

# Spark 2 For Devs

MATERIAL DO LABORATÓRIO



## Lab 0.1: Acessando o Ambiente

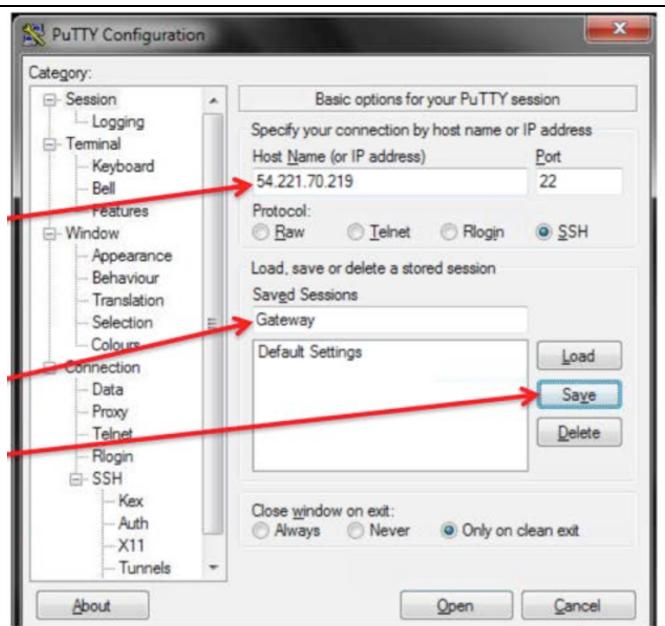
O sistema Operacional usado durante o treinamento será o Linux, sendo o mesmo acessível via ssh.

1. Ambiente de Laboratório
  - a. Foi enviado para o e-mail dos alunos as máquinas que cada um terá acesso para prática dos laboratórios.
  - b. Foi fornecido a cada aluno os keypair para acesso as máquinas de lab, sendo eles um arquivo .pem e um arquivo .ppk. O arquivo .pem é para que for acessar via ssh diretamente do prompt do seu SO. Para os alunos que forem usar o Putty deve ser usado o arquivo .ppk
2. Acessando via ssh:
  - a. **> ssh -o ServerAliveInterval=120 -i curso\_spark.pem ec2-user@<ip da sua maquina>**
3. Acessando via putty

1 – Se você ainda não tem o putty instalado, então faça o download do mesmo (<https://www.putty.org/>)

2 – No campo HostName insira o IP associado ao seu email e presente na planilha

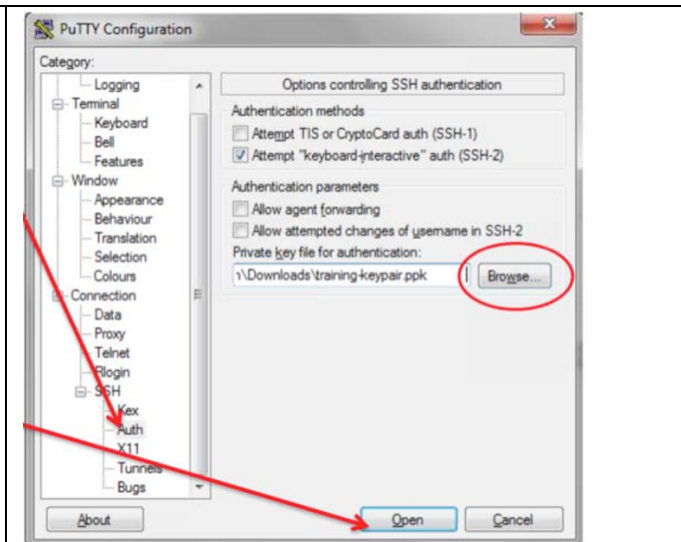
3 – Nomeie em Saved Sessions como Curso Spark e clique em Save



4 – Em configurações vá até a opção SSH -> Auth conforme figura ao lado e clique em Browse.

5 – Selecione o arquivo curso\_spark.ppk enviado por email

6 – Se for solicitado usuário deve ser informado:  
**ec2-user**



#### 4. Acessando o polynote

- a. Acessando <http://<seuip>:8192> deve ser possível acessar o polynote, verifique se o mesmo está acessível para você e comunique ao instrutor caso não.

