Relatorio ATP - Frameworks de Big Data

Aluno:

Juan Manoel Marinho Nascimento

Introdução

Buscou-se desenvolver um trabalho prático utilizando o framework Apache Spark e a tecnologia Python com a api pyspark, no estudo de sistemas distribuídos em big data dentro do ecosistema hadoop. Esta atividade é relevante pois trata-se de uma das principais tecnologias do mercado em ascenção para processamento de grandes volumes de dados, visto que grandes do mercado utilizam python e spark para seus projetos de alta performace.

Objetivo

Analisar e compreender como funciona o armazenamento e processamento de dados não estruturados por meio de um jupyter notebook desenvolvido em python com o framework Apache Spark usando como api pyspark.

Métodos

Foi análisado os repositórios de datasets públicos com dados de diferentes exemplos, reais e artificiais, dentro das bases data.gov, kaggle e outro. Durante essa analise foi percebido e observado a variedade de dados distrintos para os quais é possivel fazer analises e gerar diferentes insights. No desenvolvimento de uma analise programada em pyspark para a utilização do Apache Spark no servidor de big data.

Foi Utilização diversos algoritmos implementados em python como filtros para entender como os dados estão estruturados e poder filtras as informações, tambem foi utilizado querys em sql visto que pyspark comporta rodar queries sql e utilização de sub-queries, PS: para mais informações basta olhar a analise a baixo.

Essa base de dados mostra as principais tipos de ocorrencias e tambem passa os dados de forma não parametrizadas nas columnas, foi feito um processo de data cleaning, para limpar aa base de dados e junção do nome da columa com a base principa.

Outro processo importante foi a utilização de agrupamento de dados por dimensões de Ano, Mes e Dia. Com os dados agrupados foi possivel fazer analises das metricas de media, maximo valor, menor valor, e tambem foi possivel segmentar por tipos de crimes

Resultados e discussões

Procurou-se conhecer e entender os diferentes tipos de dados em sua maior quantidade e variedade e sua organização, fundamentais para seu processamento e analises. Foi utilizado o Apache Spark no servidor de big data da PUC-PR para reproduzir o exemplo da unidade de aprendizagem: "Aplicações simples utilizando frameworks de big data". Para a atividade, o seguinte fluxo de execução foi implementado tendo em vista a computação distribuída: criação de um jupyter notebook, criação de um Spark Context e Session Builder, criação inicial de uma instancia do objeto spark, aplicação da leitura do arquivo csv na pasta local do usuario juan_manoel, feita as transformações e execução das operações.As seguintes transformações foram executadas: filter, groupby , Função Max, mean e juntamente com as seguintes ações: construção de querys e sql utilizando as possibilidades do spark para isso.

Conclusão

Conclui-se que a utilização do Apache Spark proporciona ganhos consideráveis para o processamento de grandes conjuntos de dados de forma paralela e distribuída, facilitando e agilizando o desenvolvimento de aplicações de processamento de dados no contexto de big data.A tecnologia Python por ser amplamente utilizada em toda a industria de técnologia dentro das areas de inteligencia artifical, ciencias de dados e big data torna o contexto dos sistemas distribuídos, mais comodos para o desenvolvimento de soluções em big data.

Referências

Google Cloud Platform. Who is Apache Spark. Visão geral apache spark, 2020 disponível em:

https://cloud.google.com/learn/what-is-apache-spark

API PySpark - Documentação da API, 2020 Disponivel em:

https://spark.apache.org/docs/latest/api/python/index.html

Course Big Data with Spark - Curso Datacamp com pyspark, 2020 disponivel em:

https://www.datacamp.com/tracks/big-data-with-pyspark

Test Check PySpark

In [3]:
```python
import os
os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3.7'
os.environ['PYSPARK_DRIVER_PYTHON'] = '/usr/bin/python3.7'
```

Analise Exploratoria ATP Dataset

Dicionario de Dados

O dicionário de dados deste dataset é declarado a seguir: Campo

Campo	DESCRIÇÃO	
Dia	Dia da ocorrência	
Mes	Mes da ocorrência	
Ano	Ano da ocorrência	
Bloco	Região da ocorrência	Região da ocorrência
Tipo	Tipo da ocorrência criminal	
Descrição	Breve descrição da ocorrência	
Descrição da localização	Descrição da localização da ocorrência (rua, por exemplo)	
Latitude	Localização da ocorrência	
Longitude	Localização da ocorrência	

