

Resumo de Atividades da Semana

Exercício 01: Crie um array de 250 ints e aplique o método reverse para inverter o conteúdo.

```
data = np.random.randint(1000, size=250)
pandas_df = pd.DataFrame(data, columns=['numbers'])
df = spark.createDataFrame(pandas_df)
df.select(df.numbers, f.reverse(df.numbers)).show()
```

```
+-----+-----+
|numbers|reverse(numbers)|
+-----+-----+
|    921|             129|
|     89|              98|
|    513|             315|
|    548|             845|
|    511|             115|
|    988|             889|
|    879|             978|
|    115|             511|
|    750|             057|
|    111|             111|
|    110|             011|
|    654|             456|
|     82|              28|
|    635|             536|
|    428|             824|
|    133|             331|
|    338|             833|
|    499|             994|
|    299|             992|
|    514|             415|
+-----+-----+
only showing top 20 rows
```

Na primeira linha uso o numpy para criar um array de 250 ints aleatoriamente, de 0 a 1000. Após isso é convertido para um dataframe com a coluna de números e depois é utilizado a função reverse para inverter o conteúdo do dataframe. Logo em sequência é imprimido o resultado, com os números normais e os invertidos.

Exercício 02: Crie uma lista de 20 animais, ordene-os e itere para imprimi-las individualmente cada um deles. Depois salve num arquivo texto em formato CSV.

```
df = spark.createDataFrame([
    ['Vaca'], ['Cachorro'], ['Gato'], ['Touro'], ['Galinha'],
    ['Cobra'], ['Girafa'], ['Rinoceronte'], ['Macaco'], ['Ornitorrinco'],
    ['Leão'], ['Tigre'], ['Papagaio'], ['Tubarão'], ['Pavão'],
    ['Leopardo'], ['Avestruz'], ['Aranha'], ['Abelha'], ['Vespa']
], schema=['Animal'])

animal_df = df.orderBy('Animal')

for line in animal_df.collect():
    print(line)

animal_df.show()
animal_df.coalesce(1).write.mode('overwrite').csv('animal')
```

Inicialmente foi criado um DataFrame manual com diversos animais. Após isso o DataFrame foi ordenado com a função “orderBy”, após isso foi percorrido cada linha depois de coletar todos os dados. Por fim foi escrito e salvo em um arquivo .csv.

Resultado:

```
Row(Animal='Abelha')
Row(Animal='Aranha')
Row(Animal='Avestruz')
Row(Animal='Cachorro')
Row(Animal='Cobra')
Row(Animal='Galinha')
Row(Animal='Gato')
Row(Animal='Girafa')
Row(Animal='Leopardo')
Row(Animal='Leão')
Row(Animal='Macaco')
Row(Animal='Ornitorrinco')
Row(Animal='Papagaio')
Row(Animal='Pavão')
Row(Animal='Rinoceronte')
Row(Animal='Tigre')
Row(Animal='Touro')
Row(Animal='Tubarão')
Row(Animal='Vaca')
Row(Animal='Vespa')
+-----+
|   Animal|
+-----+
|   Abelha|
|   Aranha|
| Avestruz|
| Cachorro|
|   Cobra|
|   Galinha|
|    Gato|
|   Girafa|
| Leopardo|
|    Leão|
|   Macaco|
|Ornitorrinco|
|   Papagaio|
|    Pavão|
| Rinoceronte|
|    Tigre|
|    Touro|
|   Tubarão|
|    Vaca|
|    Vespa|
+-----+
```

Dentro do CSV:

```
1 Abelha
2 Aranha
3 Avestruz
4 Cachorro
5 Cobra
6 Galinha
7 Gato
8 Girafa
9 Leopardo
10 Leão
11 Macaco
12 Ornitorrinco
13 Papagaio
14 Pavão
15 Rinoceronte
16 Tigre
17 Touro
18 Tubarão
19 Vaca
20 Vespa
```

Exercício 03: Executar o Laboratório: “Gerar dados processar e criar arquivo texto”

```
import random
import time
import os
import names
```

```
t0 = time.time()

random.seed(89)
```

Configurações iniciais do arquivo, importações.

```
qtde_nomes_unicos = 3000
qtde_nomes_aleatorios = 1000000

print("Criando conjunto de dados com {} nomes".format(qtde_nomes_aleatorios))
```

Criando conjunto de dados com 1000000 nomes

Criado conjunto de 3000 nomes únicos e de 1000000 nomes aleatórios.

```
aux = []
dados = []

for i in range(0, qtde_nomes_unicos):
    aux.append(names.get_full_name()) # Aqui coloca no array nomes aleatorios

for i in range(0, qtde_nomes_aleatorios):
    dados.append(random.choice(aux)) # embaralhando valores
```

Criando um array auxiliar para colocar um nome e um array de dados para embaralhar os valores dos nomes.

