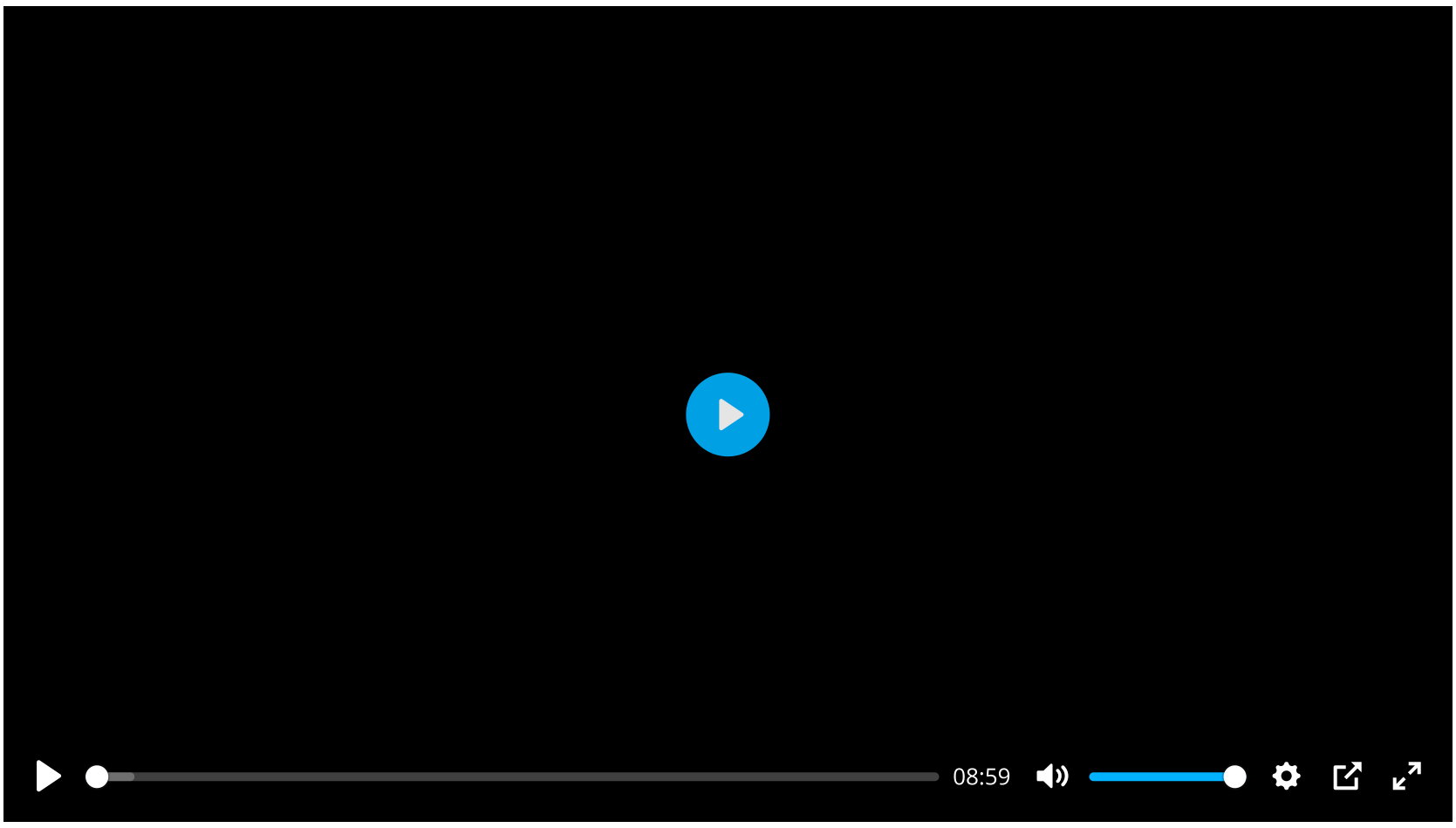
4.3 Dataframe'ы и преобразования



Текстовая расшифровка видео:

ЧТЕНИЕ И СХЕМЫ

План:

- Данные в кластере;
- Schema on read;
- Чтение из файла.