In [2]:
```python
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("ATP Analise Exploratoria").config("spark.some.config.option",
"some-value").getOrCreate()
```

In [4]:
```python
from pyspark import SparkContext
sc = SparkContext
```

In [17]:
```python
# USE APENAS QUANDO NÃO TIVER RODANDO NO SERVIDOR LOCAL
#!wget -c  https://gitlab.com/juanmanoel/dataset-atp-big-data-frameworks/-/raw/master/ocorrencias_c
riminais_corrigida.csv
```

In [20]:
```python
df = spark.read.option("header",True).csv("/home2/ead2020/SEM2/juan.manoel/ocorrencias_criminais_co
rrigida.csv")
```

Analise Dataframe entendimento da tabela

In [8]:
```python
df.show()
```

```
+---+---+----+--------------------+--------------------+--------------------+----------------------
--+------------+-------------+
|Dia|Mes| Ano|               Bloco|                Tipo|           Descricao|Descrição_da_localizac
ao|    Latitude|    Longitude|
+---+---+----+--------------------+--------------------+--------------------+----------------------
--+------------+-------------+
| 18|  3|2015|     047XX W OHIO ST|             BATTERY| AGGRAVATED: HANDGUN|                  STRE
ET|41.891398861|-87.744384567|
| 18|  3|2015|066XX S MARSHFIEL...|       OTHER OFFENSE|    PAROLE VIOLATION|                  STRE
ET|41.773371528|-87.665319468|
| 18|  3|2015|044XX S LAKE PARK...|             BATTERY|DOMESTIC BATTERY ...|               APARTME
NT| 41.81386068|-87.596642837|
| 18|  3|2015|051XX S MICHIGAN AVE|             BATTERY|              SIMPLE|               APARTME
NT|41.800802415|-87.622619343|
| 18|  3|2015|    047XX W ADAMS ST|             ROBBERY|      ARMED: HANDGUN|                SIDEWA
LK|41.878064761|-87.743354013|
| 18|  3|2015| 049XX S DREXEL BLVD|             BATTERY|              SIMPLE|               APARTME
NT|41.805443345|-87.604283976|
| 18|  3|2015|   070XX S MORGAN ST|             BATTERY|DOMESTIC BATTERY ...|               APARTME
NT|41.766402779|-87.649296123|
| 18|  3|2015| 042XX S PRAIRIE AVE|             BATTERY|DOMESTIC BATTERY ...|               APARTME
NT|41.817552577|-87.619818523|
| 18|  3|2015| 036XX S WOLCOTT AVE|           NARCOTICS|POSS: CANNABIS 30...|                  STRE
ET|41.828138428|-87.672782106|
| 18|  3|2015| 097XX S PRAIRIE AVE|             BATTERY|              SIMPLE|    RESIDENCE PORCH/
H...| 41.71745472|-87.617663257|
| 18|  3|2015|130XX S DR MARTIN...|     CRIMINAL DAMAGE|          TO VEHICLE|    PARKING LOT/GARA
G...|41.658138493|-87.613672862|
| 15|  3|2015|078XX S VINCENNES...|       OTHER OFFENSE|HARASSMENT BY TEL...|    CTA GARAGE / OTH
E...|41.752406801|-87.633792381|
| 18|  3|2015|086XX S EXCHANGE AVE|   WEAPONS VIOLATION|UNLAWFUL POSS OF ...|    DRIVEWAY - RESID
E...|41.738563465|-87.552678593|
| 18|  3|2015| 014XX S ASHLAND AVE|             BATTERY|              SIMPLE|                SIDEWA
LK| 41.86304084|-87.666288555|
| 18|  3|2015| 051XX W CHICAGO AVE|               THEFT|        RETAIL THEFT|             GAS STATI
ON|41.894945606|-87.754874977|
| 18|  3|2015|077XX S KINGSTON AVE|            BURGLARY|      FORCIBLE ENTRY|               APARTME
NT|41.754602618|-87.562650741|
| 18|  3|2015|   024XX W NORTH AVE| MOTOR VEHICLE THEFT|          AUTOMOBILE|                   OTH
ER|41.910312648|-87.687806494|
| 18|  3|2015| 069XX S LOOMIS BLVD|               THEFT|       FROM BUILDING|      GROCERY FOOD STO
RE|41.768167414|-87.659053795|
| 18|  3|2015|105XX S LAFAYETTE...|PUBLIC PEACE VIOL...|    RECKLESS CONDUCT|                   ALL
EY| 41.70284845|-87.624588931|
| 18|  3|2015| 087XX S KIMBARK AVE|               THEFT|       FROM BUILDING|            BAR OR TAVE
RN|41.736588206| -87.59299436|
+---+---+----+--------------------+--------------------+--------------------+----------------------
--+------------+-------------+
only showing top 20 rows
```