`dados`

```
['Tamara Thompson',  
'Gary Wentworth',  
'Richard Roper',  
'Brenda Hollins',  
'Stephanie Tyson',  
'Raymond Sweigart',  
'Ernest Alloway',  
'Mary Storm',  
'Anne Brown',  
'Malissa Walther',  
'Amy Fox',  
'Antony Lauzon',  
'Christopher Little',  
'William Helbert',  
'Daisy Blocker',  
'Janet Bell',  
'Adrienne Spence',  
'Angela Ehmman',  
'Brandon Joachim',
```

Prévia do array de dados

```
print("Gravando em arquivo")  
  
arquivo = open('nomes_aleatorios.txt', 'w')  
  
for item in dados:  
    arquivo.write(item + '\n')  
  
arquivo.close()  
  
tf = time.time() - t0  
  
print("Criacao finalizada em {} segundos".format(tf))
```

```
Gravando em arquivo  
Criacao finalizada em 425.72044229507446 segundos
```

Escrever array de dados em um arquivo externo, e salvar este arquivo (nomes_aleatorios.txt).

```
1 Tamara Thompson
2 Gary Wentworth
3 Richard Roper
4 Brenda Hollins
5 Stephanie Tyson
6 Raymond Sweigart
7 Ernest Alloway
8 Mary Storm
9 Anne Brown
10 Malissa Walther
11 Amy Fox
12 Antony Lauzon
13 Christopher Little
14 William Helbert
15 Daisy Blocker
16 Janet Bell
17 Adrienne Spence
18 Angela Ehmann
19 Brandon Joachim
20 Lucrecia Flores
21 Vincent Landers
22 Marion Taylor
23 Toney Slater
24 Joe Slocum
25 June Ogden
26 Jamie Fagan
27 Darby Santi
28 Edward Kulish
29 Jerry Boes
30 Patricia Leger
```

Exercício 04: Realize a execução de código de cada exemplo e exercício dos itens anteriores via spark-submit

```
spark-submit --class org.apache.spark.examples.SparkPi --master local
~/apps/spark-3.1.2-bin-hadoop3.2/examples/jars/spark-examples_2.12-3.1.2.jar 100
```

```
INFO SparkContext: Running Spark version 3.1.2
INFO ResourceUtils: =====
INFO ResourceUtils: No custom resources configured for spark.driver.
INFO ResourceUtils: =====
INFO SparkContext: Submitted application: Spark Pi
```

```
spark-submit --class org.apache.spark.examples.SparkPi --master
spark://207.184.161.138:7077 --executor-memory 20G --total-executor-cores 100
~/apps/spark-3.1.2-bin-hadoop3.2/examples/jars/spark-examples_2.12-3.1.2.jar 1000
```

```
spark-submit --master local transform.py --src
/home/rafael_ignaulin/Desktop/COMPASSO/Sprint_5/week_09/twitter_raw.csv --dest
/home/rafael_ignaulin/Desktop/COMPASSO/Sprint_5/week_09/export_parquet
```

*OBS: Script rodado para rodar o código do exercício 05 (Caso de Uso)

Exercício 05: Desafio XPTO: Carregar os dados da RAW Zone para a REF Zone, unificando os dados num Bucket S3 persistindo no formato parquet. Particionar os dados por Ano/Mês/Dia conforme a criação do Twitter. Incluir 2 colunas novas:

- Sentimento: indicando Positivo quando encontrar algum símbolo como :D ou :) ou :] etc; Indicando Negativo quando encontrar algum símbolo como :(ou :[ou :{ etc. Indicando Neutro, quando o tweet não tiver nenhum dos símbolos analisados. Se um tweet tem vários símbolos apenas o primeiro encontrado deve ser utilizado.
- Símbolo: Nesta coluna você deve adicionar o símbolo encontrado no tweet. Se um tweet tiver vários símbolos, apenas o primeiro encontrado deve ser utilizado.

Explicando o Algoritmo:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Spark Twitter Transformation"
    )
    parser.add_argument("--src", required=True)
    parser.add_argument("--dest", required=True)
    args = parser.parse_args()

    twitter_transform(args.src, args.dest)
```

- Parte inicial do algoritmo, utilizando a biblioteca padrão “argparse” conseguimos adicionar parâmetros de execução do trabalho. Neste caso, os dados de entrada e os dados de saída. A função é inicializada na última linha, passando os argumentos.

