**##jupyter\_install**

## Fazer o download do .zip na sua máquina

<https://drive.google.com/file/d/1UAhXeRMENJTRszRD0atpac91U3QVEL3u/view?usp=sharing>

## Em um terminal, instale todas as dependências:

**sudo yum install libXcomposite libXcursor libXi libXtst libXrandr alsa-lib**

**mesa-libEGL libXdamage mesa-libGL libXScrnSaver -y**

## Com as dependências instaladas, faça o download do Anaconda:

**wget https://repo.anaconda.com/archive/Anaconda2-2019.07-Linux-x86\_64.sh**

## Com o arquivo já salvo na VM, executa o arquivo:

**bash Anaconda2-2019.07-Linux-x86\_64.sh**

## Após a criação do arquivo, altere a permissão do arquivo:

**sudo chmod 777 jupyter\_notebook.sh**

**sh jupyter\_notebook.sh**

**##pyspark**

## Executar no terminal comando "pyspark" para abrir interface de execução do spark python

## Operações básicas, ações e transformações.

# criamos um rdd (Conjuntos de dados distribuídos resilientes)

**numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])**

# mostra primeiro elemento

**numeros.first()**

# mostra os 5 maiores

**numeros.top(5)**

#mostra todos elementos

**numeros.collect()**

# contar os elementos

**numeros.count()**

# media dos numeros

**numeros.mean()**

# somar os elementos

**numeros.sum()**

# mostra maior elemento

**numeros.max()**

# mostra menor elemento

**numeros.min()**

# calcula desvio padrão

**numeros.stdev()**

## Após execução de todos os comando o spark não persiste o resultado, aí vem método abaixo para fazer isso acontecer

## com método persist, assim fazendo ele armazenar em memória no exemplo abaixo.

**numeros.persist(StorageLevel.MEMORY\_ONLY\_SER)**

# vamos aplicar um filtro para pegar os elementos maiores que 2

**filtro = numeros.filter(lambda filtro: filtro > 2)**

# mostra resultado

**filtro.collect()**

# esse vamos pegar uma amostra de dados, o primeiro parâmetro do sample fala se vai ou não ter reposição, o segundo é probabilidade, e ultimo se vamos repetir os números gerados.

**amostra = numeros.sample(True,0.5,2)**

**amostra.collect()**

# vamos multiplicar cada valor do meu rdd por 2

**mapa = numeros.map(lambda mapa: mapa \* 2)**

**mapa.collect()**

# Criamos novo Rdd

**numeros2 = sc.parallelize([6,7,8,9,10])**

# vamos fazer uma intersecção entre o rdd números e numeros2

**interseccao = numeros.intersection(numeros2)**

**interseccao.collect()**

**subtrai = numeros.subtract(numeros2)**

**subtrai.collect()**

# vamos criar produto cartesiano

**cartesiano = numeros.cartesian(numeros2)**

**cartesiano.collect()**

# fazer contagem dos valores

**cartesiano.countByValue()**

**##Operações chave-valor**

**compras = sc.parallelize([(1,200), (2,300), (3,120), (4,250), (5,78)])**

# separação das chaves

**chaves = compras.keys()**

**chaves.collect()**

# separação dos valores

**valores = compras.values()**

**valores.collect()**

**compras.countByValue()**

**soma = compras.mapValues(lambda soma: soma + 1)**

**soma.collect()**

**agrupa = compras.groupByKey().mapValues(list)**

**agrupa.collect()**

**debitos = sc.parallelize([(1,20), (2,300)])**

**resultado = compras.join(debitos)**

**resultado.collect()**

**semdebito = compras.subtractByKey(debitos)**

**semdebito.collect()**

**##WordCount**

## Usando mesmo arquivo do caso Hadoop - Link para Downloads: <https://drive.google.com/file/d/1ow4LJKTs87XPGRu9QV1baFXXhUJxyteM/view?usp=sharing>

**pesquisa = sc.textFile("file:///home/cloudera/Downloads/pesquisa.txt")**

**pesquisa.collect()**

**pesquisa.take(5)**

## Agora vamos fazer Word Count no Spark usando RDD

# vamos dividir as palavras usando split

**contagem = pesquisa.flatMap(lambda palavra: palavra.split(" "))**

# vamos agora usa função map para mapear as palavras como chave valor

**contagem = contagem.map(lambda pal: (pal,1))**

# agora vamos executar reduce para agrupar as palavras assim contando elas

**contagem = contagem.reduceByKey(lambda a, b: a + b)**

**contagem.take(5)**

# vamos armazenar resultado no hdfs

**contagem.saveAsTextFile("conta")**

**hdfs dfs -ls /user/cloudera/conta/**

**hdfs dfs -cat /user/cloudera/conta/part-00000**

#excluir diretório

**sudo hdfs dfs -rm -r /user/cloudera/conta**

**##Analisar dados do Hive (DW)**

## Copiar arquivos de configuração:

**ls /usr/lib/hive/conf/hive-site.xml**

**cat /usr/lib/hive/conf/hive-site.xml**

**sudo cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/**

## Criar contexto HiveContext

#no shell do spark

**from pyspark.sql import HiveContext**

#Cria Contexto do Hive

**contexto = HiveContext(sc)**

#conectar no banco de dados na tabela

**banco = contexto.table("aula.categories")**

**banco.show()**

#Vamos registra a tabela no spark para ficar disponível para execução de querys

**banco.registerTempTable("categories")**

**contexto.sql("select \* from categories").show()**

**contexto.sql("select sum(product\_price) from categories").show()**

## Vamos criar Data Frame

#conectar no banco de dados na tabela

**banco = contexto.table("aula.categories")**

**banco.registerTempTable("categories")**

# a variável categoria é nosso dataframe

**categorias = contexto.sql("select \* from categories")**

**categorias.show()**

**categorias.show(100)**

**categorias.printSchema()**

# seria a mesma coisa que executar um Select estado, status From des\_produtos

**categorias.select('product\_name', 'product\_price').show()**

**categorias.select('product\_name', 'product\_price').distinct().show(30)**

**categorias.select('product\_name', 'product\_price').show()**

**categorias.filter(categorias.product\_name=='ENO Atlas Hammock Straps').show()**

**categorias.filter(categorias.product\_name=='ENO Atlas Hammock Straps').count()**