Данные в кластере

Рассмотрим, как можно брать данные и загружать их на кластер.

Представим, что у нас есть датасет, например, табель с оценками, где есть фамилии и оценки. Вызываем метод «CreateDataFrame», после чего указываем какие данные берем, а также указываем схему:

```
1. marks = [
2.  ("Ivanov",5),
3.  ("Petrov",4),
4.  ("Sidorov",3),
5.  ("Smirnov",4)
6. ]
7. marks_columns = ["last_name","mark"]
8. marksDF = spark.createDataFrame(data=marks, schema=marks_columns)
9. marksDF.printSchema()
10. marksDF.show(truncate=False)
```

В качестве схемы передаем список из имен колонок, более подробных метаданных не задаем.

Если выполнить данный код, можно увидеть следующий вывод:

```
root
 |-- last_name: string (nullable = true)
 |-- mark: long (nullable = true)

+---------+----+
|last_name|mark|
+---------+----+
|Ivanov   |5   |
|Petrov   |4   |
|Sidorov  |3   |
|Smirnov  |4   |
+---------+----+
```

PrintSchema выведет схему DataFrame и сам DataFrame.

В данной схеме есть загвоздка: Spark предполагает, что Mark имеет тип «Long» – большое число. Поскольку мы знаем собственные данные (оценки от 1 до 5), то можем задать тип колонки «Byte», так как в 1 байт все поместится.

Schema on read

Если мы будем действовать так, как описано выше, нам потребуется описать схему более подробно. Нам потребуется описать колонки и типы, а после передать их на старте в CreateDataFrame:

```
1. from pyspark.sql.types import (
2.   StructType,
3.   StructField,
4.   StringType,
5.   ByteType
6. )
7.
8. marks_schema = StructType([
9.   StructField('last_name', StringType(), True),
10.   StructField('mark', ByteType(), True)
11. ])
12.
13. marksDF = spark.createDataFrame(data=marks, schema=marks_schema)
14. marksDF.printSchema()
15. marksDF.show(truncate=False)
```

Импортируем из **pySpark.sql.Types**, создаем объект структуры и указываем, что Last name имеет тип «String», а Mark имеет тип «Byte». Все остальное запускаем точно так же.

Существует множество типов. Их вы можете изучить, перейдя по ссылке:
https://spark.apache.org/docs/latest/sql-ref-datatypes.html

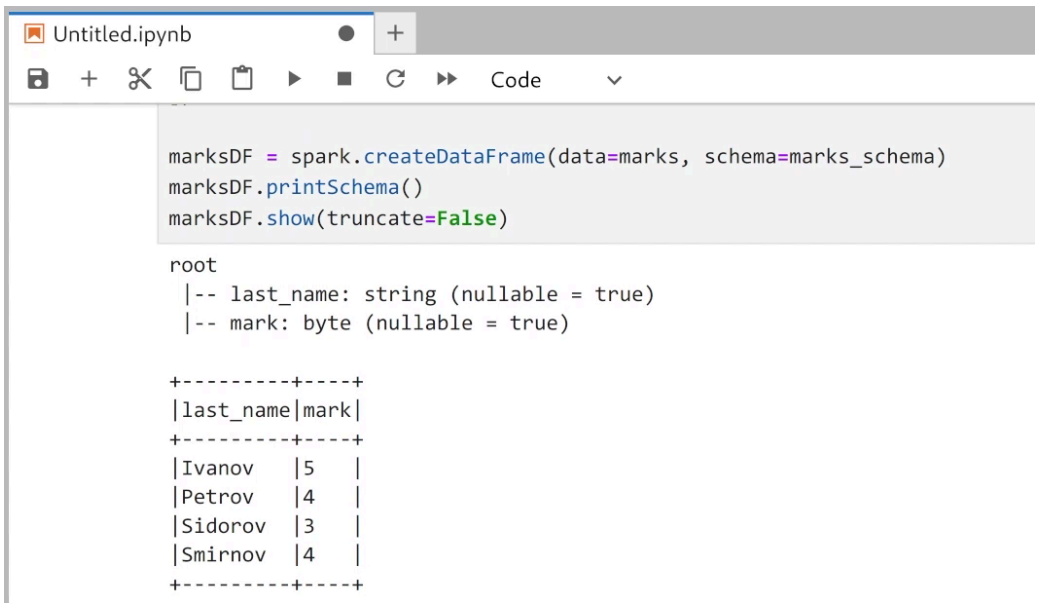
Если мы выполним этот код, то увидим практически идентичный вывод:

```
root
 |-- last_name: string (nullable = true)
 |-- mark: byte (nullable = true)

+---------+----+
|last_name|mark|
+---------+----+
|Ivanov   |5   |
|Petrov   |4   |
|Sidorov  |3   |
|Smirnov  |4   |
+---------+----+
```

