7506-2021 Parcialito de Spark

Román Vázquez Lareu

TOTAL POINTS

45 / 100

QUESTION 1

1 Pto A 25 / 50

Punto A

✓ **- 5 pts Filter tardío**

✓ **- 10 pts Utiliza como fecha solo una letra del mes.**

✓ **- 10 pts Crea un rdd por mes innecesariamente y poco escalable.**

1. El primer campo deberia ser la fecha completa.
2. Primero deberia filtrar solo el primer trimestre.
3. Podia hacer esto mismo con solo un rdd y no 3.

QUESTION 2

2 Pto B 20 / 50

Punto B

✓ **- 10 pts Operaciones innecesarias para calcular el total de tests.**

✓ **- 10 pts No filtra los tests del primer trimestre.**

✓ **- 5 pts Join innecesario para obtener las localidades con cantidad de tests mayor al promedio.**

✓ **- 5 pts Uso de takeOrdered para obtener el mayor en vez de reduce.**

4. Falta filtrar primer trimestre.
5. Todo esto es innecesario. El total es simplemente testsRDD filtrados para el primer trimeste.count()
6. Aca deberia sumar tambien el resultado del test y luego se ahorra el join de más abajo.
7. Deberia utilizar reduce para obtener solo el mayor.

```
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/hsdb to provide /usr/bin/hsdb
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/clhsdb to provide /usr/bin/clh
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/extcheck to provide /usr/bin/e
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/schemagen to provide /usr/bin/
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/xjc to provide /usr/bin/xjc (x
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/jhat to provide /usr/bin/jhat
update-alternatives: using /usr/lib/jvm/java-8-openjdk-amd64/bin/wsgen to provide /usr/bin/wsge
```

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark import SparkContext
from pyspark.sql import SQLContext
import pandas as pd


auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


# create the Spark Session
spark = SparkSession.builder.getOrCreate()

# create the Spark Context
sc = spark.sparkContext


#Data
```

1

```
tests = [("E",40,1,0),
         ("F",40,2,1),
         ("F",45,2,1),
         ("E",46,2,1),
         ("E",46,3,1),
         ("E",46,4,1),
         ("M",41,2,0),
         ("F",42,3,1),
         ("F",42,4,1),
         ("F",42,4,1),
         ("M",43,4,0),
         ("M",43,5,0)
         ]
localidades =[(1,"nom1","prov1"),
              (2,"nom2","prov1"),
              (3,"nom3","prov2"),
              (4,"nom4","prov2"),
              (5,"nom5","prov3"),
              (6,"nom6","prov4")
]
```

```
testsRDD = sc.parallelize(tests)
localidadesRDD = sc.parallelize(localidades)

# para filtrar los del primer trimestre con un filtrer y haciendo x[i] =="E" & x[i] =="F" &
# x[i] =="M"
```

Punto A

2

```
testo_por_localidad = testsRDD.map(lambda x: (x[2],x[0]))
```

```
[(1, 'E'),
 (2, 'F'),
 (2, 'F'),
 (2, 'E'),
 (3, 'E'),
 (4, 'E'),
 (2, 'M'),
 (3, 'F'),
 (4, 'F'),
 (4, 'F'),
 (4, 'M'),
 (5, 'M')]
```

```
localidad_provincia = localidadesRDD.map(lambda x: (x[0],x[2]))
```

```
[(1, 'prov1'),
 (2, 'prov1'),
 (3, 'prov2'),
 (4, 'prov2'),
 (5, 'prov3'),
 (6, 'prov4')]
```

```
test_por_provincia = localidad_provincia.join(testo_por_localidad)
```

```
[(4, ('prov2', 'E')),
 (4, ('prov2', 'F')),
 (4, ('prov2', 'F')),
 (4, ('prov2', 'M')),
 (1, ('prov1', 'E')),
 (5, ('prov3', 'M')),
 (2, ('prov1', 'F')),
 (2, ('prov1', 'F')),
 (2, ('prov1', 'E')),
 (2, ('prov1', 'M')),
 (3, ('prov2', 'E')),
 (3, ('prov2', 'F'))]
```

```
test_por_provincia_enero = test_por_provincia.filter(lambda x: x[1][1]=="E")\
                           .map(lambda x: (x[1][0],1)).reduceByKey(lambda x,y: x+y)
```

```
[('prov1', 2), ('prov2', 2)]
```

```
test_por_provincia_febrero = test_por_provincia.filter(lambda x: x[1][1]=="F")\
                            .map(lambda x: (x[1][0],1)).reduceByKey(lambda x,y: x+y)
```

```
[('prov1', 2), ('prov2', 3)]
```

```
test_por_provincia_marzo = test_por_provincia.filter(lambda x: x[1][1]=="M")/
                            .map(lambda x: (x[1][0],1)).reduceByKey(lambda x,y: x+y)
```

3

```
[('prov1', 1), ('prov3', 1), ('prov2', 1)]
```

```
relacion_enero_febrero = test_por_provincia_enero.join(test_por_provincia_febrero)\
                        .map(lambda x: (x[0],(x[1][1]/x[1][0])))
provs_enero_a_febrero_aumento_20 = relacion_enero_febrero.filter(lambda x: x[1]>1.2)
```

```
[('prov2', 1.5)]
```

```
relacion_febrero_marzo = test_por_provincia_febrero.join(test_por_provincia_marzo)\
                        .map(lambda x: (x[0],(x[1][1]/x[1][0])))
provs_febrero_a_marzo_aumento_20 = relacion_febrero_marzo.filter(lambda x: x[1]>1.2)
```