## Lab 0.2: Scala Introdução

- Vamos iniciar o lab usando o REPL do scala para isso, entre na sua máquina de laboratório
- Estando no prompt do sistema operacional digite:  
> scala
- A partir desse momento você estará no REPL do Scala

```
Last login: Mon Sep  7 17:54:43 2020 from 177.33.128.7

  __|  __|_ )
  _| (    /   Amazon Linux 2 AMI
 ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-45-99 ~]$ scala
Welcome to Scala version 2.11.0 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_261).
Type in expressions to have them evaluated.
Type :help for more information.

scala> █
```

- Execute os passos abaixo:
  - Crie uma variável mutável e outra imutável e atribua um valor numérico inteiro a ela.
  - Repare que a variável não precisa ser declarada como Int para que funcione a atribuição devido ao Scala não exigir tipagem forte.
  - Crie uma variável **imutável fortemente tipada** como Int e atribua o valor 2.3 a ela. O que aconteceu?
  - Incremente em 1 as duas primeiras variáveis que você criou
  - Utilize a notação a seguir para novamente fazer um incremento de 2
- - scala> val n:Int = 1.+(2)
  - Qual foi o resultado? Você entendeu o que aconteceu?
- Execute os passos abaixo:
  - Crie uma variável do que represente um range de número de 1 a 100 e itere com for imprimindo aquelas que são pares.
  - Será que podemos iterar sobre ela de outra forma sem o comando for ou while?

### Lab 0.3: Coleções e Funções

- Vamos iniciar o lab usando o REPL do scala para isso, entre na sua máquina de laboratório
- Estando no prompt do sistema operacional digite:  
> scala
- A partir desse momento você estará no REPL do Scala
- Execute os passos seguintes:
  - Crie uma variável como sendo um range de 1 até 10
  - Crie uma variável como sendo uma função que verifica se o número é par
  - Recupere por meio da função head a cabeça do range, ou seja, apenas o valor 1
  - Recupere por meio da função tail o restante do range, ou seja, os valores diferentes de 1
  - [Opcional – Em casa se for fazer] Sabendo como funciona o head e o tail faça um função que imprime todos os items do range sem os métodos de iteração normal usando recursividade
  - Filtre o range e atribua a uma nova variável apenas os pares
  - Itere sobre a nova variável de lista de pares e crie uma nova calculando a potência de 2 para cada item
  - Agora reduza a variável de potência a soma delas

## Lab 2.1: Primeiro contato com o Spark

- Vamos iniciar nosso primeiro contato acessando o spark-shell
- Estando no prompt do sistema operacional digite:  
>spark-shell


```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://ip-172-31-45-99.us-east-2.compute.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1599678889208).
Spark session available as 'spark'.
Welcome to

  ____      _
 / ___|    / \
 \___ \  __/ _ \
  ___) |  / ___ \
 |_____| /_/___ \
          \____ \
              \___)
              version 2.4.6

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_261)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 
```

- Acesse <http://<sua>maquina>:4040>





JobsStagesStorageEnvironmentExecutors

### Spark Jobs (?)

**User:** ec2-user  
**Total Uptime:** 1,6 min  
**Scheduling Mode:** FIFO

▼ [Event Timeline](#)  
☐ Enable zooming

Executors	
 Added	
 Removed	

Jobs	
 Succeeded	
 Failed	
 Running	

	000	100	200	300	400
	19:14:48				

- Vá na aba EXECUTOR e veja qual é o Executor ID da tabela de Executors

- Veja quantos cores tem a sua máquina
- Na aba ENVIRONMENT e veja qual o conteúdo da seção Spark Properties
- Digite `sc.isLocal` e veja o resultado
- Digite `sc.<de um tab>` e veja o resultado

## Lab 2.2: Primeiro contato com o Spark

- Entre no spark-shell
- Carregue o arquivo “README.txt”  
`> val myfile = sc.textFile(“README.txt”)`
- Imprima o arquivo  
`> myfile.foreach(println)`
- Olhe para o Spark UI e identifique quantas task foram usadas para executar a tarefa (coluna Tasks da aba Jobs)? Esse número lhe diz algo?
- Quantas linhas tem o arquivo?  
`> myfile.count()`
- Pegue apenas as duas primeiras linhas e imprima  
`> myfile.take(2).foreach(println)`
- Pegue todas as linhas do arquivo  
`> myfile.collect()`
- Saia do spark  
`> :quit`