```
def twitter_transform(src, dest):
    with SparkSession.builder.appName("Twitter Transformation").getOrCreate() as spark:
        df = import_csv(spark, src)
        df2 = organize_date_values(df)
        df3 = create_columns_data(df2)
        export_parquet(df3, dest)
```

- Primeiramente, é iniciada uma sessão do Spark utilizando o comando SparkSession. Após isso, foi dividido cada etapa do processamento em 4 funções, que são elas:
 - Import CSV: Essa função faz a importação do CSV de origem.
 - Organização de datas: Essa função ajusta os chamados “Bad Data”, que são os dados com informações erradas ou faltantes.
 - Criação de colunas de dados: Essa função é a principal responsável pela criação de 3 colunas a mais, ajustando o dataframe.
 - Export Parquet: Função para exportar o arquivo em formato Parquet para uma saída.

```
def import_csv(spark, src):
    tweets = t.StructType([
        t.StructField("id", t.LongType(), False),
        t.StructField("text", t.StringType(), False),
        t.StructField("created_at", t.TimestampType(), False)
    ])
    if src is not None:
        df = spark.read.option("quotes", "").schema(tweets).csv(src)
    return df
```

- Nesta função, primeiramente é definido o schema (a tipagem dos dados) que vamos receber. Neste caso temos um id como um long int, o texto em si no formato string e a data de publicação no formato TimeStamp.
- Após isso fazemos uma verificação simples se o argumento existe, e depois fazemos a importação do arquivo de origem para o spark com o comando Read, passando a tipagem dos dados.

```
def organize_date_values(df):
    df2 = df.filter(df.created_at > '2018-01-01')\
            .filter(df.created_at < '2020-01-01')
    return df2
```

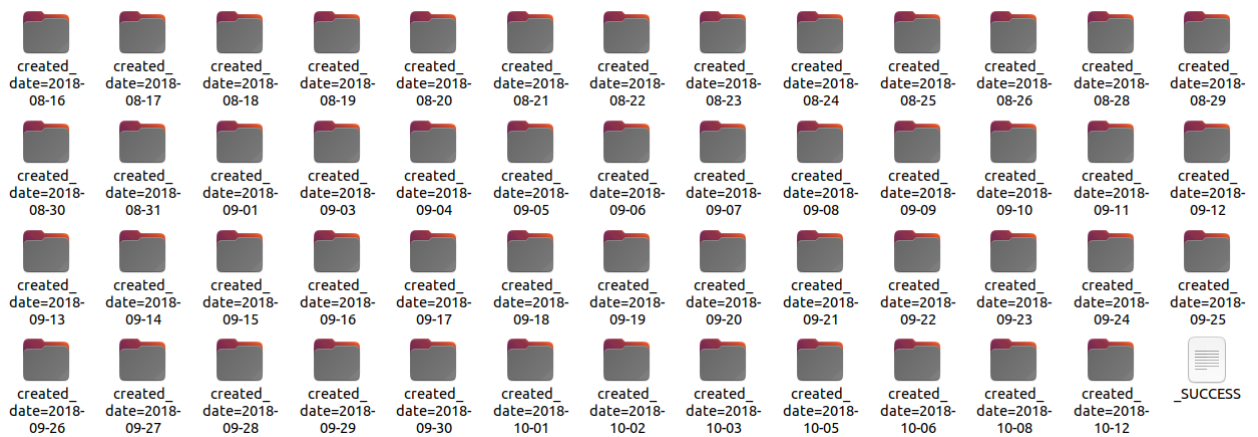
- O próximo passo é ajustar os dados com problemas ou faltantes, os chamados Bad Datas. Neste caso em específico, percebemos que existem poucos dados com datas completamente inválidas, então decidimos filtrar as datas que são válidas e que serão usadas para o processamento futuramente.