```
[]
```

Punto B

4

6

```
test_por_localidad = testsRDD.map(lambda x: (x[2],1)).reduceByKey(lambda x,y: x+y)
```

```
[(2, 4), (4, 4), (1, 1), (3, 2), (5, 1)]
```

```
cantidad_promedio_tests = test_por_localidad.map(lambda x: (1,x[1]))\
                          .reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
cantidad_promedio_tests = cantidad_promedio_tests[1]/ cantidad_promedio_tests[0]
cantidad_promedio_tests
```

5

```
2.4
```

```
localidades_mayor_al_promedio = test_por_localidad\
                                .filter(lambda x: x[1]>cantidad_promedio_tests)
```

```
[(2, 4), (4, 4)]
```

```
tests_resultado_localidad =testsRDD.map(lambda x: (x[2],x[3])).reduceByKey(lambda x,y: x+y)
```

```
[(2, 3), (4, 3), (1, 0), (3, 2), (5, 0)]
```

```
test_resultado_localidad_mayores=tests_resultado_localidad\
                          .join(localidades_mayor_al_promedio)
```

```
[(2, (3, 4)), (4, (3, 4))]
```

```
test_resultado_localidad_mayores_positividad=test_resultado_localidad_mayores\
                                        .map(lambda x: (x[0],(x[1][0]/x[1][1])))
localidad_mayor_positividad = test_resultado_localidad_mayores_positividad\
                                        .takeOrdered(1,key=lambda x: -x[1])
```

7

1 Pto A 25 / 50

Punto A

✓ **- 5 pts** **Filter tardío**

✓ **- 10 pts** **Utiliza como fecha solo una letra del mes.**

✓ **- 10 pts** **Crea un rdd por mes innecesariamente y poco escalable.**

1. El primer campo deberia ser la fecha completa.
2. Primero deberia filtrar solo el primer trimestre.
3. Podia hacer esto mismo con solo un rdd y no 3.

```
[('prov1', 2), ('prov2', 3)]
```

```
test_por_provincia_marzo = test_por_provincia.filter(lambda x: x[1][1]=="M")/
                            .map(lambda x: (x[1][0],1)).reduceByKey(lambda x,y: x+y)
```

3

```
[('prov1', 1), ('prov3', 1), ('prov2', 1)]
```

```
relacion_enero_febrero = test_por_provincia_enero.join(test_por_provincia_febrero)\
                        .map(lambda x: (x[0],(x[1][1]/x[1][0])))
provs_enero_a_febrero_aumento_20 = relacion_enero_febrero.filter(lambda x: x[1]>1.2)
```

```
[('prov2', 1.5)]
```

```
relacion_febrero_marzo = test_por_provincia_febrero.join(test_por_provincia_marzo)\
                        .map(lambda x: (x[0],(x[1][1]/x[1][0])))
provs_febrero_a_marzo_aumento_20 = relacion_febrero_marzo.filter(lambda x: x[1]>1.2)
```

```
[]
```

Punto B

4

6

```
test_por_localidad = testsRDD.map(lambda x: (x[2],1)).reduceByKey(lambda x,y: x+y)
```

```
[(2, 4), (4, 4), (1, 1), (3, 2), (5, 1)]
```

```
cantidad_promedio_tests = test_por_localidad.map(lambda x: (1,x[1]))\
                          .reduce(lambda x,y: (x[0]+y[0],x[1]+y[1]))
cantidad_promedio_tests = cantidad_promedio_tests[1]/ cantidad_promedio_tests[0]
cantidad_promedio_tests
```

5

```
2.4
```

```
localidades_mayor_al_promedio = test_por_localidad\
                                .filter(lambda x: x[1]>cantidad_promedio_tests)
```

```
[(2, 4), (4, 4)]
```

```
tests_resultado_localidad =testsRDD.map(lambda x: (x[2],x[3])).reduceByKey(lambda x,y: x+y)
```

```
[(2, 3), (4, 3), (1, 0), (3, 2), (5, 0)]
```

```
test_resultado_localidad_mayores=tests_resultado_localidad\
                          .join(localidades_mayor_al_promedio)
```

```
[(2, (3, 4)), (4, (3, 4))]
```

```
test_resultado_localidad_mayores_positividad=test_resultado_localidad_mayores\
                                        .map(lambda x: (x[0],(x[1][0]/x[1][1])))
localidad_mayor_positividad = test_resultado_localidad_mayores_positividad\
                                        .takeOrdered(1,key=lambda x: -x[1])
```

7

```
localidad_mayor_positividad
```

```
[(2, 0.75)]
```

```
localidadesRDD.filter(lambda x: x[0]==localidad_mayor_positividad[0][0])\
                                    .map(lambda x: x[1]).collect()
```

```
['nom2']
```

2 Pto B 20 / 50

Punto B

✓ **- 10 pts** **Operaciones innecesarias para calcular el total de tests.**

✓ **- 10 pts** **No filtra los tests del primer trimestre.**

✓ **- 5 pts** **Join innecesario para obtener las localidades con cantidad de tests mayor al promedio.**

✓ **- 5 pts** **Uso de takeOrdered para obtener el mayor en vez de reduce.**

4 Falta filtrar primer trimestre.

5 Todo esto es innecesario. El total es simplemente testsRDD filtrados para el primer trimeste.count()

6 Aca deberia sumar tambien el resultado del test y luego se ahorra el join de más abajo.

7 Deberia utilizar reduce para obtener solo el mayor.