1. Você está configurando um ambiente de Big Data para uma empresa e precisa demonstrar o uso básico de várias ferramentas. Para cada uma das seguintes tecnologias, escreva o comando ou script necessário para realizar a tarefa especificada:

   a. Apache Hive, crie uma tabela chamada "produtos" com as colunas id (INT), nome (STRING) e preco (FLOAT);

   b. Hadoop HDFS, copie um arquivo local chamado "dados.csv" para o diretório "/input" no HDFS;

   c. Apache Pig: carregue dados de um arquivo CSV chamado "vendas.csv" com três colunas: id, produto e valor.

Criando a tabela ‘produto’ com os campos: “id”, “nome” e “preco”

```
CREATE TABLE produtos (
    id INT,
    nome STRING,
    preco FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Copiando o arquivo para o diretório “/input”

```
hdfs dfs -put dados.csv /input/
```

Carregando dados armazenados com o apache pig:

```
vendas = LOAD 'vendas.csv' USING PigStorage(',') AS (id:int, produto:chararray, valor:float);
```

2. Em um projeto de implementação de Big Data, você precisa executar consultas básicas em diferentes ferramentas. Para cada tecnologia, escreva uma consulta ou comando para realizar a tarefa especificada:

   a. Apache Hive, selecione todos os produtos da tabela "produtos" onde o preço é maior que 100;

   b. Hadoop HDFS, liste todos os arquivos no diretório "/input" do HDFS;

   c. Apache Pig, filtre as vendas do arquivo "vendas.csv" para mostrar apenas aquelas com valor acima de 500.

Selecionando produtos com o preço maior que 100:

```
SELECT * FROM produtos WHERE price > 100;
```

Listando arquivos no diretório “/input” no HDFS

```
hdfs dfs -ls /input
```

Filtrando vendas acima de 500 com apache pig

```
vendas = LOAD 'vendas.csv' USING PigStorage(',')
         AS (id:chararray, produto:chararray, valor:double);

vendas_filtradas = FILTER vendas BY valor > 500;
```

3. Em um projeto de Big Data, você precisa carregar dados em diferentes ferramentas. Para cada tecnologia abaixo, escreva um comando ou script para realizar a tarefa de carregamento de dados especificada:

   a. Apache Hive, carregue dados de um arquivo CSV chamado "vendas.csv" para uma tabela Hive chamada "vendas" com as colunas: id (INT), produto (STRING), valor (FLOAT);

   b. Hadoop HDFS, carregue um arquivo local "clientes.txt" para o diretório "/data" no HDFS;

   c. Apache Spark, leia um arquivo JSON chamado "config.json" e crie um DataFrame.

Assumindo que temos a seguinte tabela.

```
CREATE TABLE IF NOT EXISTS vendas (
  id INT,
  produto STRING,
  valor FLOAT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Podemos executar o seguinte comando com um CSV válido:

```
LOAD DATA LOCAL INPATH 'vendas.csv'
OVERWRITE INTO TABLE vendas;
```

Carregando um arquivo clientes.txt para o diretório /data:

```
hdfs dfs -put clientes.txt /data
```

Criando uma sessão spark para ler um arquivo json:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("LoadConfig") \
    .getOrCreate()

config = spark.read.json("config.json")
config.show()

spark.stop()
```

4. Você precisa realizar operações de transformação de dados em diferentes ferramentas. Para cada tecnologia, escreva um comando ou script para realizar a tarefa de transformação especificada:

   a. Apache Hive, crie uma nova tabela "vendas_resumo" que contenha o total de vendas por produto a partir da tabela "vendas" (id INT, produto STRING, valor FLOAT);

   b. Apache Pig, agrupe os dados do arquivo "logs.csv" (colunas: data, ip, url) por URL e conte o número de acessos para cada URL;

   c. Apache Spark, usando um DataFrame "df" criado a partir de "vendas.csv" (colunas: id, produto, valor), calcule o valor total de vendas por produto.

Criando a tabela “vendas_resumo”

```sql
CREATE TABLE IF NOT EXISTS vendas_resumo AS
SELECT
  produto,
  SUM(valor) AS total_vendas
FROM vendas
GROUP BY produto;
```

Agrupando dados do arquivo “logs.csv” por url e somando os acessos:

```
logs = LOAD 'logs.csv' USING PigStorage(',')
       AS (data:chararray, ip:chararray, url:chararray);

logs_grouped = GROUP logs BY url;

acessos_por_url = FOREACH logs_grouped
                  GENERATE
                    group AS url,
                    COUNT(logs) AS total_acessos;