```
def create_columns_data(df):
    df2 = df\
        .withColumn("created_date", f.to_date("created_at")).repartition("created_date")\
        .withColumn("sentimento",
            f.when((df.text.contains(":D")) | (df.text.contains(":)")) | (df.text.contains(":]") | (df.text.contains(":P")), "Positivo")
            .when((df.text.contains("D:") | (df.text.contains(":(") | (df.text.contains(":[") | (df.text.contains(":("), "Negativo")
            .otherwise("Neutro"))\
        .withColumn("simbolo",
            f.when(df.text.contains(":D"), ":D")
            .when(df.text.contains(":)"), ":)")
            .when(df.text.contains(":]"), ":]")
            .when(df.text.contains(":P"), ":P")
            .when(df.text.contains("D:"), "D:")
            .when(df.text.contains(":("), ":(")
            .when(df.text.contains(":["), ":[")
            .otherwise(":|"))
    return df2
```

- Este é o passo de processamento, com a adição de novas colunas. Primeiramente precisamos ajustar as colunas que possuem o formato de data e hora para um formato com apenas datas (para agrupar apenas pela data da postagem, e não pelo horário), que foi definido e particionado como “created_date”.
- Segundamente, criamos a coluna de sentimento, onde é identificado se os tweets contém determinado símbolo e é definido uma classificação de sentimento para o tweet. Símbolos felizes como (:D , :)) são considerados como Positivos, enquanto símbolos tristes como (D: , :() são considerados como Negativos . Os textos que não aparecem no símbolo são considerados Neutros.
- Finalmente, criamos uma coluna contendo o símbolo que foi identificado, conforme os padrões dos símbolos comentados anteriormente.


```
def export_parquet(df, dest):
    df.write.mode("overwrite").partitionBy("created_date").parquet(dest)
```

- E por último, ocorre a exportação desses arquivos. Neste caso iremos exportar no formato Parquet, e particionado cada um deles pelo dia de criação (mostrado no created_date). Também é passado o destino, que foi configurado como um argumento no início do processo.



- Aqui está o resultado da execução do programa. Ele está particionado (separado em pastas) em cada um dos dias encontrados, e dentro de cada pasta existe um arquivo no formato Parquet contendo os dados em si.

Execução do script dentro do ambiente AWS:

- Utilizando EMR

Cluster: XPTO_CLUSTER Aguardando Cluster ready after last step completed.

Resumo Histórico do aplicativo Monitoramento Hardware Configurações Eventos Etapas Ações de bootstrap

Resumo

ID: j-28A716ITS7QVE

Data de criação: 2021-07-15 13:04 (UTC-3)

Tempo decorrido: 1 dia, 3 horas

Encerramento automático: Cluster waits

Proteção contra encerramento: Desativado [Alterar](#)

Tags: -- [Visualizar todas/Editar](#)

DNS público principal [ec2-44-193-223-33.compute-1.amazonaws.com](#) [Connect to the Master Node Using SSH](#)

Detalhes da configuração

Rótulo da versão: emr-6.1.0

Distribuição do Hadoop: Amazon

Aplicativos: Spark 3.0.0, Zeppelin 0.9.0

URI do log: s3://aws-logs-575556700570-us-east-1/elasticmapreduce/

Visualização consistente do EMRFS: Desativado

ID personalizado de AMI: --

- Criado um cluster para execução do script, utilizando o EMR 6.1 (spark 3.0)

Adicionar etapa

Tipo da etapa

Aplicativo do Spark

Nome

Twitter Transformation

Modo de implantação

Cluster

Opções de Spark-submit

Local do aplicativo*

s3://xpto-scripts/transform.py

Argumentos

--src s3://xptoraw/twitter_raw.csv
--dest s3://xpto-refined/batch/

Ação em caso de falha

Continuar

Execute o driver em um nó subordinado (modo de cluster) ou em um nó principal como um cliente externo (modo cliente).

Especifique outras opções para spark-submit.

Caminho para um arquivo JAR com o aplicativo e as dependências (modo de implantação de cliente é compatível apenas com um caminho local).

Especifique argumentos opcionais para o aplicativo.

O que fazer se a etapa falhar.

Cancelar

Adicionar

- Criação da etapa para execução do script dentro do Cluster EMR. Neste caso é uma aplicação SPARK, onde é apresentado o local do script, e os argumentos. Neste caso podemos passar diretamente os buckets como entrada e saída de dados, o AWS reconhecerá normalmente.

►	s-23ZWD8VSSMR2C	TwitterTransform	Concluído
---	-----------------	------------------	-----------

- Etapa concluída

Pelo AWS Glue:

Informações do crawler	
Nome	xpto_crawler
Tags	-

Datastores	
Datastore	S3
Incluir caminho	s3://xptoraw/batch
Connection	
Excluir padrões	

Função do IAM	
Função do IAM	arn:aws:iam::575556700570:role/xpto_processing_role

Programação	
Programação	Executar sob demanda

Saída	
Banco de dados	xpto_processing
Prefixo adicionados a tabelas (opcional)	
Usar um único esquema para cada caminho do S3	false
Table level (optional)	
▼ Opções de configuração	
Atualizações de esquema no datastore	Atualizar a definição da tabela no catálogo de dados.
Exclusão de objetos no datastore	Marcar a tabela como suspensa no catálogo de dados.

Voltar

Concluir

- Criação do Crawler para pegar os dados.

☐ [twitter_raw_csv](#)

xpto

s3://xptoraw/batch/twitter_raw.csv

- Com o crawler finalizado, criou essa tabela com o seu esquema dentro do banco de dados.

Caminho do script**Função do IAM** ⓘ

Certifique-se de que esta função tenha permissão para suas fontes, destinos, diretório temporário, scripts do Amazon S3 e quaisquer bibliotecas usadas pelo trabalho. [Criar função do IAM.](#)

Type**Glue version**

Glue Version must match for both ML Transform and Job in order to execute a job. Machine learning transforms are currently not supported for Glue 2.0.

Linguagem do ETL

☒ Python ☐ Scala

Diretório temporário

- Agora criamos um job para utilização do script pyspark, com algumas modificações.

