Criando pipelines de dados eficientes com Spark e Python

- ## Primeiro contato com Databricks

- ## Criando um sparkContext

```
from pyspark.sql import SparkSession

spark_session = SparkSession.builder.enableHiveSupport().getOrCreate() > ######
#getOrCreate() pega uma sessão já existente
```

- ## Duas maneiras de acessar o contexto do spark a partir da sessão spark

```
spark_context = spark_session._sc

spark_context = spark_session.sparkContext
```

- ## Executa uma tarefa que pega o timestamp e imprime a cada segundo

```
from pyspark.streaming import StreamingContext

ssc = StreamingContext(sc, 1)

lines = ssc.socketTextStream('localhost', 9999)

counts = lines.flatMap(lambda line: line.split(" ")).map(lambda word: (word, > ######
1)).reduceByKey(lambda a, b: a+b)

counts.pprint()

ssc.start()

ssc.awaitTermination()
```

- ## Spark SQL

- Path - dataset1

```
path_dataset1 = "/FileStore/tables/country_vaccinations.csv"
```

- Path - RDD

```
path_rdd = "/FileStore/tables/arquivo_rdd.txt"
```

Leitura de Dataframe

- **Opção 1**

```
df1 =
spark.read.format("csv").option("header","true").option("inferSchema","false").load(p
ath_dataset1) # inferSchema permite ao spark adivinhar o data type
```

- **Opção 2**

```
df1 = spark.read.csv(path_dataset1)
```

```
df1 = spark.read.option("header","true").option("delimiter",
"|").option("inferSchema","true").csv(path_dataset1)
```

- **Outras formas de leitura de arquivos com PySpark**

```
path = "/../../arquivoXPTO"
```

- **Criando um dataframe a partir de um JSON**

```
dataframe = spark.read.json(path)
```

- **Criando um dataframe a partir de um ORC**

```
dataframe = spark.read.orc(path)
```

- **Criando um dataframe a partir de um PARQUET**

```
dataframe = spark.read.parquet(path)
```

Exibindo dataframe

```
df1.show(2)
```

```
#df1.dtypes
```

- **Salva ou sobreescreve caso exista o arquivo em formato parquet**

```
df1.write.format("parquet").mode("overwrite").save("/FileStore/tables/RAW_ZONE_PARQUET/")
```

- **Exibe 3 linhas do dataframe**

```
df1.take(3)

df1.take(1)[0]
```

- **Imprimindo tipos de campos**

```
df1.dtypes

#df1.printSchema
```

- ## Outros tipos de SELECT

- **Diferentes formas de selecionar uma coluna**

```
from pyspark.sql.functions import *

df1.select("country").show(5)

df1.select('country').show(5)

df1.select(col("country")).show(5)

df1.select(column("country")).show(5)

df1.select(expr("country")).show(5)
```

- ## Leitura de um RDD

```
rdd = sc.textFile(path_rdd)

#rdd.show() = Errado, não é possível exibir um SHOW() de um RDD, somente um Dataframe

rdd.collect()
```

- **Desta maneira cria um arquivo posicional de uma unica coluna**

```
dfff = spark.read.format("csv").load(path_rdd)
```

```
display(dfff)
```

- ## Criando uma tabela temporária

```
nome_tabela_temporiaria = "tempTableTerere"

df1.createOrReplaceTempView(nome_tabela_temporiaria)
```

- **Lendo a tabela temporaria opcao 1**

```
spark.read.table(nome_tabela_temporiaria).show()
```

- **Executando SQL ANSI no Spark e atribui a um dataframe chamado "dfterere"**

```
dfterere = spark.sql("SELECT count(*) tt, country FROM tempTableTerere Group By country")

dfterere.dtypes
```

- ## Visualização do Databricks

```
display(spark.sql("SELECT * FROM tempTableDataFrame1"))
```

- ## Usando functions

- **Scala**

```
#import org.apache.spark.sql.functions._
```

- **Python**

```
from pyspark.sql.functions import col, column
```

- **Usando function col ou column**

```
df1.select(col("country"), col("date"), column("iso_code")).show()
```

- **SELECT usando expressions**

```
df1.selectExpr("country", "date", "iso_code").show()
```

```
Scala import

org.apache.spark.sql.types._
```

- ## Criando um Schema manualmente no PySpark

```
from pyspark.sql.types import *

dataframe_ficticio = StructType([

StructField("col_String_1", StringType()),

StructField("col_Integer_2", IntegerType()),

StructField("col_Decimal_3", DecimalType())

])

dataframe_ficticio
```

- ## Função para gerar Schema (campos/colunas/nomes de colunas)

- **Scala**

```
org.apache.spark.sql.types._

def getSchema(fields : Array[StructField]) : StructType = {

new StructType(fields)

}
```

- **PySpark**

```
def getSchema(fields):

return StructType(fields)

schema = getSchema([StructField("coluna1", StringType()), StructField("coluna2",
StringType()), StructField("coluna3", StringType())])
```

- **Gravando um novo CSV**

```
path_destino="/FileStore/tables/CSV/"

nome_arquivo="arquivo.csv"

path_geral= path_destino + nome_arquivo

df1.write.format("csv").mode("overwrite").option("sep", "\t").save(path_geral)
```

- **Gravando um novo JSON**

```
path_destino="/FileStore/tables/JSON/"

nome_arquivo="arquivo.json"

path_geral= path_destino + nome_arquivo

df1.write.format("json").mode("overwrite").save(path_geral)
```

