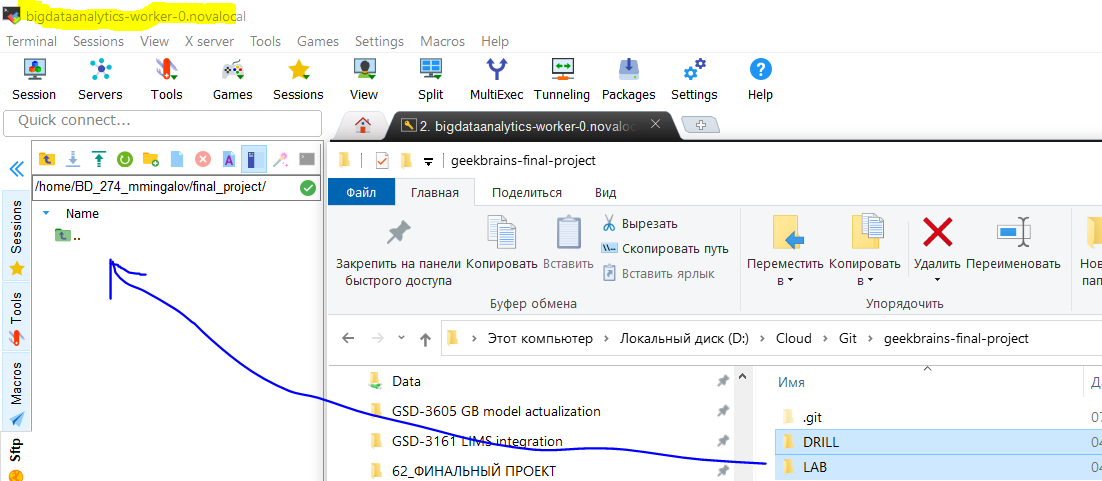
**<https://github.com/mmingalov/geekbrains-final-project/blob/master/temp.py>**

# **ПОДГОТОВКА ДАННЫХ**

**Открываем сессию mobaXterm**

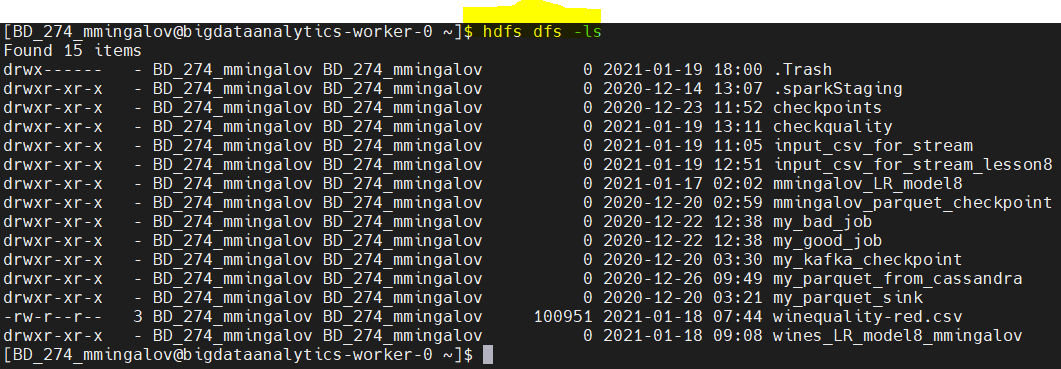
**Выполняем drag&drop копирование файлов в директорию проекта:**



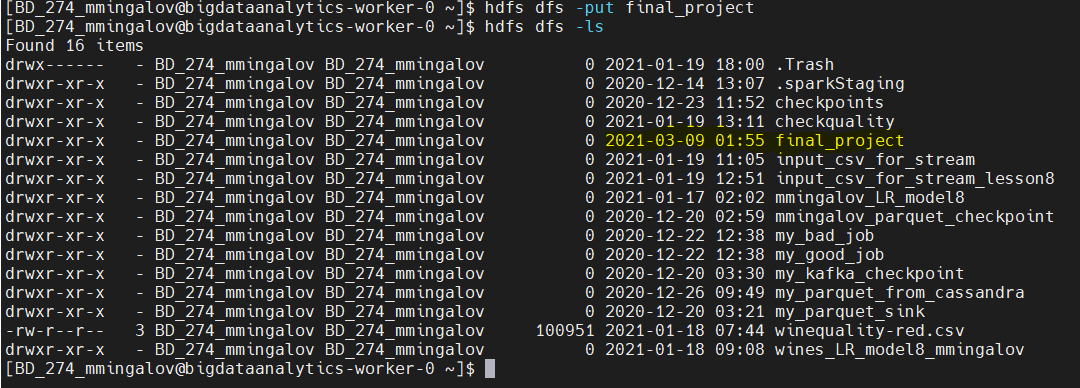
**Файлы скопированы в локальный каталог worker-0**

