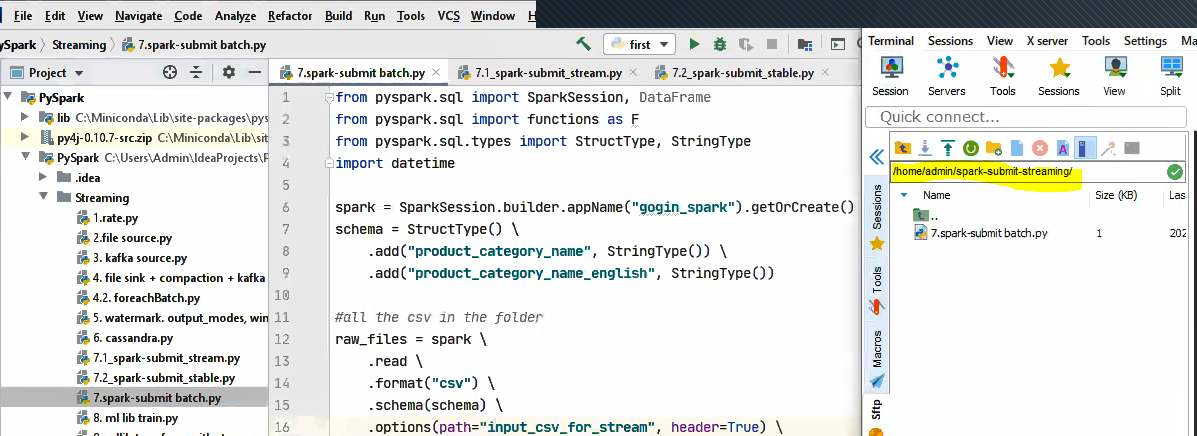
# **ЗАДАНИЕ 07 (Аналитика признаков в пакетном режиме):**

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**

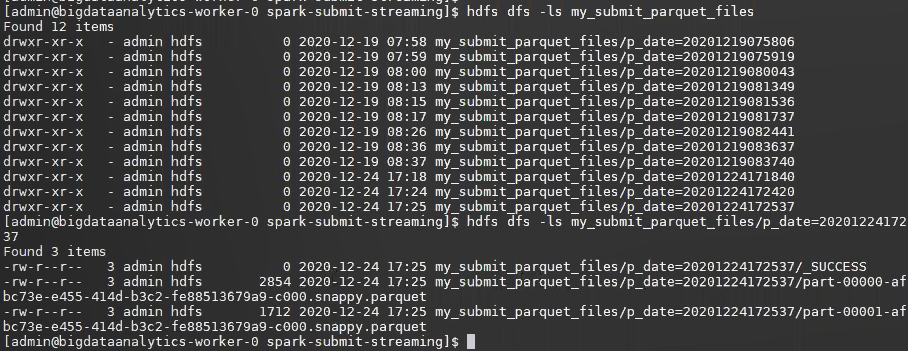
from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql import functions as F  
from pyspark.sql.types import StructType, StringType  
import datetime  
  
spark = SparkSession.builder.appName(**"mmingalov\_spark"**).getOrCreate()  
schema = StructType() \  
 .add(**"product\_category\_name"**, 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:

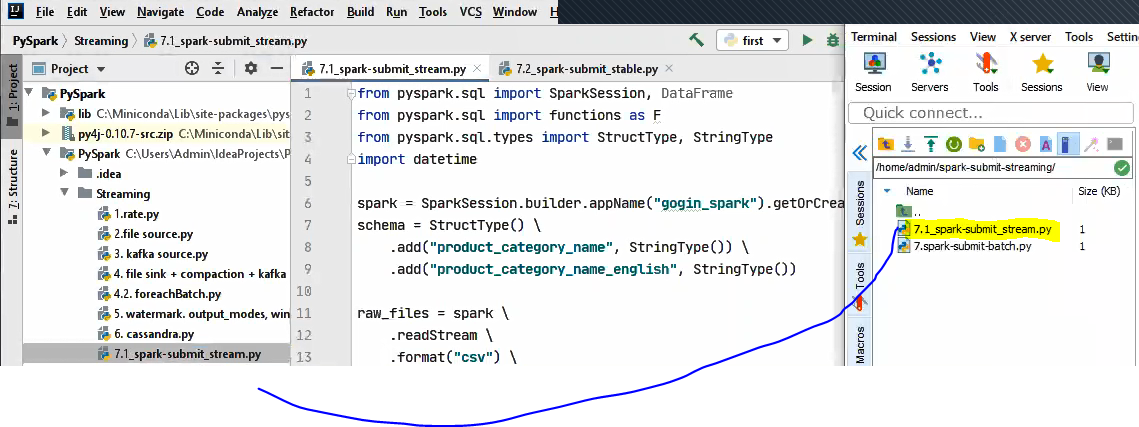




## **SPARK SUBMIT STREAM**

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**

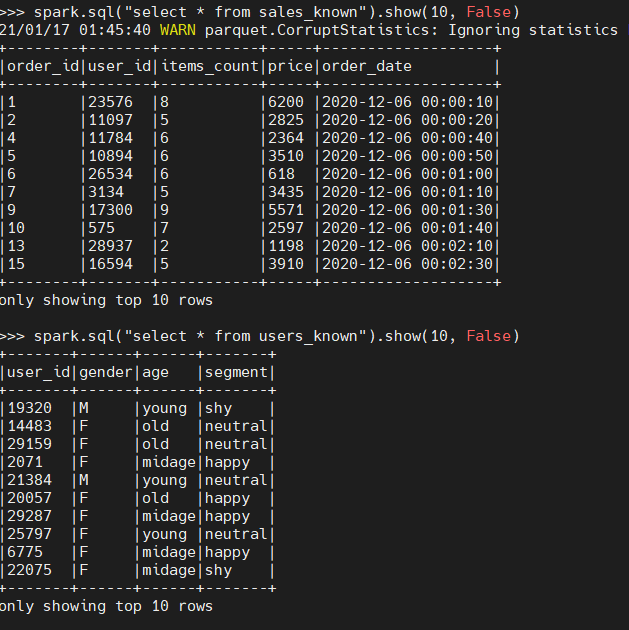
**Открываем новую сессию в mobaXterm, запускаем команды:**

export SPARK\_KAFKA\_VERSION=0.10

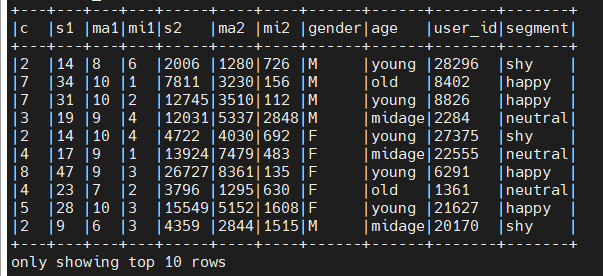
/spark2.4/bin/pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.5,com.datastax.spark:spark-cassandra-connector\_2.11:2.4.2 --driver-memory 512m --driver-cores 1 --master local[1]

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:**

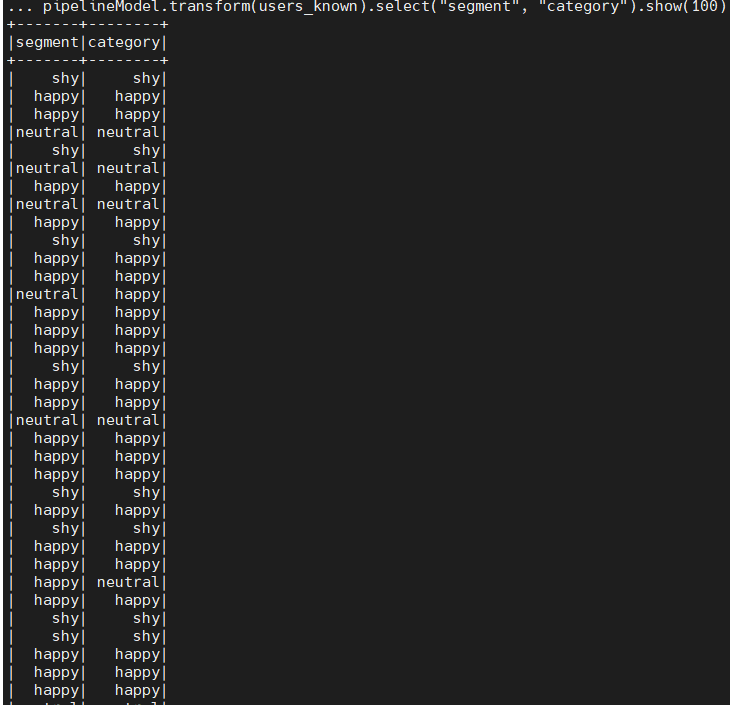
#подробное описание модели https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa  
#и https://spark.apache.org/docs/latest/ml-features.html  
#в общем - все анализируемые колонки заносим в колонку-вектор features  
categoricalColumns = ['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)