- **Gravando um novo PARQUET**

```
path_destino="/FileStore/tables/PARQUET/"

nome_arquivo="arquivo.parquet"

path_geral= path_destino + nome_arquivo

df1.write.format("parquet").mode("overwrite").save(path_geral)
```

- **Gravando um novo ORC**

```
path_destino="/FileStore/tables/ORC/"

nome_arquivo="arquivo.orc"

path_geral= path_destino + nome_arquivo

df1.write.format("orc").mode("overwrite").save(path_geral)
```

- **Define uma nova coluna com um valor constante**

```
df2 = df1.withColumn("nova_coluna", lit(1)) # cria uma coluna de nome "nova_coluna" e atribui um valor constante de "1"

#display(df1)

#display(df2)
```

- ## Adicionar coluna

```
teste = expr("total_vaccinations < 40")

#df1.select("country", "total_vaccinations").withColumn("teste", teste).show(5)
```

- ## Renomear uma coluna

```
df1.select(expr("total_vaccinations as total_de_vacinados")).show(5)

df1.select(col("country").alias("pais")).show(5)

df1.select("country").withColumnRenamed("country", "pais").show(5)
```

- ## Remover uma coluna

```
df3 = df1.drop("country")

df3.columns
```

- ## Filtrando dados e ordenando

where() é um alias para filter().

- **Seleciona apenas os primeiros registros da coluna "total_vaccinations"**

```
df1.filter(df1.total_vaccinations > 55).orderBy(df1.total_vaccinations).show(2)
```

- ## Filtra por país igual Argentina

```
df1.select(df1.total_vaccinations, df1.country).filter(df1.country ==
"Argentina").show(5)
```

- ## Filtra por país diferente Argentina

```
df1.select(df1.total_vaccinations, df1.country).where(df1.country !=
"Argentina").show(5) # python type
```

- ## Mostra valores únicos

```
df1.select("country").distinct().show()
```

- **Especificando vários filtros em comando separados**

```
filtro_vacinas = df1.total_vaccinations < 100

filtro_pais = df1.country.contains("Argentina")

df1.select(df1.total_vaccinations, df1.country,
df1.vaccines).where(df1.vaccines.isin("Sputnik V",
"Sinovac")).filter(filtro_vacinas).show(5)

df1.select(df1.total_vaccinations, df1.country,
df1.vaccines).where(df1.vaccines.isin("Sputnik V",
"Sinovac")).filter(filtro_vacinas).withColumn("filtro_pais", filtro_pais).show(5)
```

- **Convertendo dados**

```
df5 = df1.withColumn("PAISSSSS", col("country").cast("string").alias("PAISSSSSSS"))

df5.select(df5.PAISSSSS).show(2)
```

- **Trabalhando com funções**

- **Usando funções**

```
df1.select(upper(df1.country)).show(3)

df1.select(lower(df1.country)).show(4)
```

- **Criando um dataframe genérico**

```
d = [{'name': 'Alice', 'age': 1}]

df_A = spark.createDataFrame(d)

df_A.show()

rdd1 = [{"nome": "Marco","idade": 33,"status": 'true'},

{"nome": "Antonio","idade":33,"status": 'true'},

{"nome":"Pereira","idade":33,"status": 'true'},

{"nome":"Helena","idade":30,"status": 'true'},

{"nome":"Fernando","idade":35,"status": 'true'},
```

```
{"nome":"Carlos","idade":28,"status": 'true'},
{"nome":"Lisa","idade":26,"status": 'true'},
{"nome":"Candido","idade":75,"status": 'false'},
{"nome":"Vasco","idade":62,"status": 'true'}
]

dff1 = spark.createDataFrame(rdd1)
dff1.show()
```

```
rdd2 = [
{"nome":"Marco","PaisOrigem":"Brasil"},
{"nome":"Helena","PaisOrigem":"Brasil"},
{"nome":"Gabriel","PaisOrigem":"Brasil"},
{"nome":"Vasco","PaisOrigem":"Portugal"},
{"nome":"Medhi","PaisOrigem":"Marocco"}]

dff2 = spark.createDataFrame(rdd2)
dff2.show()
```

- ## Fazendo Join no Spark

```
join_type = "inner"
join_condition = dff1.nome == dff2.nome
df3 = dff1.join(dff2, join_condition, join_type)
df3.show()
#df1.groupBy("status").agg(countDistinct(col("idade"))).show()
```

- ## Path - RDD

```
path_rdd = "/FileStore/tables/arquivo_rdd.txt"
rdd = sc.textFile(path_rdd)
```

```
df_pre = spark.read.text(path_rdd)
from pyspark.sql import functions as f
df_pre
x = lambda y : y + 1
```

- **Para TXT's com header**

```
path_rdd = "/FileStore/tables/arquivo_rdd.txt" # especificar o caminho do Bucket
df_pre = spark.read.text(path_rdd)
posicao = ((0,1),(1,5), (5,8))
header= "nome;tipo;texto"
func = lambda p,name,df : df.withColumn(name, df['value'].substr(p[0], p[1]))
def concatenaCampo(posicao, header, df_):
i = 0
header_ = header.split(";")
for p in posicao:
df_ = func(p,header[i],df)
print(header_[i])
i=i+1

return df_.drop('value')
header.split(";")
dd = concatenaCampo(posicao, header, df_pre)
dd.show()
```

- **Link para o notebook na databricks**

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5191291087587071/3239850342304675/2520349622255950/latest.html