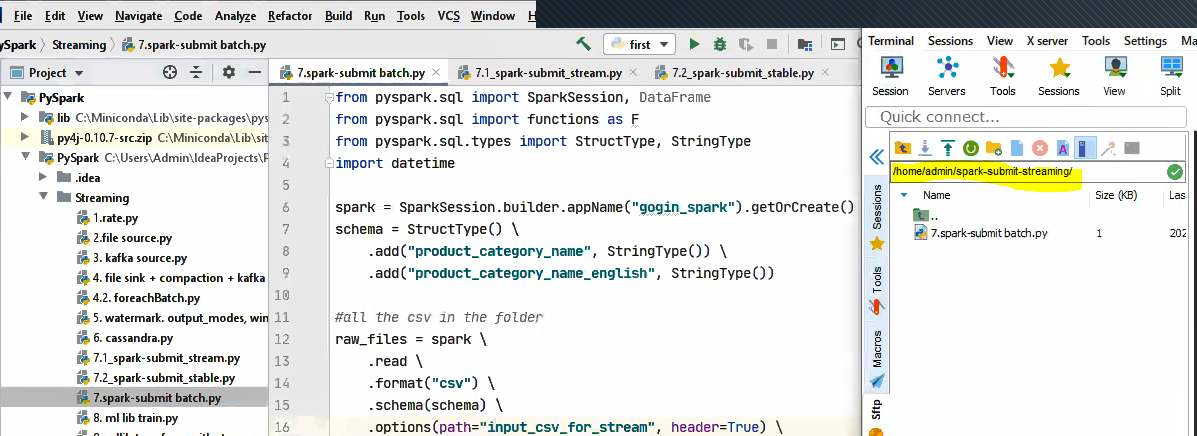
# **ЗАДАНИЕ 07 (Аналитика признаков в пакетном режиме):**

Повторить запуск Spark приложений с такими параметрами (можно еще добавлять свои) /spark2.4/bin/spark-submit --driver-memory 512m --driver-cores 1 --master local[1] my\_script.py

## **SPARK SUBMIT BATCH**

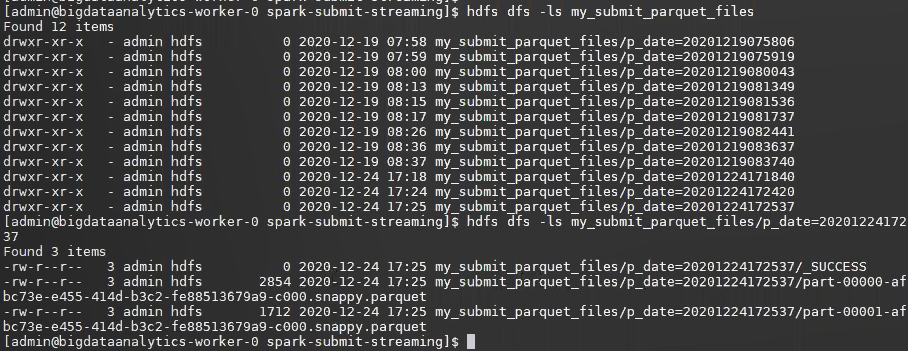
from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql import functions as F  
from pyspark.sql.types import StructType, StringType  
import datetime  
  
spark = SparkSession.builder.appName(**"mmingalov\_spark"**).getOrCreate()  
schema = StructType() \  
 .add(**"product\_category\_name"**, StringType()) \  
 .add(**"product\_category\_name\_english"**, StringType())  
  
#читаем все csv в батче  
raw\_files = spark \  
 .read \  
 .format(**"csv"**) \  
 .schema(schema) \  
 .options(path=**"input\_csv\_for\_stream"**, header=True) \  
 .load()  
  
#fix timestamp  
load\_time = datetime.datetime.now().strftime(**"%Y%m%d%H%M%S"**)  
print(**"START BATCH LOADING. TIME = "** + load\_time)  
  
#пишем паркеты в партиции  
raw\_files.withColumn(**"p\_date"**, F.lit(**"load\_time"**)) \  
 .write \  
 .mode(**"append"**) \  
 .parquet(**"my\_submit\_parquet\_files/p\_date="** + str(load\_time))  
  
print(**"FINISHED BATCH LOADING. TIME = "** + load\_time)  
  
spark.stop() #в конце stop не обязательно. SUBMIT сам вырубает spark в конце

Файл можно перетащить на сервер (mobaXterm + drag&drop):



Запускаем из консоли вот так:

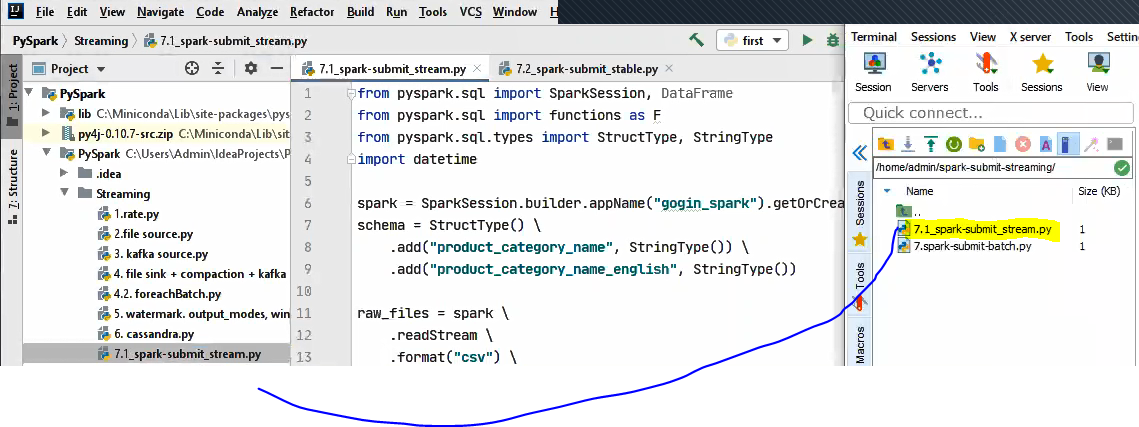




## **SPARK SUBMIT STREAM**

from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql import functions as F  
from pyspark.sql.types import StructType, StringType  
import datetime  
  
spark = SparkSession.builder.appName(**"gogin\_spark"**).getOrCreate()  
schema = StructType() \  
 .add(**"product\_category\_name"**, StringType()) \  
 .add(**"product\_category\_name\_english"**, StringType())  
  
#читаем csv файлы в стриме  
raw\_files = spark \  
 .readStream \  
 .format(**"csv"**) \  
 .schema(schema) \  
 .options(path=**"input\_csv\_for\_stream"**, header=True) \  
 .load()  
  
# разово проставляем время загрузки  
load\_time = datetime.datetime.now().strftime(**"%Y%m%d%H%M%S"**)  
  
#ВСЕГДА ПИЩЕМ И ОДНУ ДИРЕКТОРИЮ  
def file\_sink(df, freq):  
 return df.writeStream.format(**"parquet"**) \  
 .trigger(processingTime=**'%s seconds'** % freq ) \  
 .option(**"path"**,**"my\_submit\_parquet\_files/p\_date="** + str(load\_time)) \  
 .option(**"checkpointLocation"**, **"checkpionts/my\_parquet\_checkpoint"**) \  
 .start()  
  
timed\_files = raw\_files.withColumn(**"p\_date"**, F.lit(**"load\_time"**))  
  