In [9]:
```python
from pyspark.sql import functions as F
from pyspark.sql.types import IntegerType
from pyspark.sql.functions import col, avg
import pyspark.sql.functions as func
from pyspark.sql.functions import countDistinct, avg, stddev
```

Quantidade de crimes por ano?

- Usando Select para formatar a data da coluna Ano e definindo meu alias para Ano
- Em seguida passei o agrupamento por Ano
- contei os ocorridos por ano.

In [10]:
```python
df.select(F.date_format('Ano','dd/MM/yyyy').alias('Ano')).groupby('Ano').count().show()
```

```
+----------+-------+
|       Ano|  count|
+----------+-------+
|01/01/2011| 359474|
|01/01/2004| 482572|
|01/01/2006| 451114|
|01/01/2013| 308550|
|01/01/2002| 478166|
|01/01/2001| 497358|
|01/01/2007| 447292|
|      null|6418484|
|01/01/2008| 425068|
|01/01/2018| 120884|
|01/01/2014| 279712|
|01/01/2005| 459528|
|01/01/2015| 266472|
|01/01/2003| 489758|
|01/01/2012| 346588|
|01/01/2010| 382404|
|01/01/2009| 393972|
|01/01/2017| 270630|
|01/01/2016| 279158|
+----------+-------+
```

Quantidade de crimes por ano que sejam do tipo NARCOTICS?

Criando uma View temporaria em memoria com o dataframe

In [11]:
```python
df.createOrReplaceTempView("ocorrencias")
```

Usando SQL para afzer as querys

- Primeiro defini meu target(ano)
- Depois passei um count(*) para pegar todas as ocorrencias
- Como Condicional passei o tipo da ocorrencias NARCOTICS
- Agrupando por Ano e Ordenando por Ano

In [12]:
```python
CrimesPorAnoNarc = spark.sql("SELECT  Ano, COUNT(*) as ocorrencias FROM ocorrencias WHERE Tipo = 'N
ARCOTICS' GROUP BY Ano ORDER BY Ano")
```

In [13]:
```python
CrimesPorAnoNarc.show()
```

```
+----+-----------+
| Ano|ocorrencias|
+----+-----------+
|2001|      50996|
|2002|      50086|
|2003|      55342|
|2004|      58500|
|2005|      57266|
|2006|      55966|
|2007|      55936|
|2008|      45876|
|2009|      42708|
|2010|      44344|
|2011|      38916|
|2012|      36490|
|2013|      34778|
|2014|      29752|
|2015|      21986|
|2016|      13988|
|2017|      11902|
|2018|       5760|
+----+-----------+
```

Quantidade de crimes por ano que sejam do tipo NARCOTICS e tenham ocorrido em dias pares?

- Primeiro defini meu target(ano)
- Depois passei um count(*) para pegar todas as ocorrencias
- Como Condicional passei o tipo da ocorrencias NARCOTICS
- Agrupando por Ano e Ordenando por Ano
- Passei o boolean type AND com o Parametro Dia % 2 == 0

In [14]:
```python
sql = """SELECT Ano, COUNT(*) as ocorrencias FROM ocorrencias WHERE Tipo = 'NARCOTICS' AND (Dia %
 2) == 0  GROUP BY Ano  ORDER BY Ano """

CrimesPorAnoNarcParDay = spark.sql(sql)
```

In [15]:
```python
CrimesPorAnoNarcParDay.show()
```

```
+----+-----------+
| Ano|ocorrencias|
+----+-----------+
|2001|      24732|
|2002|      24610|
|2003|      26726|
|2004|      28296|
|2005|      27610|
|2006|      27648|
|2007|      27736|
|2008|      22302|
|2009|      20894|
|2010|      21632|
|2011|      19330|
|2012|      17884|
|2013|      16822|
|2014|      14644|
|2015|      10746|
|2016|       6690|
|2017|       5732|
|2018|       2678|
+----+-----------+
```

Mês com maior ocorrência de crimes?