## Lab 3.1: RDD básico

Nesse lab vamos trabalhar com RDD manipulando vários arquivos de tamanho diferente e exercitando o que já vimos

Tempo aproximado: 30 – 40 minutos

- Copiamos para o diretório /home/ec2-user/lab da sua máquina todos os arquivos necessários para seguir com os laboratórios
- Entre na sua máquina e vá até o diretório /opt/polynote
- Agora vamos monitorar tudo o que iremos fazer no polynote  
> tail -f nohup.out
- Entre no polynote “http://<seuip>:8192” e configure para que seja possível rodarmos o spark nele.
- Na parte central do Polynote haverá uma seção chamada “Configuration & dependencies”, abra essa seção e vá até a sub-seção “Spark Config” entre com os seguintes valores:
  - spark.master e no campo seguinte local[\*]
  - **Clique em Save & Restart**
- Pode minimizar a seção de Configuração
- Agora vamos iniciar uma nova célula passando o mouse lentamente abaixo da seção até que apareça um sinal de + clique para iniciarmos
- Começaremos lendo o arquivo abaixo

```
val f = spark.sparkContext.textFile("/home/ec2-user/lab/twinkle/sample.txt")
```

- Vamos agora verificar quantas linhas com a palavra “twinkle” existe no dataset. Faremos isso usando a função filter.

```
val filtered = f.filter(line => line.contains("twinkle"))
```

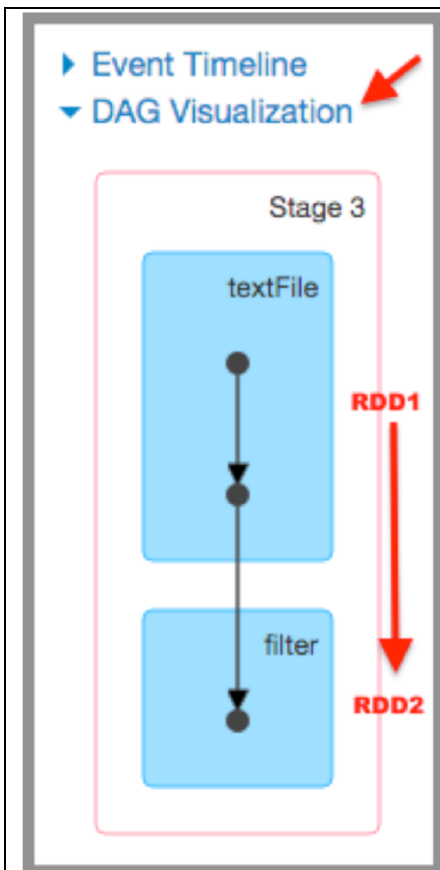
- **Quantas linhas vieram?**
  - Você pode usar a função count() sobre a variável filtered
- Vá para o Spark UI <http://<seuip>:4040>
- Clique na seção “Completed Jobs” no link “count at ..”



## ▼ Completed Jobs (1)

Job Id ▼	Description
0	count at Cell1:3 count at Cell1:3

- Visualize a DAG



- Crie uma nova célula
- Podemos mudar o comportamento do shell alterando o nível do log

Altere o nível do log para "INFO", e repare ao fazer o experimento que a definição da variável `filtered` não gera qualquer saída de log

```
spark.sparkContext.setLogLevel("INFO")
```

Crie a variável `filtered` novamente, observe o log (veja o console da sua máquina e não do polynote) e depois rode o `count()`

```
val filtered_2 = f.filter(line => line.contains("twinkle"))
```

- Você verá muito log no output apenas depois que o `count()` é executado

```
filtered_2.count()
```

- Por que o log só aparece depois do `count()` ?
- Volte o log para "WARN"

```
spark.sparkContext.setLogLevel("WARN")
```

- Vamos processar agora arquivos maiores e então ver o que acontece no Spark UI
- Crie uma nova célula (.. esperamos que você comece a entender qual o melhor momento para criar uma nova célula, repare que a

chamada de execução por célula gera uma unidade de processamento apenas daquela célula, porém, aproveitando as variáveis de células anteriores)

- Use o `spark.sparkContext` para carregar o arquivo `100M.data`
- Conte o número de linhas que tem a palavra “diamond”
- Verifique no Spark UI quantas tasks foram usadas