#запускаем стрим всегда в одну директорию  
stream = file\_sink(timed\_files,10)  
  
#will always spark.stop() at the end  
  
#СТРИМ ТУТ ЖЕ ЗАКОНЧИТСЯ ПОТОМУ ЧТО В КОНЦЕ SPARK.STOP()

Аналогично копируем на сервер:



Здесь имеем ситуацию, что приложение по коду доходит до конца и закрывается само. Бесконечно stream крутиться не будет

Поэтому попробуем модифицировать, привнеся ЦИКЛ

## **SPARK SUBMIT STABLE**

from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql import functions as F  
from pyspark.sql.types import StructType, StringType  
import datetime  
  
spark = SparkSession.builder.appName(**"gogin\_spark"**).getOrCreate()  
schema = StructType() \  
 .add(**"product\_category\_name"**, StringType()) \  
 .add(**"product\_category\_name\_english"**, StringType())  
  
raw\_files = spark \  
 .readStream \  
 .format(**"csv"**) \  
 .schema(schema) \  
 .options(path=**"input\_csv\_for\_stream"**, header=True) \  
 .load()  
  
#пишем стрим в foreachBatch, чтобы делать логику в зависимости от каждого микробатча  
def file\_sink(df, freq):  
 return df.writeStream.foreachBatch(foreach\_batch\_function) \  
 .trigger(processingTime=**'%s seconds'** % freq ) \  
 .option(**"checkpointLocation"**, **"checkpionts/my\_parquet\_checkpoint"**) \  
 .start()  
  
#в каждом микробатче фиксируем время, логируем на экран, пишем файлы в свою директорию  
def foreach\_batch\_function(df, epoch\_id):  
 load\_time = datetime.datetime.now().strftime(**"%Y%m%d%H%M%S"**)  
 print(**"START BATCH LOADING. TIME = "** + load\_time)  
 df.withColumn(**"p\_date"**, F.lit(**"load\_time"**)) \  
 .write \  
 .mode(**"append"**) \  
 .parquet(**"my\_submit\_parquet\_files/p\_date="** + str(load\_time))  
 print(**"FINISHED BATCH LOADING. TIME = "** + load\_time)  
  
stream = file\_sink(raw\_files,10)  
  
#запускаем бесконечный цикл  
while(True):  
 print(**"I'M STILL ALIVE"**)  
 stream.awaitTermination(9)  
  
#unreachable  
spark.stop()

# **ЗАДАНИЕ 08 (Streaming + Spark ML + Cassandra):**

В качестве итоговой работы необходимо написать свою ML модель. Обучить ее. Затем применять на стриме.  
Необходимо найти / сгенерировать собственные входные данные и объяснить, какую задачу решает ваша ML модель и почему именно так.

## **ПОВТОР ПРАКТИКИ УРОКА**

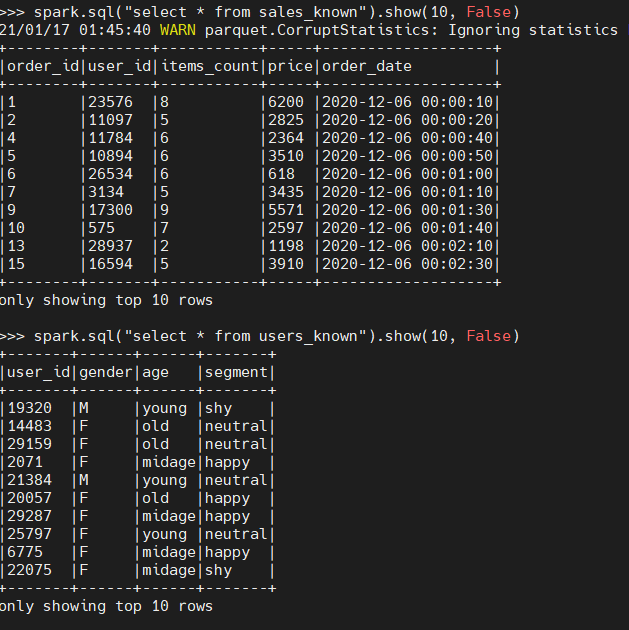
**Открываем новую сессию в mobaXterm, запускаем команды:**

export SPARK\_KAFKA\_VERSION=0.10

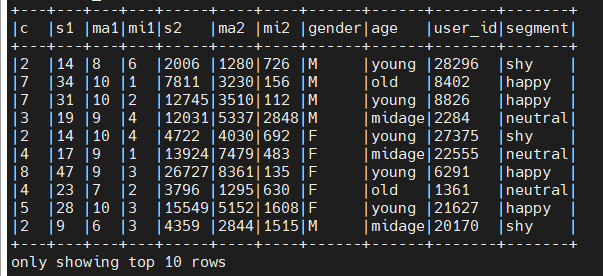
/spark2.4/bin/pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.5,com.datastax.spark:spark-cassandra-connector\_2.11:2.4.2 --driver-memory 512m --driver-cores 1 --master local[1]

from pyspark.ml import Pipeline, PipelineModel  
from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql.types import StructType, StringType, IntegerType, TimestampType  
from pyspark.sql import functions as F  
from pyspark.ml.classification import LogisticRegression  
from pyspark.ml.feature import OneHotEncoderEstimator, VectorAssembler, CountVectorizer, StringIndexer, IndexToString

spark.read.parquet("/apps/spark/warehouse/sint\_sales.db/sales\_known").createOrReplaceTempView("sales\_known")  
spark.read.parquet("/apps/spark/warehouse/sint\_sales.db/users\_known").createOrReplaceTempView("users\_known")  
  
spark.sql("select \* from sales\_known").show(10, False)  
spark.sql("select \* from users\_known").show(10, False)



**#считаем сегмент зависимым от количества покупок клиента, суммы всех купленых товаров клиента, максимального числа купленных товаров клиента, минимального числа купленных товаров клиента, суммы потраченных рублей клиента, максимально потраченных рублей клиента, минимально потраченных рублей клиента**users\_known = spark.sql("""  
select count(\*) as c, sum(items\_count) as s1, max(items\_count) as ma1, min(items\_count) as mi1,  
sum(price) as s2, max(price) as ma2, min(price) as mi2 ,u.gender, u.age, u.user\_id, u.segment   
from sales\_known s join users\_known u   
where s.user\_id = u.user\_id   
group by u.user\_id, u.gender, u.age, u.segment""")