DUMP acessos_por_url;
```

Calculando o total de vendas por produto com spark:

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum

spark = SparkSession.builder \
    .appName("ResumoVendas") \
    .getOrCreate()

df = spark.read.csv("vendas.csv", header=True, inferSchema=True)

resumo_df = df.groupBy("produto") \
              .agg(sum("valor").alias("total_vendas"))

resumo_df.show()
```

5. Em um projeto de implementação de Big Data utilizando o ecossistema Hadoop, você precisa demonstrar o entendimento do conceito de ETL/ELT. Para cada tecnologia, descreva uma tarefa simples de ETL ou ELT:
   a. Apache Hive, descreva como você usaria o Hive para extrair dados de uma tabela "vendas_brutas", transformar calculando o total de vendas por produto, e carregar em uma nova tabela "vendas_resumo";
   b. Hadoop HDFS e MapReduce, explique como você implementaria um processo ETL para contar palavras em arquivos de texto armazenados no HDFS;
   c. Apache Pig, explique como você usaria o Pig para realizar um processo ETL que filtra registros de log e calcula estatísticas básicas.

Extraindo dados da tabela “vendas_brutas”.

```
SELECT
  produto,
  SUM(valor)      AS total_vendas
FROM vendas_brutas
GROUP BY produto;
```

Como implementar um ETL para contar palavras.

1. **Extração (E - Extract):**

Os arquivos de texto são armazenados no HDFS. O job MapReduce lê esses arquivos como entrada.

2. **Transformação (T - Transform):**

O **Mapper** lê cada linha, divide em palavras e emite tuplas (“1” : Beterraba)

O **Reducer** soma tuplas com palavras iguais

3. Carga (L - Load):

O resultado é gravado novamente no HDFS.

As etapas poderão ser iniciadas por um **cronjob**

Como usar ETL em um processamento com apache PIG:

Passos:

1. **Extração (E):** Carregar os arquivos de log.
2. **Transformação (T):** Filtrar linhas com erros e agrupar por tipo.
3. **Carga (L):** Salvar o resultado em um diretório no HDFS.

6. Para importar dados de um banco de dados SQL Server para o ecossistema Hadoop, utilizando Apache Spark, como você escreveria um código PySpark para ler os dados de uma tabela chamada "clientes" diretamente do SQL Server.

```python
from pyspark.sql import SparkSession

spark = (SparkSession.builder
    .appName("LeituraSQLServer")
    .config("spark.jars", "/caminho/para/mssql-jdbc-9.2.1.jre8.jar")
    .getOrCreate()
)

jdbc_url = (
    "jdbc:sqlserver://<HOST>:<PORT>;"
    "databaseName=<DATABASE>;"
    "encrypt=true;"
    "trustServerCertificate=true"
)

connection_properties = {
    "user": "<USERNAME>",
    "password": "<PASSWORD>",
    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"
}

df_clientes = (spark.read
    .format("jdbc")
    .option("url", jdbc_url)
    .option("dbtable", "clientes")
    .options(**connection_properties)
    .load()
)

df_clientes.show(10)

spark.stop()
```

7. Em um projeto de Big Data utilizando HBase, você precisa realizar operações básicas e integrar o HBase com outras ferramentas do ecossistema Hadoop. Para cada item, escreva um comando ou explique brevemente como realizaria a tarefa:
   a. HBase, crie uma tabela chamada "clientes" com uma família de colunas chamada "info";
   b. HBase, insira um novo registro na tabela "clientes" com a chave de linha "1001", nome "Alice" e email "alice@email.com";
   c. Apache Pig, escreva um script Pig para ler dados da tabela "clientes" do HBase.

Criando e inserindo clientes no HBase

```
create 'clientes', 'info'

--inserting clients

put 'clientes', '1001', 'info:nome', 'Mateus'
put 'clientes', '1001', 'info:email', 'Mateus@email.com'
```

Lendo a tabela de clientes com PIG

```
clientes = LOAD 'hbase://clientes'
    USING org.apache.pig.backend.hadoop.hbase.HBaseStorage(
      'info:nome info:email', '-loadKey true')
    AS (rowkey:bytearray, nome:chararray, email:chararray);
```

8. Utilizando o HBase, você precisa configurar e gerenciar tabelas. Para cada item, escreva um comando ou explique brevemente como realizaria a tarefa: crie uma tabela chamada "vendas" com duas famílias de colunas: "info" e "detalhes", altere a tabela "vendas" para adicionar uma nova família de colunas chamada "estatísticas", liste todas as tabelas existentes no HBase e mostre a estrutura detalhada da tabela "vendas".

```
-- Criando a tabela com os campos: "info" e "detalhes"
create 'vendas', 'info', 'detalhes'

-- Adicionando o campo estatísticas
alter 'vendas', 'estatísticas'

-- Listando as tabelas disponíveis
list

-- Exibindo detalhes da tabela de vendas
describe 'vendas'
```