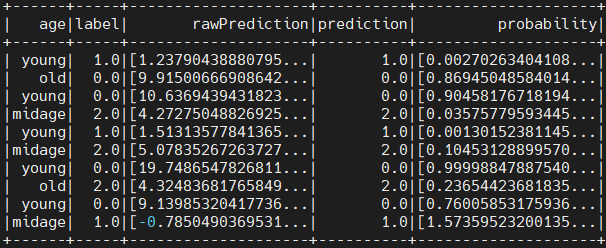
#сохраняем модель на HDFS  
pipelineModel.write().overwrite().save("mmingalov\_LR\_model8")

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

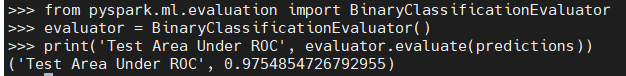


predictions = pipelineModel.transform(users\_known)

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.**

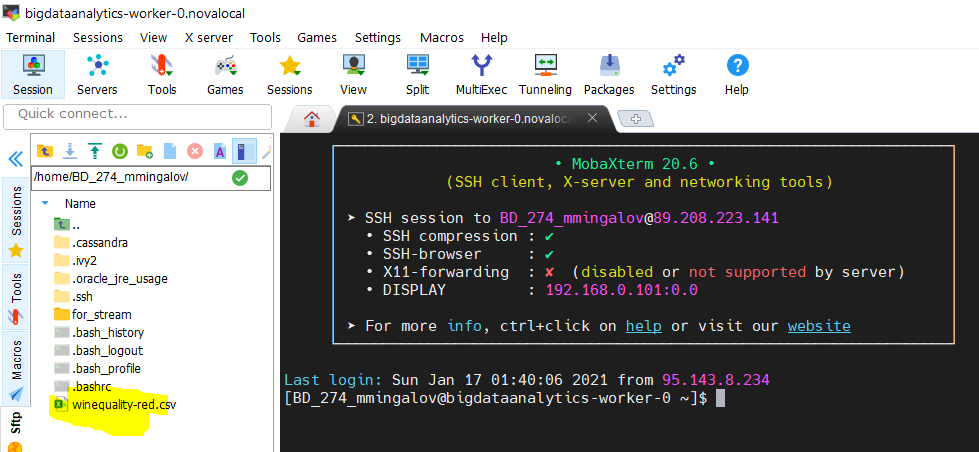
Для этого взят небольшой датасет по винам:

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

### **STAGE 1 – DATA PREPARING**

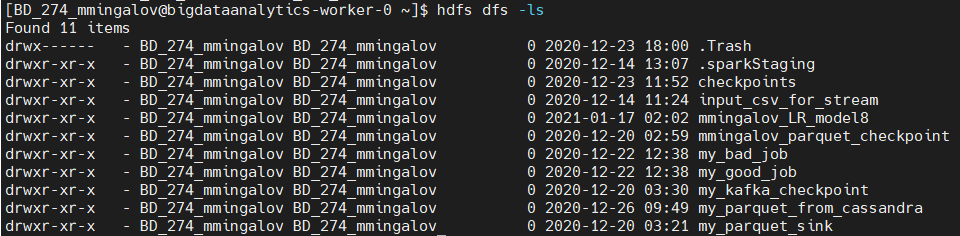
**Открываем новую сессию в mobaXterm подключением к worker-0:**