**Подготовка датасетов, тренировка модели:**

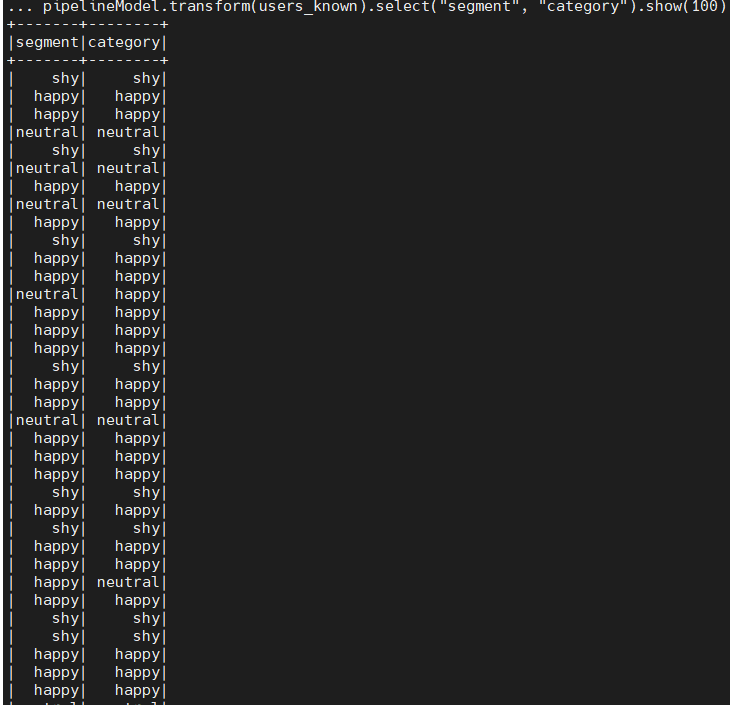
#подробное описание модели https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa  
#и https://spark.apache.org/docs/latest/ml-features.html  
#в общем - все анализируемые колонки заносим в колонку-вектор features  
categoricalColumns = ['gender', 'age']  
stages = []  
for categoricalCol in categoricalColumns:  
 stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index').setHandleInvalid("keep")  
 encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"]).setHandleInvalid("keep")  
 stages += [stringIndexer, encoder]  
  
label\_stringIdx = StringIndexer(inputCol = 'segment', outputCol = 'label').setHandleInvalid("keep")  
stages += [label\_stringIdx]

numericCols = ['c' ,'s1', 'ma1', 'mi1','s2', 'ma2', 'mi2']  
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols  
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features").setHandleInvalid("keep")  
stages += [assembler]

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)  
stages += [lr]  
  
label\_stringIdx\_fit = label\_stringIdx.fit(users\_known)  
indexToStringEstimator = IndexToString().setInputCol("prediction").setOutputCol("category").setLabels( label\_stringIdx\_fit.labels)  
  
stages +=[indexToStringEstimator]  
  
pipeline = Pipeline().setStages(stages)  
pipelineModel = pipeline.fit(users\_known)

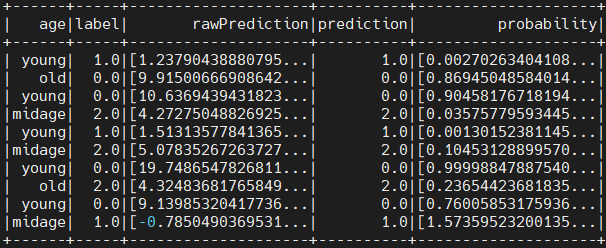
#сохраняем модель на HDFS  
pipelineModel.write().overwrite().save("mmingalov\_LR\_model8")

###для наглядности  
pipelineModel.transform(users\_known).select("segment", "category").show(100) #можно посчитать процент полной сходимости

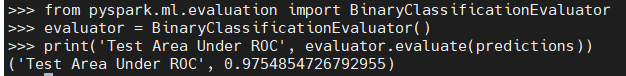


predictions = pipelineModel.transform(users\_known)

predictions.select('age', 'label', 'rawPrediction', 'prediction', 'probability').show(10)



**Evaluate our Logistic Regression model**.



## **САМОСТОЯТЕЛЬНАЯ РАБОТА**

**Задача: пройти все стадии процесса, показанного на уроке, с другими данными.**

**Заметки к решению:**

Для этого взят небольшой датасет по винам:

<https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009>

Обучать модель будем на этом исходном датасете по всему набору features.

В stage TRANSFORM сэмулируем поступление данных из разных источников:

- из KAFKA будет поступать поднабор wines\_acid\_info с значениями

'fixed acidity',

'volatile acidity',

'citric acid'

- из CASSANDRA будет поступать поднабор wines\_other\_info с значениями

'residual sugar',

'chlorides',

'free sulfur dioxide',

'total sulfur dioxide',

'density',

'pH',

'sulphates',

'alcohol'

- логику слияния двух поднаборов, агрегирование будем производить в forEachBatch

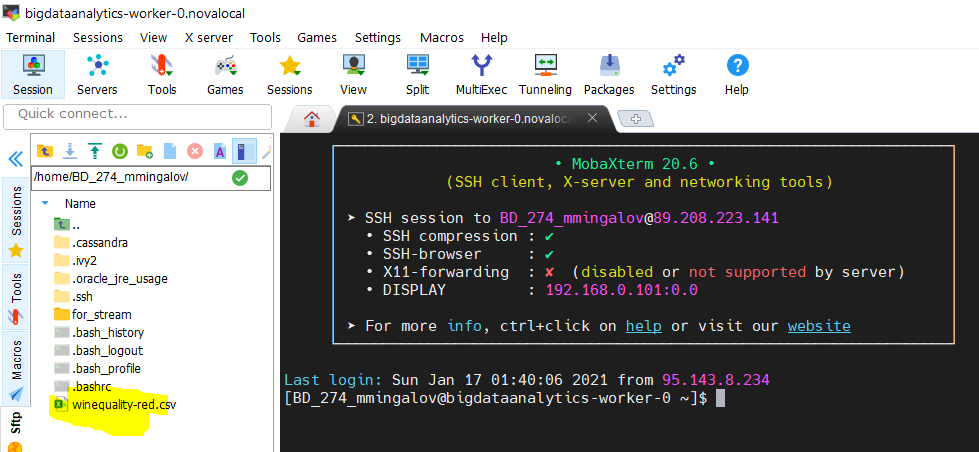
Поднаборы были получены из исходного набора

### **STAGE 1 – DATA AND ENVIRONMENT PREPARING**

**Открываем новую сессию в mobaXterm подключением к worker-0:**

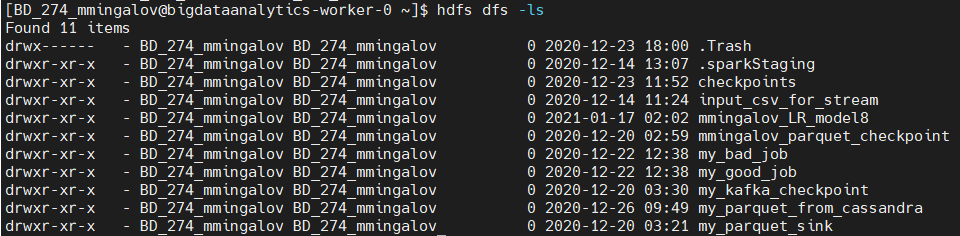
Нам потребуется обучать модель. У нас есть файл с данными для этого.

Методом drag&drop копируем в директорию терминала наш датасет:

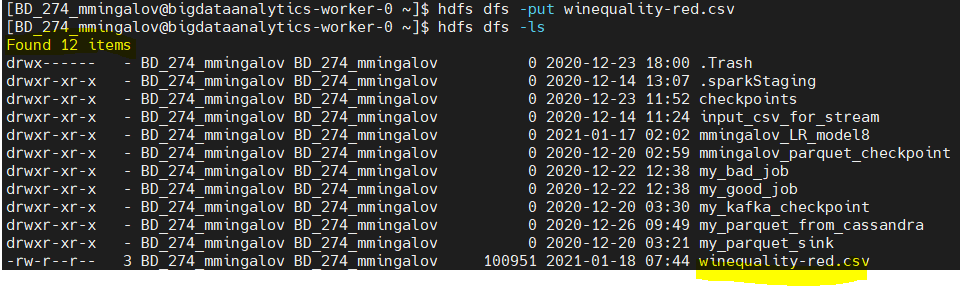


Теперь поместим его в HDFS.