#### ▼ Completed Jobs (7)

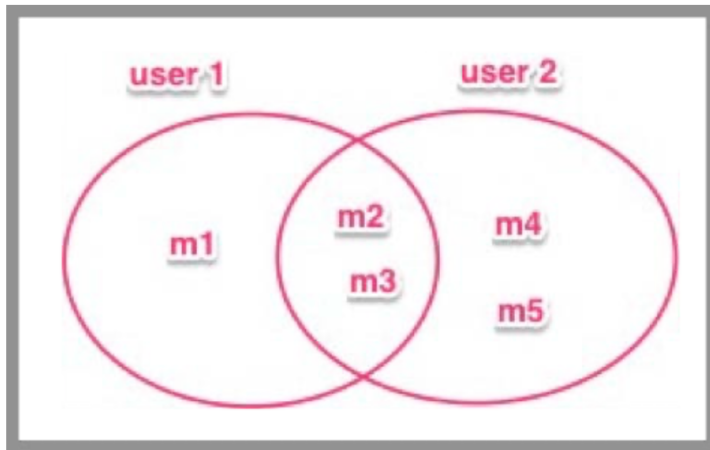
Job Id ▼	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	count at Cell3:3 <a href="#">count at Cell3:3</a>	2020/09/10 10:49:04	2 s	1/1	4/4

- Conte agora o número de linhas que **NÃO** tem a palavra “diamond”
- Agora verifique se o número de linhas que NÃO tem com os que tem equivale ao número total de linhas do arquivo
- Repita o mesmo processo para os arquivos `500M.data` e `1G.data`
  - Note o tempo e número de tasks que serão usados em cada um do caso
  - Clique no “link” count para ver detalhes dos Stages
  - Repare em como o arquivo vou quebrado na coluna “Input Size / Record” da seção Tasks
- Vamos carregar agora múltiplos arquivos especificando como endereço para os arquivos a expressão “\*.data”
- Revise o Spark UI, repare novamente no “Input Size/Records”

## Lab 3.2: RDD na prática

**Tempo aproximado: 30 – 40 minutos**

- Nosso cenário de teste vamos assumir que temos usuários que participam de vários meetups. Começaremos com 2 usuários:
  - User1 atende aos meetups: m1, m2 e m3
  - User2 atende aos meetups: m2, m3, m4 e m5



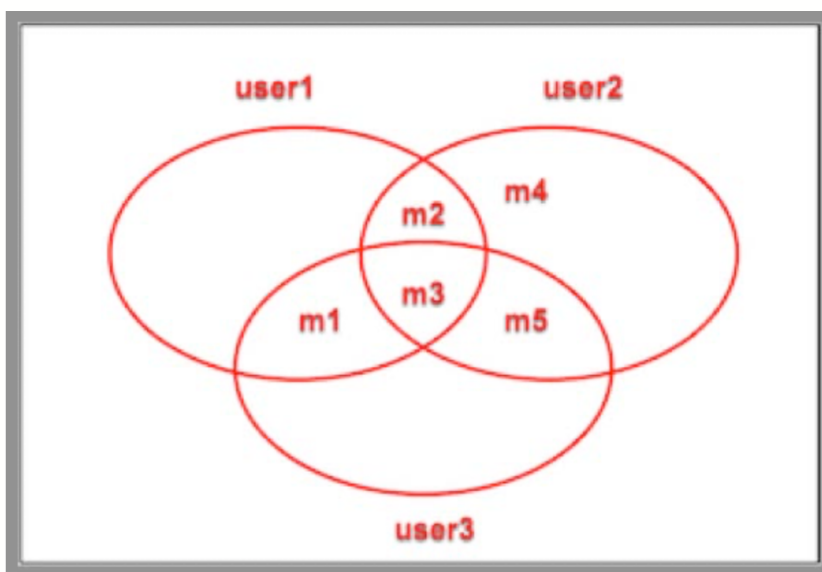
- Cada meetup dos usuários estarão separados por dois RDD. Devemos analisar os dados dos usuários aplicando operações sobre os RDD. Utilizaremos operações como union, intersection, distinct and subtract
- Iniciamos criando os dois RDD que representam os meetups de cada usuário

```
val u1 = spark.sparkContext.parallelize(List("m1", "m2", "m3"))
```

```
val u2 = spark.sparkContext.parallelize(<agora eh com vcs>)
```