Методом drag&drop копируем в директорию терминала наш датасет:

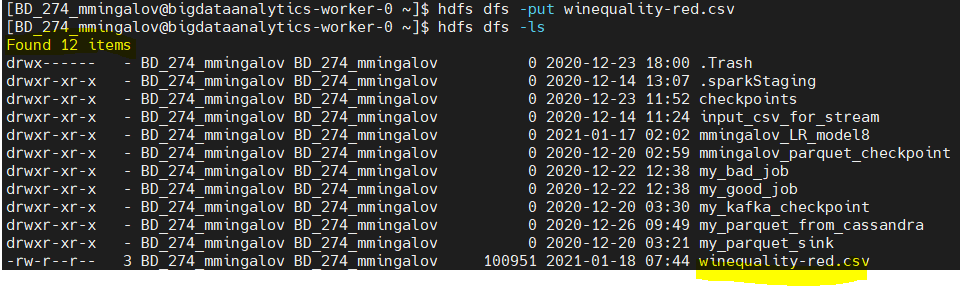


Теперь поместим его в HDFS.

Посмотрим, что у нас имеется в HDFS



Выполним копирование командой:



Теперь наш файл на HDFS

### **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:

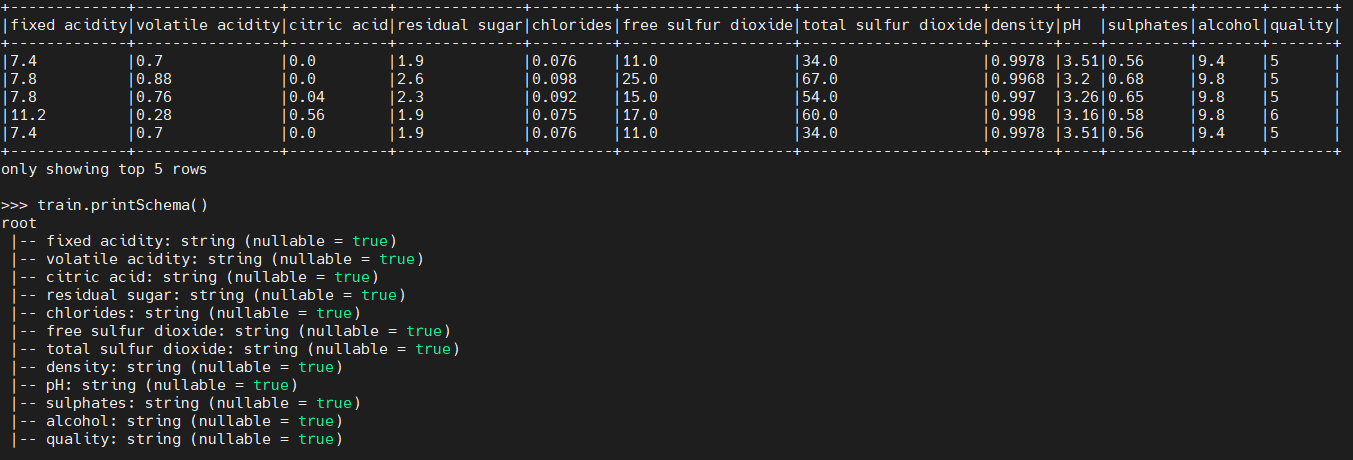


from pyspark.ml import Pipeline, PipelineModel  
from pyspark.sql import SparkSession, DataFrame  
from pyspark.sql.types import StructType, StringType, IntegerType, TimestampType  
from pyspark.sql import functions as F  
from pyspark.ml.regression import LinearRegression  
from pyspark.ml.feature import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