Посмотрим, что у нас имеется в HDFS



Выполним копирование командой:



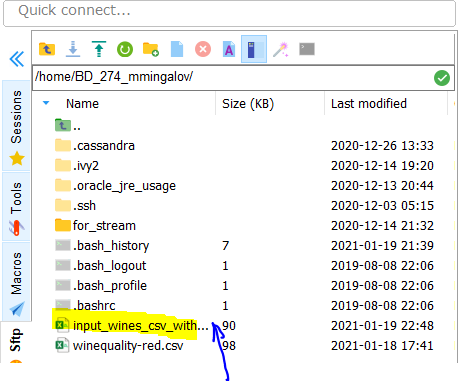
Теперь наш файл для обучения модели на HDFS

**ДАЛЕЕ**

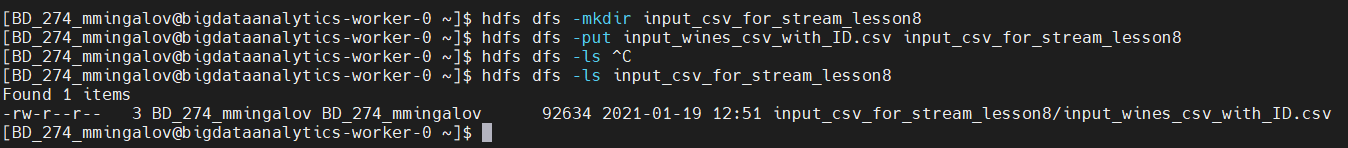
Для Kafka и Cassandra нам потребуется CSV-файл в HDFS, в котором есть все те же features +столбец ID. Я подготовил такой файл на основе исходного.

Копируем его (input\_wines\_csv\_with\_ID.csv) в директорию сервера

**bigdataanalytics-worker-0**

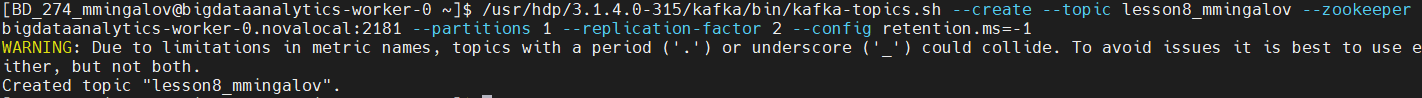


А теперь переносим его в HDFS в новую директорию input\_csv\_for\_stream\_lesson8:



Создаем topic в Kafka командой:

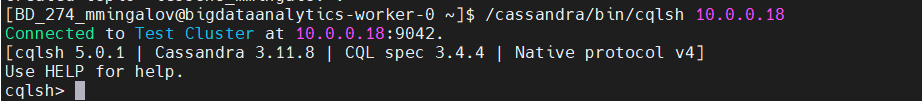
/usr/hdp/3.1.4.0-315/kafka/bin/kafka-topics.sh --create --topic lesson8\_mmingalov --zookeeper bigdataanalytics-worker-0.novalocal:2181 --partitions 1 --replication-factor 2 --config retention.ms=-1



Запускаем Cassandra и создаем схему и таблицы

Схема должна содержать все features исходного датасета.

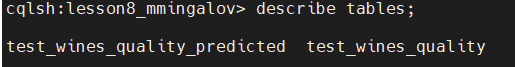
/cassandra/bin/cqlsh 10.0.0.18



создать схему  
CREATE KEYSPACE lesson8\_mmingalov  
 WITH REPLICATION = {  
 'class' : 'SimpleStrategy', 'replication\_factor' : 1 } ;  
  
use lesson8\_mmingalov;  
  
DROP TABLE test\_wines\_quality;  
DROP TABLE test\_wines\_quality\_predicted;

CREATE TABLE test\_wines\_quality  
(Id int primary key,   
fixed\_acidity float,  
volatile\_acidity float,  
citric\_acid float,  
residual\_sugar float,  
chlorides float,  
free\_sulfur\_dioxide float,  
total\_sulfur\_dioxide float,  
density float,  
pH float,  
sulphates float,  
alcohol float  
);  
  
# большие буквы переводит в маленькие, тот же формат, что и фичи для записи  
CREATE TABLE test\_wines\_quality\_predicted  
(Id int primary key,   
fixed\_acidity float,  
volatile\_acidity float,  
citric\_acid float,  
residual\_sugar float,  
chlorides float,  
free\_sulfur\_dioxide float,  
total\_sulfur\_dioxide float,  
density float,  
pH float,  
sulphates float,  
alcohol float,  
quality float);

Видим, что таблицы создались:



### **STAGE 2 – MODEL TRAIN**

Код данного этапа выложен по ссылке:

<https://github.com/mmingalov/geekbrains-spark-streaming/blob/master/ml-wines-train.py>

Попробуем построить по нему линейную регрессию, обучить на исходных данных и сохранить в файл

В нашей сессии mobaXterm запускаем команды:

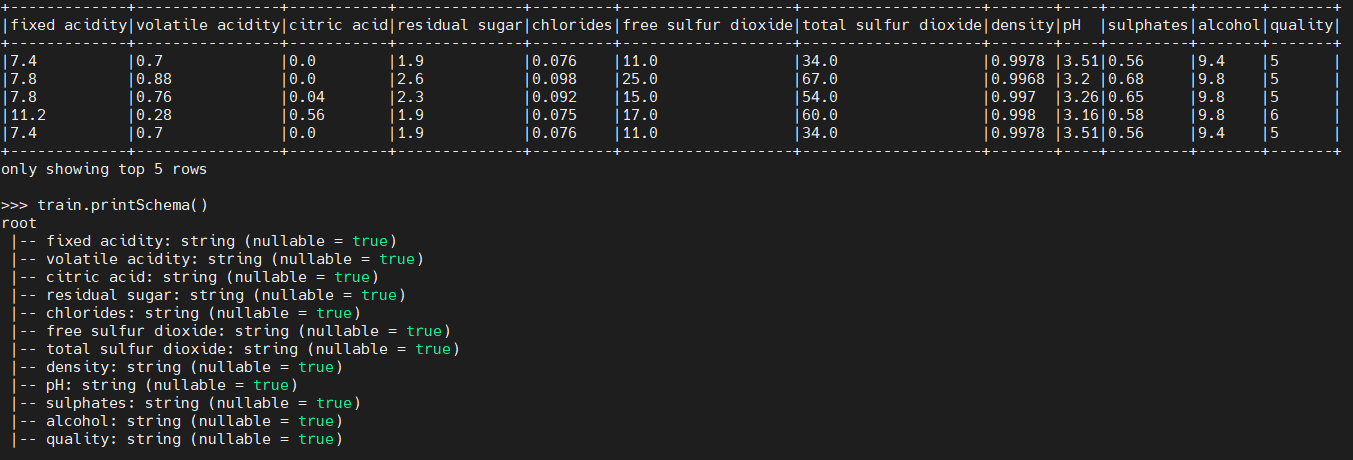


from pyspark.ml import Pipeline, PipelineModel  
from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql.types import StructType, StringType, IntegerType, TimestampType  
from pyspark.sql import functions as F  
from pyspark.ml.regression import LinearRegression  
from pyspark.ml.feature import OneHotEncoderEstimator, VectorAssembler, CountVectorizer, StringIndexer, IndexToString

train = spark.read\  
 .option("header", True)\  
 .csv("winequality-red.csv", sep=',')\  
 .cache()

train.show(5, False)  
train.printSchema()