- O que desejamos:
  - Usando os dois RDD's, encontre meetups que são comuns a ambos os usuários
    - Qual operação vc usou para fazer isso?
    - Olhando os Jobs pelo Spark UI você vê algo? (Deveria)
    - Vá nos detalhes do job (clcando na coluna description) e veja a DAG.
    - Clicando nas caixas azuis do diagrama você terá mais detalhes .
  - Encontre meetups que são do usuário User1 e User2
    - Como você fez isso? E se tiver resultados duplicados? Como você remove eles?
    - Olhe a DAG depois de ter feito essas transformações
  - Encontre os meetups atendidos apenas pelo u1.
    - Ou seja, aqueles que o u1 atende mas o u2 não
    - Olhe a DAG

- Cria dois datasets de recomendação de meetups, onde cada usuário recomenda ao outro aquele que outro ainda não atende
  - u1 recomendará para o u2 : m1
  - u2 recomendará para o u1 : m4 e m5
  - Como vc pode fazer isso? Que operações são necessárias?
  - Olhe a DAG
- Recomendações com 3 Usuários
  - Vamos contar agora com mais um usuário o u3:
    - u3 atende aos meetups: m1, m3 e m5



- Considere fazer recomendações considerando os seguintes requerimentos:
  - O usuário não deve estar atendendo aquele meetup para pode ser recomendado a ele.
  - O meetup deve ser atendido aos outros dois usuários para que possa ser recomendado a ele
  - A resposta esperada é:
    - U1: m5
    - U2: m1
    - U3: m2
  - Olhe a DAG depois de feito.

## Lab 4.1: Formatos de dados

**Tempo: 15 minutos**

- Vamos carregar dados textos e Json nesse momento, nos labs a seguir vamos ver outros tipos como Parquet
- Lembre-se que os caminhos dos arquivos são sempre “/home/ec2-user/lab”
- Carregue utilizando o SparkSession e o DataFrameReader o arquivo people.json e wiki-pageviews.txt cada um em uma variável diferente

```
>val folksDF = //Carregue o json
>val viewsDF = //Carregue o txt
```
- Se vocês olharem o SparkUI irão reparar que o carregamento do JSON foi imediato enquanto o txt não. Por quê?
- Chame o count para a variável viewDF
- Veja uma parte dos dados

```
>folksDF.limit(5).show()
>viewsDF.limit(5).show()
```
- Carregue o dado github.json e armazene na variável githubDF
- Olhe um parte dos dados usando o limite (Não faça um show sem o limit)
- Escreva o dado em formato parquet no diretório “/home/ec2-user/lab/output/folks.parquet”
- Entre na máquina e veja o “arquivo” que você criou

## Lab 4.2: Schemas

**Tempo: 15 minutos**

- Tendo já carregado o folksDF, viewsDF vamos agora carregar o arquivo people.json e colocar na variável peopleDF
- Vamos ver o schema de cada arquivo

```
>folksDF.printSchema()
```
- Declarando um schema explicitamente

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

```
val mySchema = (new StructType).add("name",  
StringType).add("gender", StringType).add("age",  
IntegerType)
```

- Carregue novamente o people.json porém fornecendo o schema acima
  - Repare que como foi informado o schema ele não faz um carregamento imediato
- Carregue o arquivo people-with-address.json na variável folksAddressDF
- Printe o schema
  - Veja que existe uma estrutura aninhada para o endereço
- Crie você o schema desse dataset, pois esse que você está vendo veio do inferência
  - Crie primeiro o schema para o endereço separadamente
  - Crie agora o schema para o dataset, atribuindo o schema acima ao campo endereço.  
>val addressSchema = ...  
>val schemaWithAddress = (new  
StructType).add("address", addressSchema)...