Passei o Parametro Mes dentro da subquery contando o numero de linhas dos Mes e chamando de ocorrencias com isso agrupei por Mes e ordenei para apresentar de ordem descendente, limitando apenas 1 unico valor de saida

In [16]:
```python
sql = "select Mes from (select Mes, count(Mes) as ocorrencias from ocorrencias group by Mes order b
y 'ocorrencias' desc) LIMIT 1"

MesMaiorOcorrencia = spark.sql(sql)
MesMaiorOcorrencia.show()
```

```
+---+
|Mes|
+---+
|  7|
+---+
```

Mês com a maior média de ocorrência de crimes?

Aqui fiz 4 querys

- 1 - Para agrupar por mes a media score das ocorrencias
- 2 - A segunda pega o maximo valor dessa avg_ocorrencias
- 3 - Faz um select de mes, AVG(total_linhas) agrupando por Mes
- 4 - Ultima query ele pega os Valores Maximos de Mes, e Avg_ocorre, usando condicional where para passar a segunda query com condição para avg_ocorre = MAX valor da média

In [17]:
```python
sum_count = df.count()
```

In [18]:
```python
print(sum_count)
```

```
13157184
```

In [19]:
```python
sql = "SELECT Mes, avg_ocorre  FROM (SELECT Mes, AVG(13157184) AS avg_ocorre FROM ocorrencias GROUP
 BY Mes)   WHERE avg_ocorre = (SELECT MAX(avg_ocorre) FROM (SELECT Mes, AVG(13157184) AS avg_ocorre
 FROM ocorrencias GROUP BY Mes)) LIMIT 1"
spark.sql(sql).show()
```

```
+---+-----------+
|Mes| avg_ocorre|
+---+-----------+
|  7|1.3157184E7|
+---+-----------+
```

Mês por ano com a maior ocorrência de crimes?

- Aqui passei um groubBy(mes) para grupar por mes
- Em Seguida passei um agregador agg para o valor Max do Ano
- Limitando apenas uma unica saida

In [20]:
```python
df.groupBy("Mes").agg(F.max("Ano")).limit(1).show()
```

```
+---+--------+
|Mes|max(Ano)|
+---+--------+
|  7|    2017|
+---+--------+
```

Mês com a maior ocorrência de crimes do tipo DECEPTIVE PRACTICE?

Aqui pensei que poderia rodar um test checando pelo pyspark e tambem com sql para validar as duas saidas

A Primeira DF_DECEPTIVE_PRACTICE passei um filter para pegar apenas os tipos DECEPTIVE PRACTICE

Em Seguida agrupei Mes e Ano usando agregação para contar o as vezes que os dados da coluna mes aparece chamando de Mes_ocorr

In [21]:
```python
DF_DECEPTIVE_PRACTICE = df.filter(df['Tipo']  == "DECEPTIVE PRACTICE")
DF_DECEPTIVE_PRACTICE = DF_DECEPTIVE_PRACTICE.groupby(['Mes', 'Ano']).agg(F.count("Mes").alias("Mes
_Ocorr"))
```

Com a DF criado passei os parametros MAX em todas as dimenções de atenção da tabela(Mes, Ano, Mes_Ocorr)

In [22]:
```python
value = DF_DECEPTIVE_PRACTICE.agg({"Mes_Ocorr": "max","Mes":"max","Ano":"Max"}).collect()
print(value)
```

```
[Row(max(Ano)='2018', max(Mes_Ocorr)=1824, max(Mes)='9')]
```

Validação com SQL

Com o dado do dataframe pensei em passar pelo sql para validar a saida dos dados pela query a baixo

Passei um MAX(mes, Mes_, Ano) e em uma subquery passei os parametros e Mes,Ano e Count(Mes)

