtraInfoDF.withColumn("transc_max", max(col("dias_aceptacion")).

kivaExtraInfoDF = kivaExtraInfoDF.withColumn("diff_dias", (kivaExtraInfoDF["transc_max"
```

```
r = kivaExtraInfoDF.where("id = '658540'").head()
assert(r.country == 'Burkina Faso')
assert(r.transc_medio - 11.02 < 0.05)
assert(r.dias_aceptacion == 35)
assert(r.diff_dias - 24.0 < 0.001)
```