Видим, что CSV читается как STRING



Прочитаем иначе с указанием типов данных столбцов. Для этого:

Способ 1

.withColumn("quality", F.expr("CAST(quality as FLOAT)"))\

Был бы возможен, но возник трабл со столбцами, у которых заголовок содержит пробел – не получилось использовать что-то типа:

.withColumn("fixed acidity", F.expr("CAST('fixed acidity' as FLOAT)"))\

В таких столбцах попадет NULL

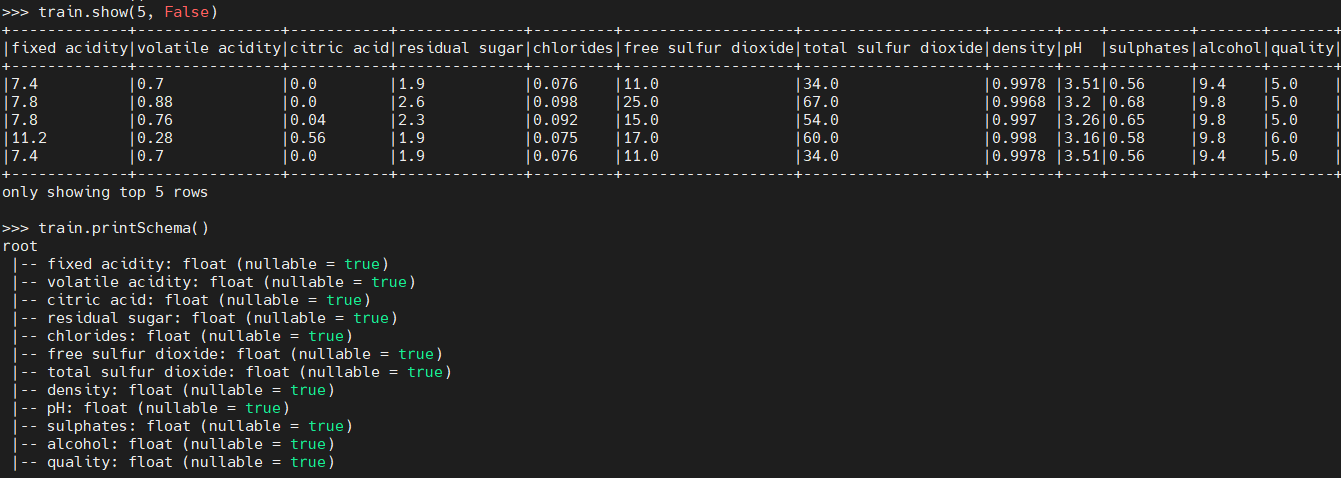
Поэтому идем другим путем:

Способ 2

from pyspark.sql.functions import \*  
from pyspark.sql.types import \*

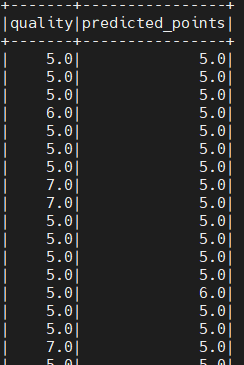
train = spark.read\  
 .option("header", True)\  
 .csv("winequality-red.csv", sep=',')\  
 .withColumn("fixed acidity", col("fixed acidity").cast('float'))\  
 .withColumn("volatile acidity", col("volatile acidity").cast('float'))\  
 .withColumn("citric acid", col("citric acid").cast('float'))\  
 .withColumn("residual sugar", col("residual sugar").cast('float'))\  
 .withColumn("chlorides", col("chlorides").cast('float'))\  
 .withColumn("free sulfur dioxide", col("free sulfur dioxide").cast('float'))\  
 .withColumn("total sulfur dioxide", col("total sulfur dioxide").cast('float'))\  
 .withColumn("density", col("density").cast('float'))\  
 .withColumn("pH", col("pH").cast('float'))\  
 .withColumn("sulphates", col("sulphates").cast('float'))\  
 .withColumn("alcohol", col("alcohol").cast('float'))\  
 .withColumn("quality", col("quality").cast('float'))\  
 .cache()  
  
train.show(5, False)  
train.printSchema()

ИТОГ – прочитали и изменили тип на нужный нам

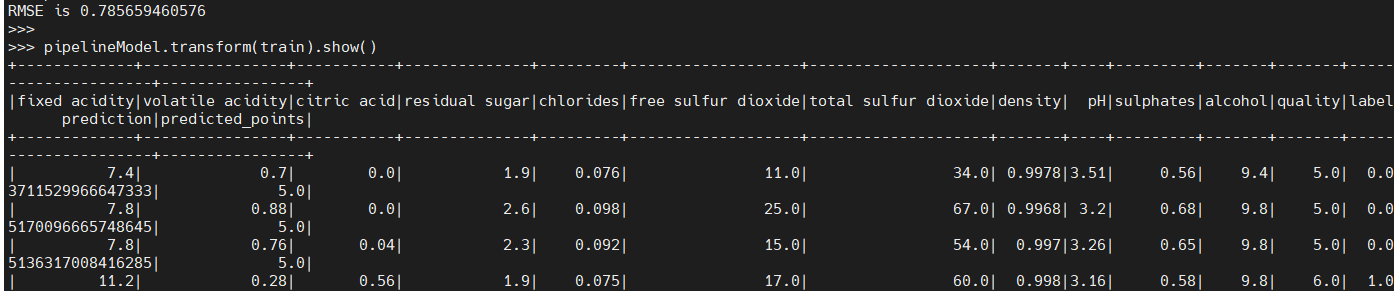


STAGES:

stages = []  
label\_stringIdx = StringIndexer(inputCol = 'quality', outputCol = 'label').setHandleInvalid("keep")  
stages += [label\_stringIdx]  
  
assemblerInputs = ['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide','total sulfur dioxide','density','pH','sulphates','alcohol']  
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features").setHandleInvalid("keep")  
stages += [assembler]  
  
lr = LinearRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)  
stages += [lr]  
  
label\_stringIdx\_fit = label\_stringIdx.fit(train)  
indexToStringEstimator = IndexToString().setInputCol("prediction").setOutputCol("predicted\_points").setLabels(label\_stringIdx\_fit.labels)  
  
stages +=[indexToStringEstimator]  
  
pipeline = Pipeline().setStages(stages)  
pipelineModel = pipeline.fit(train)  
  
#сохраняем модель на HDFS  
pipelineModel.write().overwrite().save("wines\_LR\_model8\_mmingalov")  
  
###для наглядности  
pipelineModel.transform(train).select("quality", "predicted\_points").show(100)