Дедлайн 07 июля, 23:59 Мск

Проверим работает ли данный код. Мы видим, что код работает, и тип в схеме – «Byte»:



Чтение из файла

Рассмотрим пример:

```
df = spark.read.csv("Experiments/slurm/airport-codes.csv")
# абсолютно идентично, но с явным указанием входного формата
df = spark.read.format("csv") \
  .load("Experiments/slurm/airport-codes.csv")
df.printSchema()
"""
root
|-- _c0: string (nullable = true)
|-- _c1: string (nullable = true)
|-- _c2: string (nullable = true)
|-- _c3: string (nullable = true)
...
"""
# берем заголовок из файла
df2 = spark.read.option("header",True) \
  .csv("Experiments/slurm/airport-codes.csv")
```

Здесь мы видим одно из отличий от RDD: нам нужно самостоятельно прочитать и распарсить файл, превратив его в RDD.

В данном случае мы можем использовать конкретные читалки. Если данные храним в CSV (в исходном файле), то просто говорим «Spark, прочитай CSV». Помимо этого, мы можем вызвать метод CSV, а также задать через «.format» то, какой формат хотим читать.

Читаем данные из файла. Точно так же Spark пытается вывести схему, имеющуюся в файле. **Нюанс**: по умолчанию Spark не предполагает, что в файле есть заголовок с колонками. Мы можем либо передать их при чтении, либо указать «option (**"headler" True**)», тогда колонки в DataFrame будут называться не «C0» или «C1». Они будут прочитанными из самого заголовка.

Также **опциями (option)** можно указывать **разделитель (delimiter), автовывод схемы** (inferSchema может потребовать больше времени) и многое другое в зависимости от входного формата.

Если мы запустим данный код с указанным headler=true, то названия колонок возьмутся из headler, и мы увидим следующую схему:

Дедлайн 07 июля, 23:59 Мск

```
root
 |-- ident: string (nullable = true)
 |-- type: string (nullable = true)
 |-- name: string (nullable = true)
 |-- elevation_ft: string (nullable = true)
 |-- continent: string (nullable = true)
 |-- iso_country: string (nullable = true)
 |-- iso_region: string (nullable = true)
 |-- municipality: string (nullable = true)
 |-- gps_code: string (nullable = true)
 |-- iata_code: string (nullable = true)
 |-- local_code: string (nullable = true)
 |-- coordinates: string (nullable = true)
```

Прочитать данные в DataFrame несложно. Читать можно из Python-объектов, из DataFrame Pandas, CSV, Parquet и т.д. Также можно указать не файл, а каталог, где лежат пачки CSV-файлов и т.д., прочитав одной командой все файлы.

Как вам урок? 😠 😐 🔥

Изучил, далее ›

Дедлайн 07 июля, 23:59 Мск