```

import sys
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql import functions as f
from pyspark.sql import types as t

from awsglue.utils import getResolvedOptions
from awsglue.context import GlueContext
from awsglue.dynamicframe import DynamicFrame
from awsglue.job import Job

args = getResolvedOptions(sys.argv, ['JOB_NAME'])

spark_context = SparkContext.getOrCreate()
glue_context = GlueContext(spark_context)
session = glue_context.spark_session

job = Job(glue_context)
job.init(args['JOB_NAME'], args)

#Read movie data to Glue dynamic frame
dynamic_frame_read = glue_context.create_dynamic_frame.from_catalog(database = 'xpto', table_name = 'twitter_raw_csv')

#Convert dynamic frame to data frame to use standard pyspark functions
df = dynamic_frame_read.toDF()

#Arrumar Schema inicial
df = df.select(df.col0.alias('id'), df.col1.alias('text'), f.to_timestamp(df.col2, "yyyy-MM-dd HH:mm:ss").alias('created_at'))

#Filtrar Bad Data
df2 = df.filter(df.created_at > '2018-01-01')\
        .filter(df.created_at < '2020-01-01')

#Adicionar novas colunas
df3 = df2\
    .withColumn("created_date", f.to_date("created_at")).repartition("created_date")\
    .withColumn("sentimento",
        f.when((df.text.contains("D")) | (df.text.contains(":")) | (df.text.contains(":]")) | (df.text.contains(":P")), "Positivo")
        .when((df.text.contains("D:") | (df.text.contains(":(") | (df.text.contains(":[")), "Negativo")
        .otherwise("Neutro"))\
    .withColumn("sinbolo",
        f.when(df.text.contains("D:"), ":D")
        .when(df.text.contains(":)"), ":)")
        .when(df.text.contains(":]"), ":]")
        .when(df.text.contains(":P"), ":P")
        .when(df.text.contains("D:"), "D:")
        .when(df.text.contains(":("), ":(")
        .when(df.text.contains(":["), ":[")
        .otherwise(":|"))



















# WRITE TO S3 PATH
df_tweets_write = DynamicFrame.fromDF(df3, glue_context, "df_tweets_write")
glue_context.write_dynamic_frame.from_options(
    frame = df_tweets_write,
    connection_type = "s3",
    connection_options = {
        "path": 's3://xpto-refined/batch/',
        "partitionKeys": ["created_date"]
    }, format = "parquet")

job.commit()

```

- Foram feitas algumas modificações no código em relação a usar o schema do crawler, fazer um cast antes de acessar os dados utilizando Dynamic Frames (Glue)

* OBS: O código foi testado em um glue notebook, pela configuração via docker (para testar e não causar gastos desnecessários)

 created_date=2018-08-28/	Pasta
 created_date=2018-08-29/	Pasta
 created_date=2018-08-30/	Pasta
 created_date=2018-08-31/	Pasta
 created_date=2018-09-01/	Pasta
 created_date=2018-09-03/	Pasta
 created_date=2018-09-04/	Pasta
 created_date=2018-09-05/	Pasta
 created_date=2018-09-06/	Pasta
 created_date=2018-09-07/	Pasta
 created_date=2018-09-08/	Pasta
 created_date=2018-09-09/	Pasta
 created_date=2018-09-10/	Pasta
 created_date=2018-09-11/	Pasta
 created_date=2018-09-12/	Pasta
 created_date=2018-09-13/	Pasta
 created_date=2018-09-14/	Pasta
 created_date=2018-09-15/	Pasta

- Arquivos exportados da mesma forma, dentro do bucket do S3.