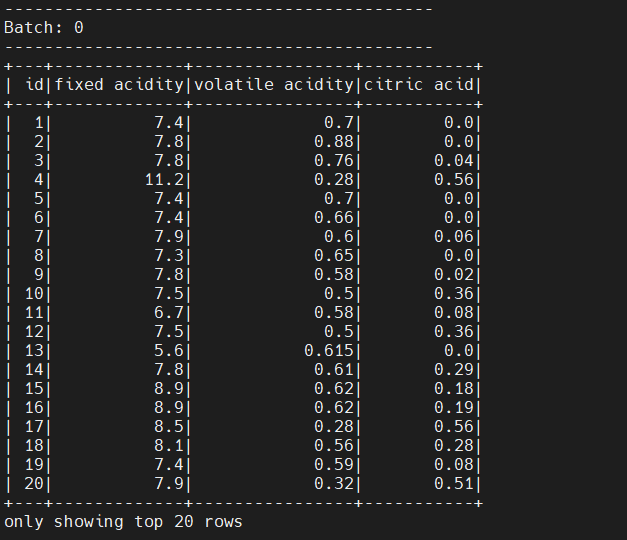
# rmse-метрика  
from pyspark.ml.evaluation import RegressionEvaluator  
regressionEvaluator = RegressionEvaluator(  
 predictionCol="predicted\_points",  
 labelCol="quality",  
 metricName="rmse")  
  
prediction = pipelineModel.transform(train).select(F.col("quality").cast("Float"),  
 F.col("predicted\_points").cast("Float"))  
rmse = regressionEvaluator.evaluate(prediction)  
print("RMSE is " + str(rmse))  
  
pipelineModel.transform(train).show()



### **STAGE 3 – TRANSFORM**

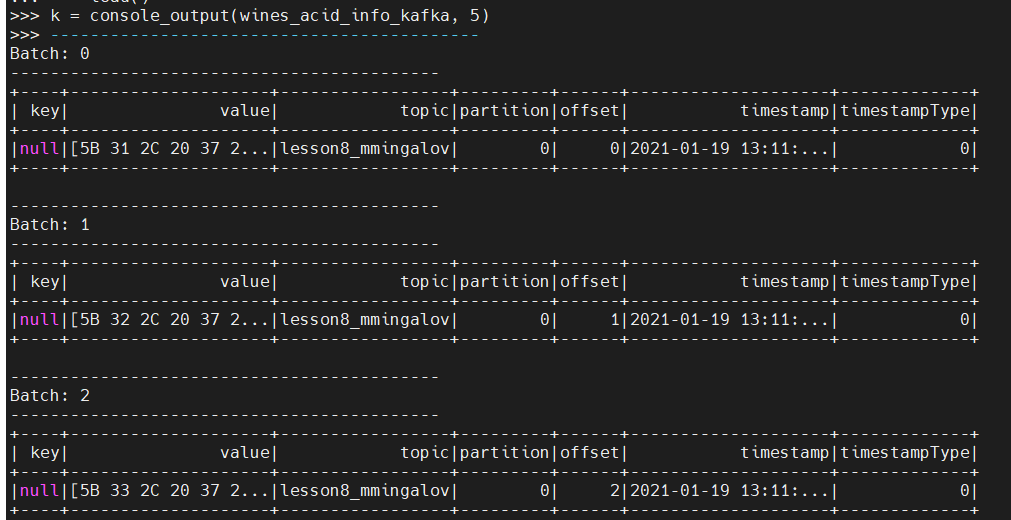
#export SPARK\_KAFKA\_VERSION=0.10  
#/spark2.4/bin/pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.5,com.datastax.spark:spark-cassandra-connector\_2.11:2.4.2 --driver-memory 512m --driver-cores 1 --master local[1]  
from pyspark.ml import Pipeline, PipelineModel  
from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql.types import StructType, StringType, IntegerType, FloatType, TimestampType  
from pyspark.sql import functions as F  
  
spark = SparkSession.builder.appName("mmingalov\_spark").getOrCreate() #не нужна для консоли, только для IDE  
#kafka\_brokers = "bigdataanalytics-worker-1.novalocal:6667"  
kafka\_brokers = "bigdataanalytics-worker-0.novalocal:6667"  
  
def console\_output(df, freq):  
 return df.writeStream \  
 .format("console") \  
 .trigger(processingTime='%s seconds' % freq ) \  
 .options(truncate=True) \  
 .start()  
  
#набор features 1 для Kafka  
schema = StructType() \  
 .add("id", IntegerType()) \  
 .add("fixed acidity", FloatType()) \  
 .add("volatile acidity", FloatType()) \  
 .add("citric acid", FloatType())  
 # .add("residual sugar", FloatType()) \  
 # .add("chlorides", FloatType()) \  
 # .add("free sulfur dioxide", FloatType()) \  
 # .add("total sulfur dioxide", FloatType()) \  
 # .add("density", FloatType()) \  
 # .add("pH", FloatType()) \  
 # .add("sulphates", FloatType()) \  
 # .add("alcohol", FloatType())  
  
  
# csv - чтение из файла, sample  
wines\_acid\_info = spark \  
 .readStream\  
 .format("csv")\  
 .schema(schema)\  
 .options(path="input\_csv\_for\_stream\_lesson8",  
 sep=";",  
 header=True,  
 maxFilesPerTrigger=1)\  
 .load()  
  
out = console\_output(wines\_acid\_info, 5)  
out.stop()

Результат в консоли:



# запись в Кафку - sink, заранее создаем топик командой:  
# /usr/hdp/3.1.4.0-315/kafka/bin/kafka-topics.sh --create --topic lesson8\_mmingalov --zookeeper bigdataanalytics-worker-0.novalocal:2181 --partitions 1 --replication-factor 2 --config retention.ms=-1  
# не забываем удалять чекпоинт!  
def kafka\_sink(df, freq):  
 return df.selectExpr("CAST(null AS STRING) as key", "CAST(struct(\*) AS STRING) as value") \  
 .writeStream \  
 .format("kafka") \  
 .trigger(processingTime='%s seconds' % freq ) \  
 .option("topic", "lesson8\_mmingalov") \  
 .option("kafka.bootstrap.servers", kafka\_brokers) \  
 .option("checkpointLocation", "checkquality/kafka\_checkpoint") \  
 .start()  
  
stream = kafka\_sink(wines\_acid\_info, 5)  
  
#этот процесс надо оставить, чтобы ловить обновления  
#stream.stop()  
  
#читаем кафку по одной записи, но можем и по 1000 за раз  
wines\_acid\_info\_kafka = spark.readStream. \  
 format("kafka"). \  
 option("kafka.bootstrap.servers", kafka\_brokers). \  
 option("subscribe", "lesson8\_mmingalov"). \  
 option("startingOffsets", "earliest"). \  
 option("maxOffsetsPerTrigger", "1"). \  
 load()  
  
k = console\_output(wines\_acid\_info\_kafka, 5)

В консоли появляется следующее:

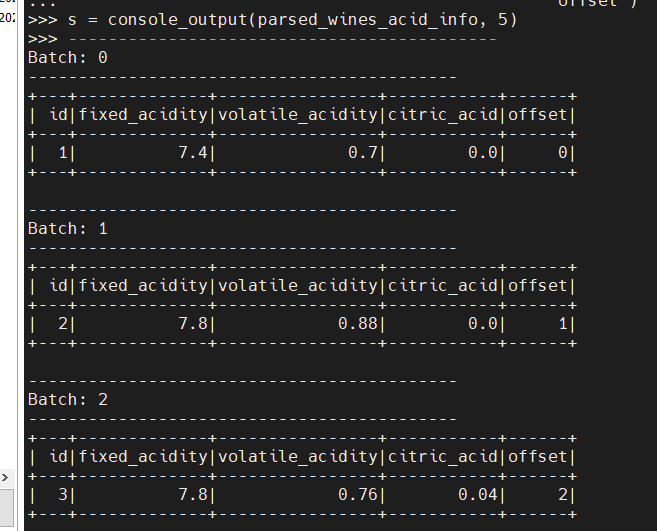


k.stop()

Запускаем команды:

# VALUE строкой. Принимает только cast("String"), другие форматы не дает задать  
value\_wines\_acid\_info = wines\_acid\_info\_kafka.select(F.regexp\_replace(F.col("value").cast("String"),  
 "([\[\]])", "").alias("value"),  
 "offset")

# сразу переводим в числовой формат  
parsed\_wines\_acid\_info = value\_wines\_acid\_info.selectExpr(  
 "CAST(split(value, ',')[0] as INTEGER) as id",  
 "CAST(split(value, ',')[1] as FLOAT) as fixed\_acidity",  
 "CAST(split(value, ',')[2] as FLOAT) as volatile\_acidity",  
 "CAST(split(value, ',')[3] as FLOAT) as citric\_acid",  
 "offset")  
  
s = console\_output(parsed\_wines\_acid\_info, 5)

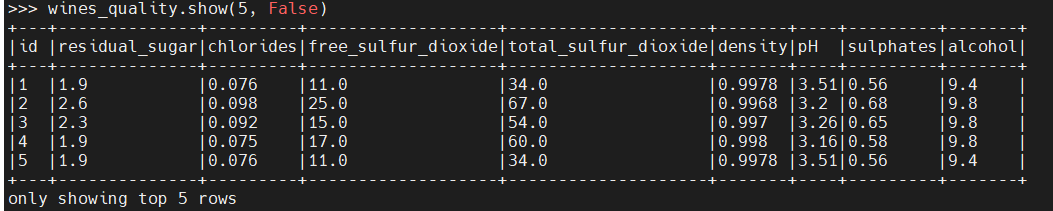


Остановим

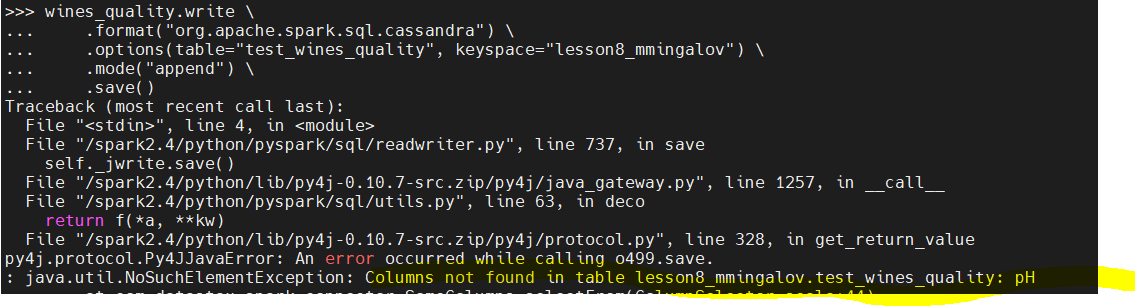
s.stop()

#прочитаем всю схему как она есть в CSV и возьмем потом только столбцы для поднабора 2  
schema\_ = StructType() \  
 .add("id", IntegerType()) \  
 .add("fixed acidity", FloatType()) \  
 .add("volatile acidity", FloatType()) \  
 .add("citric acid", FloatType()) \  
 .add("residual sugar", FloatType()) \  
 .add("chlorides", FloatType()) \  
 .add("free sulfur dioxide", FloatType()) \  
 .add("total sulfur dioxide", FloatType()) \  
 .add("density", FloatType()) \  
 .add("pH", FloatType()) \  
 .add("sulphates", FloatType()) \  
 .add("alcohol", FloatType())  
  
#path="for\_cassandra"  
wines\_ = spark.read\  
 .format("csv")\  
 .schema(schema\_)\  
 .options(path="input\_csv\_for\_stream\_lesson8",  
 sep=";",  
 header=True,  
 maxFilesPerTrigger=1)\  
 .load()  
wines\_.show(5, False)

#собственно поднабор 2  
wines\_quality = wines\_.select(  
 "id",  
 F.col("residual sugar").alias("residual\_sugar"),  
 "chlorides",  
 F.col("free sulfur dioxide").alias("free\_sulfur\_dioxide"),  
 F.col("total sulfur dioxide").alias("total\_sulfur\_dioxide"),  
 "density",  
 "pH",  
 "sulphates",  
 "alcohol"  
)  
  
wines\_quality.show(5, False)

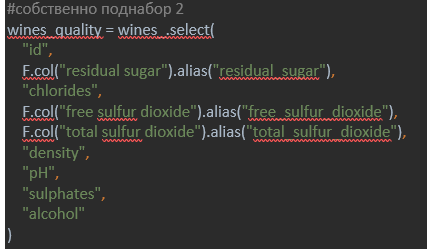
  
  
# положить "исторические" данные в Кассандру  
wines\_quality.write \  
 .format("org.apache.spark.sql.cassandra") \  
 .options(table="test\_wines\_quality", keyspace="lesson8\_mmingalov") \  
 .mode("append") \  
 .save()