Usando WHERE passei o tipo do dado buscado e agrupei por Mes e Ano

In [23]:
```python
sql = """
select max(Mes) as Mes_, max(Ano) as Ano_, max(Mes_ocorr) from
(select Mes,Ano,count(Mes) as Mes_ocorr FROM ocorrencias WHERE Tipo = 'DECEPTIVE PRACTICE' GROUP BY
 Mes, Ano)

"""
spark.sql(sql).show()
```

```
+----+----+--------------+
|Mes_|Ano_|max(Mes_ocorr)|
+----+----+--------------+
|   9|2018|          1824|
+----+----+--------------+
```

Dia do ano com a maior ocorrência de crimes?

PS: Por falha e erro jo job não consegui rodar essa query

Contudo a ideia era agrupar Mes, Ano e Dia e ordenar por ocorrencias e com isso fazer um count do mes

In [25]:
```python
sql = """
  SELECT Mes, Ano, Dia  FROM (
        SELECT Mes,
        COUNT(Mes) as ocorrencias,
        Ano,
        Dia
        FROM ocorrencias  GROUP BY Mes, Ano, Dia
        ORDER BY 'ocorrencias' DESC) LIMIT 1
"""
spark.sql(sql).show()
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-25-1b7c0c0da4b4> in <module>
      8         ORDER BY 'ocorrencias' DESC) LIMIT 1
      9 """
---> 10 spark.sql(sql).show()
     11 
     12 

~/.local/lib/python3.7/site-packages/pyspark/sql/dataframe.py in show(self, n, truncate, vertical)
    438         """
    439         if isinstance(truncate, bool) and truncate:
--> 440             print(self._jdf.showString(n, 20, vertical))
    441         else:
    442             print(self._jdf.showString(n, int(truncate), vertical))

~/.local/lib/python3.7/site-packages/py4j/java_gateway.py in __call__(self, *args)
   1303         answer = self.gateway_client.send_command(command)
   1304         return_value = get_return_value(
-> 1305             answer, self.gateway_client, self.target_id, self.name)
   1306 
   1307         for temp_arg in temp_args:

~/.local/lib/python3.7/site-packages/pyspark/sql/utils.py in deco(*a, **kw)
    126     def deco(*a, **kw):
    127         try:
--> 128             return f(*a, **kw)
    129         except py4j.protocol.Py4JJavaError as e:
    130             converted = convert_exception(e.java_exception)

~/.local/lib/python3.7/site-packages/py4j/protocol.py in get_return_value(answer, gateway_client, t
arget_id, name)
    326                 raise Py4JJavaError(
    327                     "An error occurred while calling {0}{1}{2}.\n".
--> 328                     format(target_id, ".", name), value)
    329             else:
    330                 raise Py4JError(

Py4JJavaError: An error occurred while calling o84.showString.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 15 in stage 40.0 failed 1
 times, most recent failure: Lost task 15.0 in stage 40.0 (TID 1718, sandbox-host.hortonworks.com,
 executor driver): java.io.FileNotFoundException: /tmp/blockmgr-ff9c78ab-fdb4-43b1-846f-2de9dc87e1a
6/14/temp_shuffle_4c9dfea7-cb4a-4cad-872b-5bbaf1d38b7d (Too many open files)
	at java.io.FileOutputStream.open0(Native Method)
	at java.io.FileOutputStream.open(FileOutputStream.java:270)
	at java.io.FileOutputStream.<init>(FileOutputStream.java:213)
	at org.apache.spark.storage.DiskBlockObjectWriter.initialize(DiskBlockObjectWriter.scala:10
5)
	at org.apache.spark.storage.DiskBlockObjectWriter.open(DiskBlockObjectWriter.scala:118)
	at org.apache.spark.storage.DiskBlockObjectWriter.write(DiskBlockObjectWriter.scala:245)
	at org.apache.spark.shuffle.sort.BypassMergeSortShuffleWriter.write(BypassMergeSortShuffleW
riter.java:158)
	at org.apache.spark.shuffle.ShuffleWriteProcessor.write(ShuffleWriteProcessor.scala:59)
	at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:99)
	at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:52)
	at org.apache.spark.scheduler.Task.run(Task.scala:127)
	at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:446)
	at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1377)
	at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:449)
	at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
	at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
	at java.lang.Thread.run(Thread.java:748)

Driver stacktrace:
	at org.apache.spark.scheduler.DAGScheduler.failJobAndIndependentStages(DAGScheduler.scala:2
059)
	at org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2(DAGScheduler.scala:2008)