**Нам необходимо перенести их в HDFS**

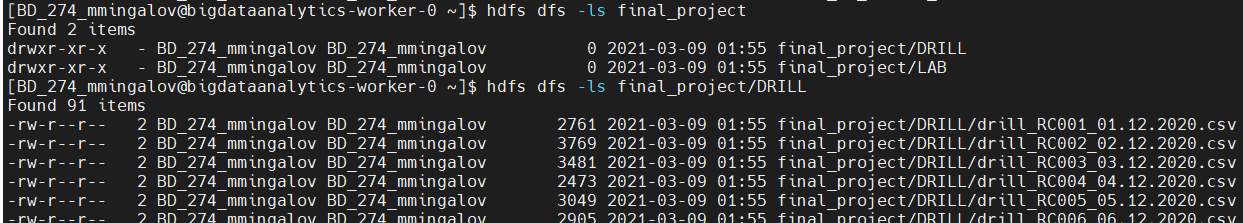
**Посмотрим, что за папки на данный момент есть в HDFS:**



**Создадим директорию под финальный проект: просто перенесем final\_project в HDFS командой** **put**



**Посмотрим внутрь..**



**Видим, что данные на месте и готовы к дальнейшей работе.**

# **ЧТЕНИЕ ДАННЫХ**

**Примечание: нижеприведенный код есть в проекте в виде .py файла**

[**https://github.com/mmingalov/geekbrains-final-project/blob/master/temp.py**](https://github.com/mmingalov/geekbrains-final-project/blob/master/temp.py)

**Запускаем SPARK командой:**

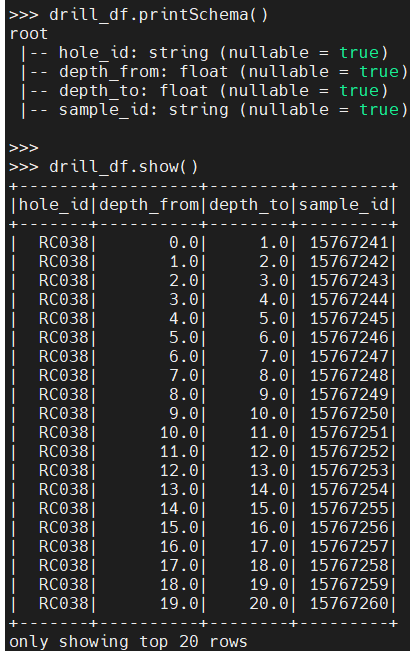
**export SPARK\_KAFKA\_VERSION=0.10**

**/spark2.4/bin/pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.5 --driver-memory 512m --num-executors 1 --executor-memory 512m --master local[1]**

drill\_df = spark.read.format("csv")\  
 .option("header", "false")\  
 .schema("hole\_id STRING, depth\_from float, depth\_to float, sample\_id STRING")\  
 .load("final\_project/DRILL/\*.csv")  
lab\_df = spark.read.format("csv")\  
 .option("header", "false")\  
 .schema("sample\_id STRING, result FLOAT")\  
 .load("final\_project/LAB/\*.csv")

drill\_df.printSchema()