НЕ СРАБОТАЕТ

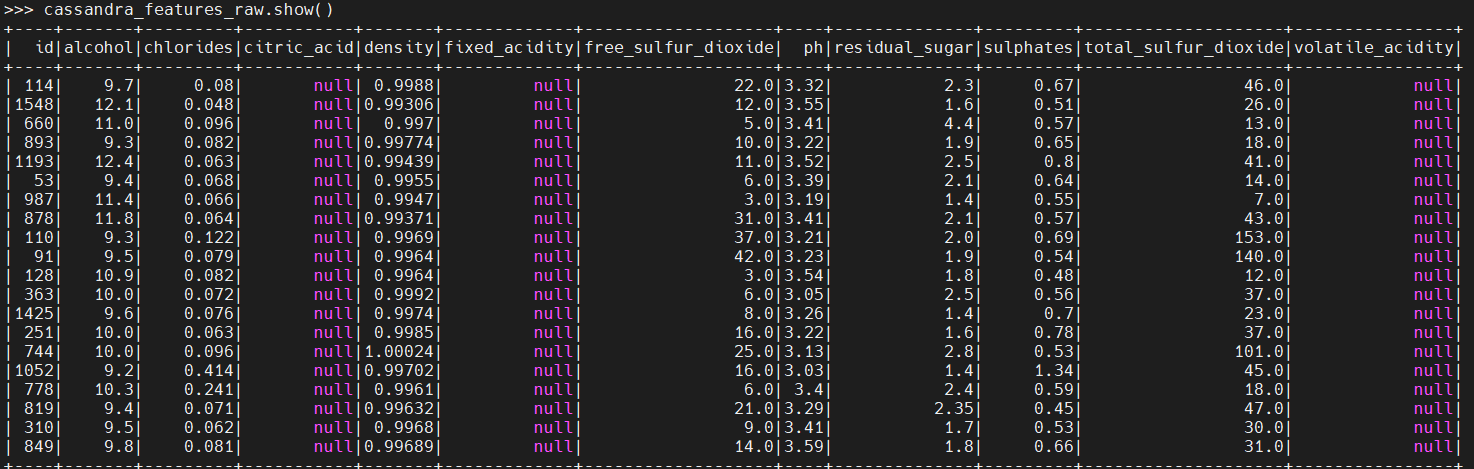


НЕ НРАВИТСЯ что столбец задали как "pH",

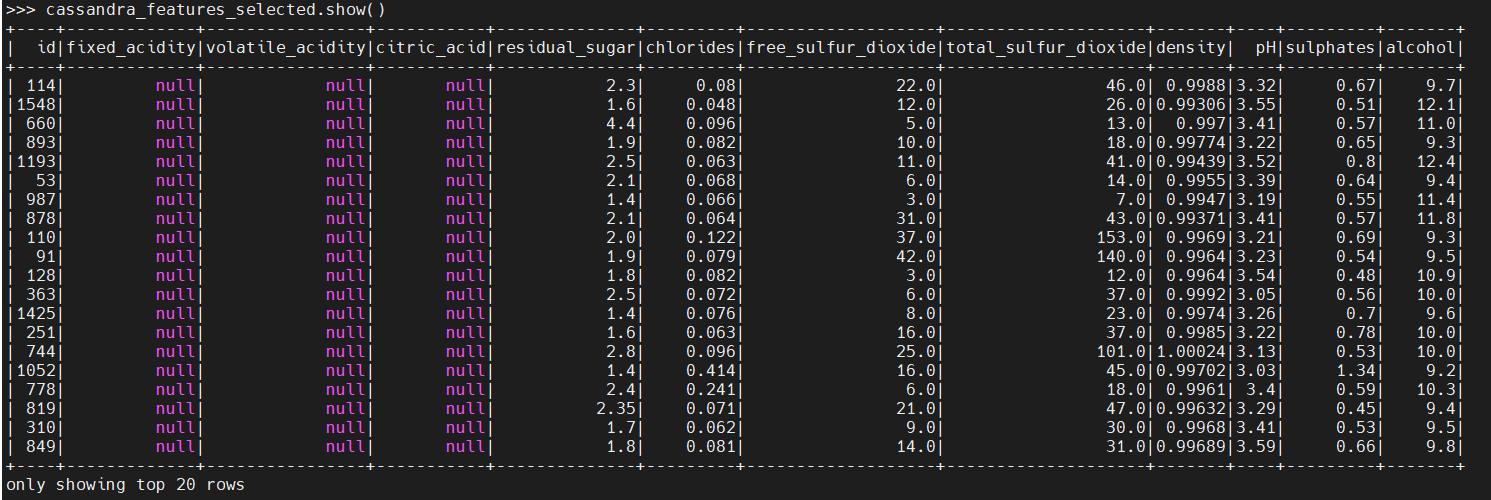
Ему надо с маленькой буквы. Поэтому:



# читаем из Кассандры  
cassandra\_features\_raw = spark.read \  
 .format("org.apache.spark.sql.cassandra") \  
 .options(table="test\_wines\_quality", keyspace="lesson8\_mmingalov" ) \  
 .load()  
  
cassandra\_features\_raw.show()



cassandra\_features\_selected = cassandra\_features\_raw.select ('id','fixed\_acidity',  
 'volatile\_acidity','citric\_acid','residual\_sugar',  
 'chlorides','free\_sulfur\_dioxide','total\_sulfur\_dioxide',  
 'density','pH','sulphates','alcohol')  
cassandra\_features\_selected.show()



#подгружаем ML из HDFS  
pipeline\_model = PipelineModel.load("wines\_LR\_model8\_mmingalov")

#вся логика в этом foreachBatch  
def writer\_logic(df, epoch\_id):  
 df.persist()  
 print("---------I've got new batch--------")  
 print("This is what I've got from Kafka source:")  
 df.show()  
 features\_from\_kafka = df.groupBy("id") \  
 .agg(F.lit(0.0).alias("fixed\_acidity"),  
 F.lit(0.0).alias("volatile\_acidity"), \  
 F.lit(0.0).alias("citric\_acid"), \  
 F.lit(0.0).alias("residual\_sugar"), \  
 F.lit(0.0).alias("chlorides"),  
 F.lit(0.0).alias("free\_sulfur\_dioxide"), \  
 F.lit(0.0).alias("total\_sulfur\_dioxide"), \  
 F.lit(0.0).alias("density"), \  
 F.lit(0.0).alias("ph"), \  
 F.lit(0.0).alias("sulphates"), \  
 F.lit(0.0).alias("alcohol"))  
 print("Here is the sums from Kafka source:")  
 features\_from\_kafka.show()  
 teachsers\_list\_df = features\_from\_kafka.select("id").distinct()  
 #превращаем DataFrame(Row) в Array(Row)  
 wines\_list\_rows = teachsers\_list\_df.collect()  
 #превращаем Array(Row) в Array(String)  
 wines\_list = map( lambda x: str(x.\_\_getattr\_\_("id")) , wines\_list\_rows)  
 where\_string = " id = " + " or id = ".join(wines\_list)  
 print("I'm gonna select this from Cassandra:")  
 print(where\_string)  
 print("Here is what I've got from Cassandra:")  
 cassandra\_features\_selected.where(where\_string).show()  
 features\_from\_cassandra = cassandra\_features\_selected.where(where\_string).na.fill(0)  
 features\_from\_cassandra.persist()  
 print("I've replaced nulls with 0 from Cassandra:")  
 features\_from\_cassandra.show()  
 #объединяем микробатч из кафки и микробатч касандры  
 cassandra\_file\_union = features\_from\_kafka.union(features\_from\_cassandra)  
 cassandra\_file\_aggregation = cassandra\_file\_union.groupBy("id") \  
 .agg(F.lit("fixed\_acidity").alias("fixed\_acidity"),  
 F.lit("volatile\_acidity").alias("volatile\_acidity"), \  
 F.lit("citric\_acid").alias("citric\_acid"), \  
 F.lit("residual\_sugar").alias("residual\_sugar"), \  
 F.lit("chlorides").alias("chlorides"),  
 F.lit("free\_sulfur\_dioxide").alias("free\_sulfur\_dioxide"), \  
 F.lit("total\_sulfur\_dioxide").alias("total\_sulfur\_dioxide"), \  
 F.lit("density").alias("density"), \  
 F.lit("ph").alias("ph"), \  
 F.lit("sulphates").alias("sulphates"), \  
 F.lit("alcohol").alias("alcohol"))  
 print("Here is how I aggregated Cassandra and file:")  
 cassandra\_file\_aggregation.show()  
 predict = pipeline\_model.transform(cassandra\_file\_aggregation)  
 print("I've got the prediction:")  
 predict.show()  
 predict\_short = predict.select('id', 'fixed acidity','volatile acidity','citric acid',  
 'residual sugar','chlorides','free sulfur dioxide',  
 'total sulfur dioxide','density','ph','sulphates','alcohol',  
 F.col('predicted\_quality').cast(FloatType()).alias('quality'))  
 print("Here is what I've got after model transformation:")  
 predict\_short.show()  
 #обновляем исторический агрегат в касандре - записываем в другую таблицу  
 predict\_short.write \  
 .format("org.apache.spark.sql.cassandra") \  
 .options(table="test\_wines\_quality\_predicted", keyspace="lesson8\_mmingalov") \  
 .mode("append") \  
 .save()  
 features\_from\_cassandra.unpersist()  
 print("I saved the prediction and aggregation in Cassandra. Continue...")  
 df.unpersist()  
  
#связываем источник и foreachBatch функцию, не забываем удалять чекпоинт  
stream\_foreachBatch = parsed\_wines\_acid\_info \  
 .writeStream \  
 .trigger(processingTime='30 seconds') \  
 .foreachBatch(writer\_logic) \  
 .option("checkpointLocation", "checkquality/test\_wines\_checkpoint")  
  
#поехали  
s = stream\_foreachBatch.start()  
  
s.stop()  
  
def killAll():  
 for active\_stream in spark.streams.active:  
 print("Stopping %s by killAll" % active\_stream)  
 active\_stream.stop()  
  
killAll()