### Lab 4.3: Dataframe transformações

Tempo: 30 - 40 minutos

- Carregue o arquivo people.json
- Apresente o dado na tela
- Conte quantos são “F” e quantos são “M”
- Encontre a pessoa mais velha do genero “F”
  - Utilize spark.sql(“..”) para isso
  - Não esqueça de registrar uma tabela temporária
  - Use uma subquery para encontrar a idade maxima
- Carregue o arquivo github.json
- Olhe o schema e 5 amostra de dados
  - É difícil entender essa visão pois há muitos registros numa única linha
  - Selecione apenas uma coluna e coloque o resultado numa variável separada  
`>val actorDF = ...`
  - Veja o schema desse dataframe e mostre 5 amostras
  - Quando estiver usando o comando show use assim  
`>actorDF.show(false)`
- Selecione a coluna login e mostre ele
  - Use “actor.login” Para acesso a elementos aninhados
  - Veja algumas linhas
  - Encontre quantos logins únicos existe no dataset
- Verifique nesse dataset quantos registros onde a coluna Type possua o valor CreateEvent

### Lab 4.4: A API Tipada do Dataset

Tempo: 30 - 40 minutos

- Iremos trabalhar sobre o dataset de itens musicais, sendo eles:
  - Title: O titulo do item
  - Artist: O artista que lançou o titulo
  - Price: O preço do item
  - Category: A categoria musical
- Crie um dataframe do arquivo “data/music.json”  
`>val musicDF = ...`
  - Visualize o schema do DataFrame
  - Apresente os dados do DataFrame com “show”

- Defina um case class de nome MusicItem para o arquivo music.json
- Crie um DataSet usando o case class que você criou para o DataFrame musicDF
  - >case class MusicItem...
  - >val musicDS = ...
    - Qual é o tipo do musicDS? Apresente os dados do dataset
- Comparando o Dataframe vs DataSet
  - Filtro
    - Apresente todos os itens da Categoria Pop usando o musicDF e musicDS
  - Pegue o item de menor preço por categoria
    - Você precisará agrupar e depois agregar
    - Pegue o item de menor preço dentro da categoria para o musicDF e musicaDS
  - Transforme o dado para que o preço tenha uma redução de 10% (Isso pode ser feito multiplicando por 0.9)
    - Para o musicDF você pode usar o select com literal para criação do novo valor
    - Para o musicDS você pode usar map() e transformar o valor para Double para fazer a conta
  - Cometa erros!
    - Usando o musicDF e musicDS tente multiplicar o preço pela string "R". O que aconteceu?

## Lab 4.5: Divisão de dados

**Tempo: 30 - 40 minutos**

- Nesse vamos trabalhar com um dataset de dados textos, mesmo o dataset sendo regularmente estruturado o Spark não consegue deduzir sua estrutura interna por si só. Usaremos algumas funções do DataFrame para criar um DataFrame com schema para que fique fácil fazer buscas sobre os dados
- O dataset do arquivo "data/wiki-pageviews.txt" contém um dump de dados de pageview de alguns projetos do wikimedia. Os dados contêm quatro campos:
  - Dono do Projeto
  - Nome da Pagina
  - Número de Pageviews
  - Tempo de resposta em bytes



- Por ser um dataset em formato text o mesmo será carregado em uma única coluna, nossa atividade será aplicar um schema melhor.
- Crie um DataFrame do arquivo “data/wiki-pageviews.txt”
- Visualize algumas linhas para se familiarizar com o formato do dados
 

```
>val viewsDF = spark.read.text("/home/ec2-user/lab/data/wiki-pageviews.txt")
```
- Crie um DataFrame a partir do split de cada linha
  - Use split() para dividir os dados por espaço em branco (padrão “\\s+”)
  - Chame a coluna resultante de splitLineDF renomeando a coluna por meio do “as(newName)”
- Visualize o schema desse DataFrame (veja poucas pois há muitas), utilize o comando show(false) para que o dado não fique truncado
- Vamos melhorar o schema
  - Crie um schema com a seguinte estrutura
    - domain:String
    - pagename:String
    - viewCount:String
    - size:Long
  - Use o select para criar um dataframe com o schema que definimos
    - Você pode selecionar o nth elemento de uma linha do tipo array usando a função splitLine(n)
    - Você pode converter os valores de cada elemento usando a função cast(dataType)
    - Você pode renomear o nome da coluna usando “as(newName)”
  - Feito isso
    - Visualize o schema
    - Visualize algumas linhas
- Tente algumas queries
  - Busque linhas que possua mais de 500 visualizações (viewCount > 500) e apresente 5 delas
  - Faça a mesma busca, porém filtre também pelo domain igual “en”