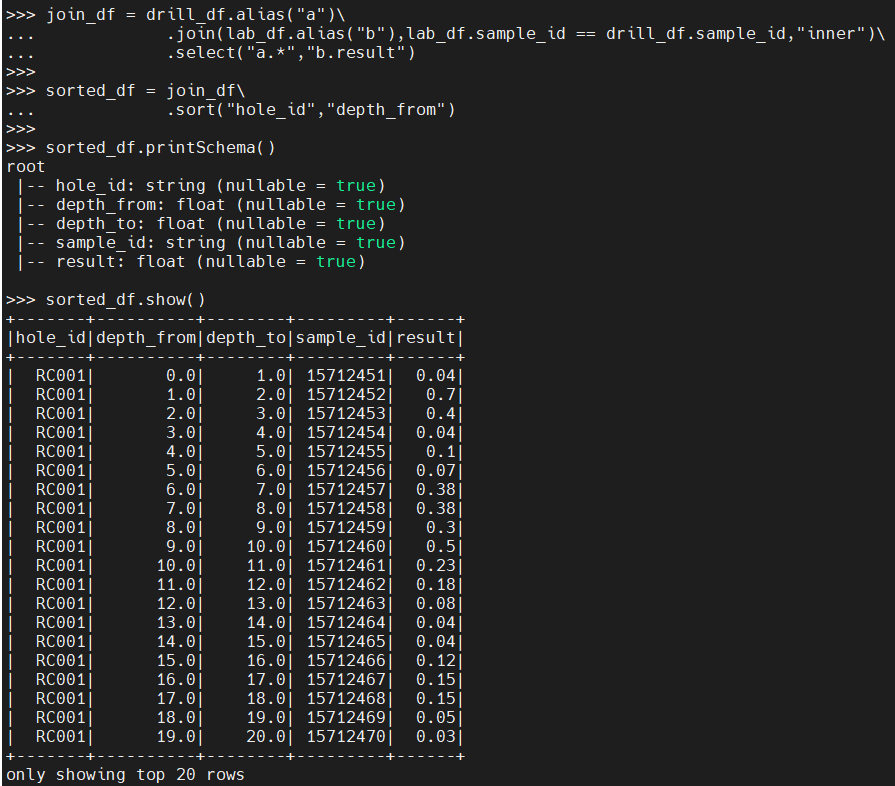
drill\_df.show()



# **СЛИЯНИЕ ДАННЫХ**

**Выполняем слияние двух spark\_df в общий датасет, сортируя по «номеру скважины» и глубине «ОТ»:**

#JOIN AND SORTING  
join\_df = drill\_df.alias("a")\  
 .join(lab\_df.alias("b"),lab\_df.sample\_id == drill\_df.sample\_id,"inner")\  
 .select("a.\*","b.result")  
  
sorted\_df = join\_df\  
 .sort("hole\_id","depth\_from")  
  
sorted\_df.printSchema()  
sorted\_df.show()



# **РАСЧЕТ КОМПОЗИТОВ (UDF)**

**Для расчета композитов нам понадобятся вспомогательные столбцы.**

**Будем рассчитывать их при помощи пользовательских функций**

**UDF**

from pyspark.sql.functions import udf

**при расчете композитов у нас есть глобальные параметры: содержание по борту и мощность рудного интервала**

Cb = 0.37  
Mr = 2

**Подготовим функции под шаги расчета композитов**

**Сперва вспомогательные:**

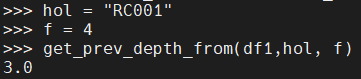
def get\_prev\_depth\_from(df,hol, f):  
 try:  
 result = df.filter((df["depth\_to"] == f) & (df["hole\_id"] == hol)).select(["depth\_from"]).collect()[0].\_\_getitem\_\_("depth\_from")  
 except:  
 result = -1  
 return result  
udf\_gpdf = udf(get\_prev\_depth\_from)  
  
def get\_next\_depth\_to(df, hol,t):  
 try:  
 result = df.filter((df["depth\_from"] == t) & (df["hole\_id"] == hol)).select(["depth\_to"]).collect()[0].\_\_getitem\_\_("depth\_to")  
 except:  
 result = -1  
 return result  
  
udf\_gndt = udf(get\_next\_depth\_to)

**потом основная логика:**

def calc\_step1(res):  
 if res < Cb:  
 result = 0 #'П'  
 else:  
 result = 1 #'Р?'  
 return result  
udf\_step1 = udf(calc\_step1)

#STEPS EXECUTING  
df1 = sorted\_df \  
 .withColumn("step1", udf\_step1(join\_df["result"]))

**сейчас уткнулся в проблему, что исходная функция срабатывает**



**А зарегистрированная по ней UDF функция выдает ошибку:**