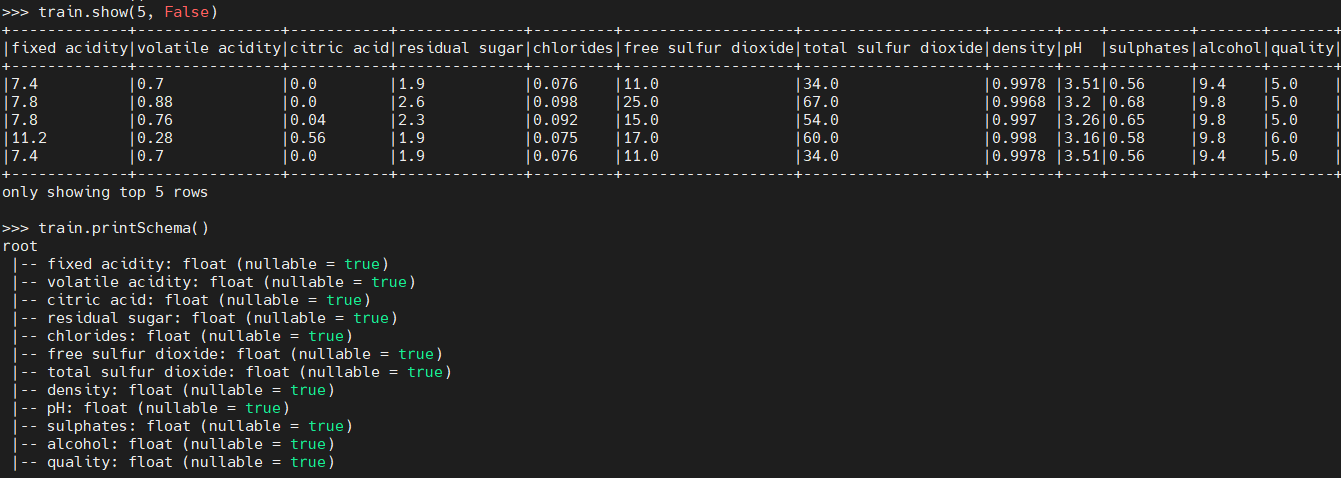
Поэтому идем другим путем:

Способ 2

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

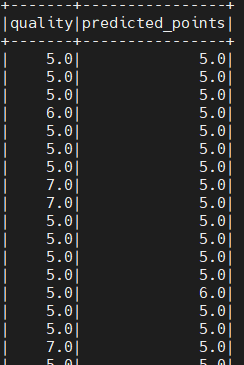
train = spark.read\  
 .option("header", True)\  
 .csv("winequality-red.csv", sep=',')\  
 .withColumn("fixed acidity", col("fixed acidity").cast('float'))\  
 .withColumn("volatile acidity", col("volatile acidity").cast('float'))\  
 .withColumn("citric acid", col("citric acid").cast('float'))\  
 .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

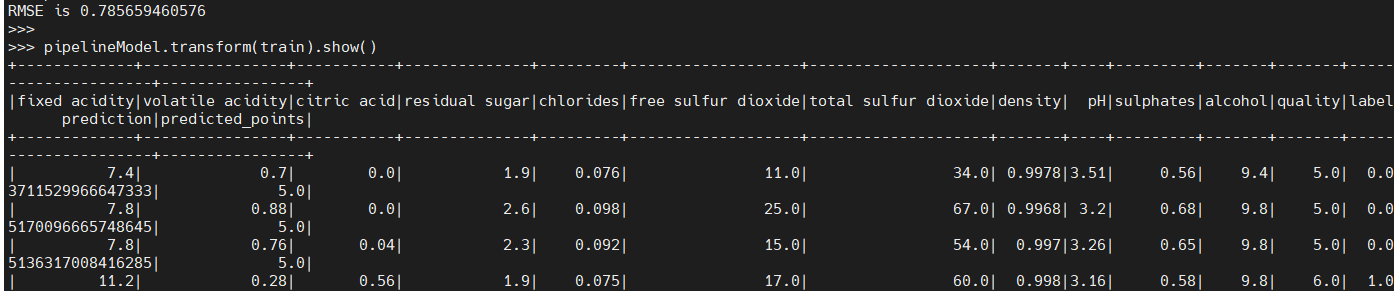


STAGES:

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**