	at org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2$adapted(DAGScheduler.scal
a:2007)
	at scala.collection.mutable.ResizableArray.foreach(ResizableArray.scala:62)
	at scala.collection.mutable.ResizableArray.foreach$(ResizableArray.scala:55)
	at scala.collection.mutable.ArrayBuffer.foreach(ArrayBuffer.scala:49)
	at org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:2007)
	at org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1(DAGScheduler.scal
a:973)
	at org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1$adapted(DAGSchedu
ler.scala:973)
	at scala.Option.foreach(Option.scala:407)
	at org.apache.spark.scheduler.DAGScheduler.handleTaskSetFailed(DAGScheduler.scala:973)
	at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.doOnReceive(DAGScheduler.scala:2
239)
	at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:218
8)
	at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:217
7)
	at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)
	at org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:775)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2099)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2120)
	at org.apache.spark.SparkContext.runJob(SparkContext.scala:2139)
	at org.apache.spark.sql.execution.SparkPlan.executeTake(SparkPlan.scala:467)
	at org.apache.spark.sql.execution.SparkPlan.executeTake(SparkPlan.scala:420)
	at org.apache.spark.sql.execution.CollectLimitExec.executeCollect(limit.scala:47)
	at org.apache.spark.sql.Dataset.collectFromPlan(Dataset.scala:3627)
	at org.apache.spark.sql.Dataset.$anonfun$head$1(Dataset.scala:2697)
	at org.apache.spark.sql.Dataset.$anonfun$withAction$1(Dataset.scala:3618)
	at org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId$5(SQLExecution.
scala:100)
	at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQLExecution.scala:16
0)
	at org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId$1(SQLExecution.
scala:87)
	at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:764)
	at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.scala:64)
	at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3616)
	at org.apache.spark.sql.Dataset.head(Dataset.scala:2697)
	at org.apache.spark.sql.Dataset.take(Dataset.scala:2904)
	at org.apache.spark.sql.Dataset.getRows(Dataset.scala:300)
	at org.apache.spark.sql.Dataset.showString(Dataset.scala:337)
	at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
	at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
	at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
	at java.lang.reflect.Method.invoke(Method.java:498)
	at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
	at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357)
	at py4j.Gateway.invoke(Gateway.java:282)
	at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
	at py4j.commands.CallCommand.execute(CallCommand.java:79)
	at py4j.GatewayConnection.run(GatewayConnection.java:238)
	at java.lang.Thread.run(Thread.java:748)
Caused by: java.io.FileNotFoundException: /tmp/blockmgr-ff9c78ab-fdb4-43b1-846f-2de9dc87e1a6/14/tem
p_shuffle_4c9dfea7-cb4a-4cad-872b-5bbaf1d38b7d (Too many open files)
	at java.io.FileOutputStream.open0(Native Method)
	at java.io.FileOutputStream.open(FileOutputStream.java:270)
	at java.io.FileOutputStream.<init>(FileOutputStream.java:213)
	at org.apache.spark.storage.DiskBlockObjectWriter.initialize(DiskBlockObjectWriter.scala:10
5)
	at org.apache.spark.storage.DiskBlockObjectWriter.open(DiskBlockObjectWriter.scala:118)
	at org.apache.spark.storage.DiskBlockObjectWriter.write(DiskBlockObjectWriter.scala:245)
	at org.apache.spark.shuffle.sort.BypassMergeSortShuffleWriter.write(BypassMergeSortShuffleW
riter.java:158)
	at org.apache.spark.shuffle.ShuffleWriteProcessor.write(ShuffleWriteProcessor.scala:59)
	at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:99)
	at org.apache.spark.scheduler.ShuffleMapTask.runTask(ShuffleMapTask.scala:52)
	at org.apache.spark.scheduler.Task.run(Task.scala:127)
	at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:446)
	at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1377)
	at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:449)
	at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
	at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
	... 1 more
```

In [ ]:

In [ ]: