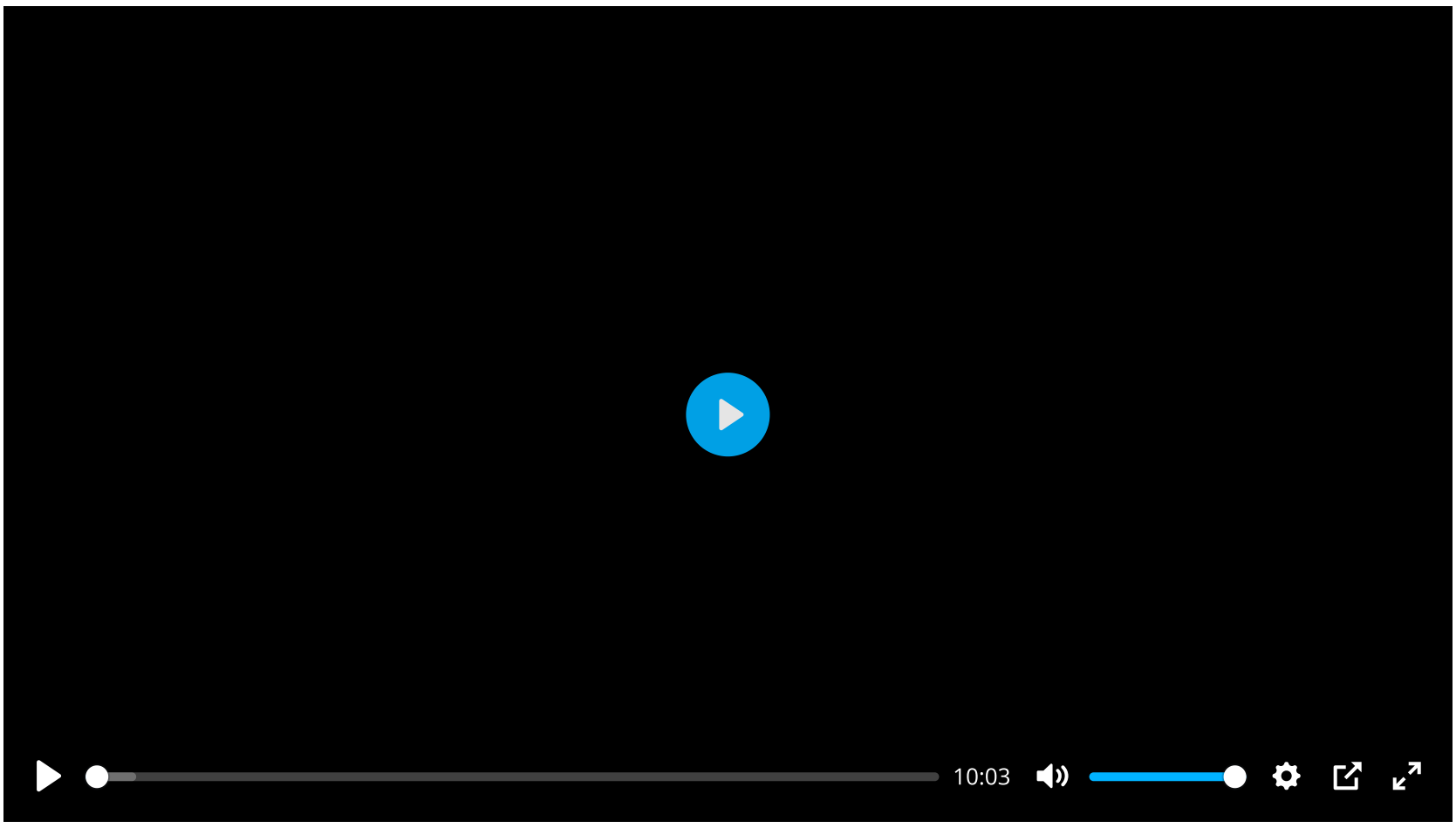
4.3 Dataframe'ы и преобразования



Текстовая расшифровка видео:

КЕЙС ОЧИСТКИ ДАННЫХ

План:

- Специально битые данные;
- Семейства функций в pySpark.

Специально битые данные

Рассмотрим в качестве примера битый набор Json-документов:

```
testData = """{ "name":"Moscow", "country":"Rossiya", "continent": "Europe", "population": 12
    { "name":"Madrid", "country":"Spain" }
    { "name":"Paris", "country":"France", "continent": "Europe", "population" : 2196936}
    { "name":"Berlin", "country":"Germany", "continent": "Europe", "population": 3490105}
    { "name":"Barselona", "country":"Spain", "continent": "Europe" }
    { "name":"Cairo", "country":"Egypt", "continent": "Africa", "population": 11922948 }
    { "name":"Cairo", "country":"Egypt", "continent": "Africa", "population": 11922948 }
    { "name":"New York, "country":"USA","""
```

Для очистки датасета:

- Удалим строку с невалидным Json;
- Удалим дубликаты;
- Заполним null'ы в колонках;
- Исправим «Rossiya» на «Russia».

Итак, у нас есть SparkContext и тестовые данные в виде Json. Обернем данным запросом все в

na.ipynb

Code

```
spark = SparkSession(sc)

[2]: from pyspark.sql.functions import col, lower, lit, length, split, explode, from_json

[3]: testData = """{ "name":"Moscow", "country":"Rossiya", "continent": "Europe", "population": 12380664}
        { "name":"Madrid", "country":"Spain" }
        { "name":"Paris", "country":"France", "continent": "Europe", "population" : 2196936}
        { "name":"Berlin", "country":"Germany", "continent": "Europe", "population": 3490105}
        { "name":"Barselona", "country":"Spain", "continent": "Europe" }
        { "name":"Cairo", "country":"Egypt", "continent": "Africa", "population": 11922948 }
        { "name":"Cairo", "country":"Egypt", "continent": "Africa", "population": 11922948 }
        { "name":"New York, "country":"USA","""

[4]: raw = spark.range(0,1).select(lit(testData).alias("value"))

[5]: jsonStrings = split(col("value"), "\n").alias("value")

[6]: df_split = raw.select(explode(jsonStrings))

[7]: df = spark.read.json(df_split.rdd.map(lambda r: r.col))
```

Зададим правило: разделить данные по символам переноса строки.

По этому правилу применим функцию «explode». Применяя эту функцию, мы разбиваем данные в DataFrame на «кусочки», вследствие чего появляется DataFrame с разбиением.

Мы видим, что в каждой строчке находится Json-документ (пока в не очень очищенном виде):

na.ipynb

Code

```
[4]: raw = spark.range(0,1).select(lit(testData).alias("value"))

[5]: jsonStrings = split(col("value"), "\n").alias("value")

[6]: df_split = raw.select(explode(jsonStrings))

[7]: df_split.show()

+--------------------+
|                 col|
+--------------------+
|{ "name":"Moscow"...|
|    { "name":"Mad...|
|    { "name":"Par...|
|    { "name":"Ber...|
|    { "name":"Bar...|
|    { "name":"Cai...|
|    { "name":"Cai...|
|    { "name":"New...|
+--------------------+
```

Проходимся по этим данным и десериализуем их в Json. Конвертируем DataFrame в RDD, проходимся по нему и читаем Json из получившегося RDD стандартным ридером Spark – **Spark Read Json**. В результате такого действия получится DataFrame, в котором Spark разложит данные, прочитанные нами:

na.ipynb

Code

```
[6]: df_split = raw.select(explode(jsonStrings))

[8]: df = spark.read.json(df_split.rdd.map(lambda r: r.col))

[9]: df.show(truncate=False)

+---------------------------------------+---------+-------+---------+----------+
|_corrupt_record                        |continent|country|name     |population|
+---------------------------------------+---------+-------+---------+----------+
|null                                   |Europe   |Rossiya|Moscow   |12380664  |
|null                                   |null     |Spain  |Madrid   |null      |
|null                                   |Europe   |France |Paris    |2196936   |
|null                                   |Europe   |Germany|Berlin   |3490105   |
|null                                   |Europe   |Spain  |Barselona|null      |
|null                                   |Africa   |Egypt  |Cairo    |11922948  |
|null                                   |Africa   |Egypt  |Cairo    |11922948  |
|    { "name":"New York, "country":"USA",|null     |null   |null     |null      |
+---------------------------------------+---------+-------+---------+----------+

[8]: df.select(col("_corrupt_record")).na.drop("all").collect()
```

По значениям он автоматически создаст все колонки, а где значений нет, укажет «null», а для записи,

Дедлайн 07 июля, 23:59 Мск

данных куда-либо для дальнейшего разбора. В данном примере эти данные можно просто снести.

Семейства функций в PySpark

Поговорим о семействах функций в PySpark, которые находятся в модуле «NA».

NA (Not Available) – это значение, которое отсутствует.

В NA три разные функции:

- **Drop** – функция, которая удаляет записи null'ами по определенным правилам;
- **Fill** – функция, которая заполняет данные на основе пропущенных значений;
- **Replace** – функция, которая принимает словарь, заменяя одно значение на другое:

```
[10]: df.select(col("_corrupt_record")).na.drop("all").collect()

[10]: [Row(_corrupt_record='    { "name":"New York, "country":"USA",')]

[9]: fill_data = {"continent": "Undefined", "population": 0}
     replace_data = {"Rossiya": "Russia"}

     clean_data = df.drop(col("_corrupt_record")) \
       .na.drop("all") \
       .na.fill(fill_data) \
       .na.replace(replace_data, subset="country") \
       .dropDuplicates()

[ ]:
```

Повторим во избежание путаницы:

Fill – функция, при которой мы заменяем null'ы в колонках дефолтным значением.

Replace – функция, при которой мы ищем значение, указанное в ключе, и заменяем его на то, которое указано в значении.

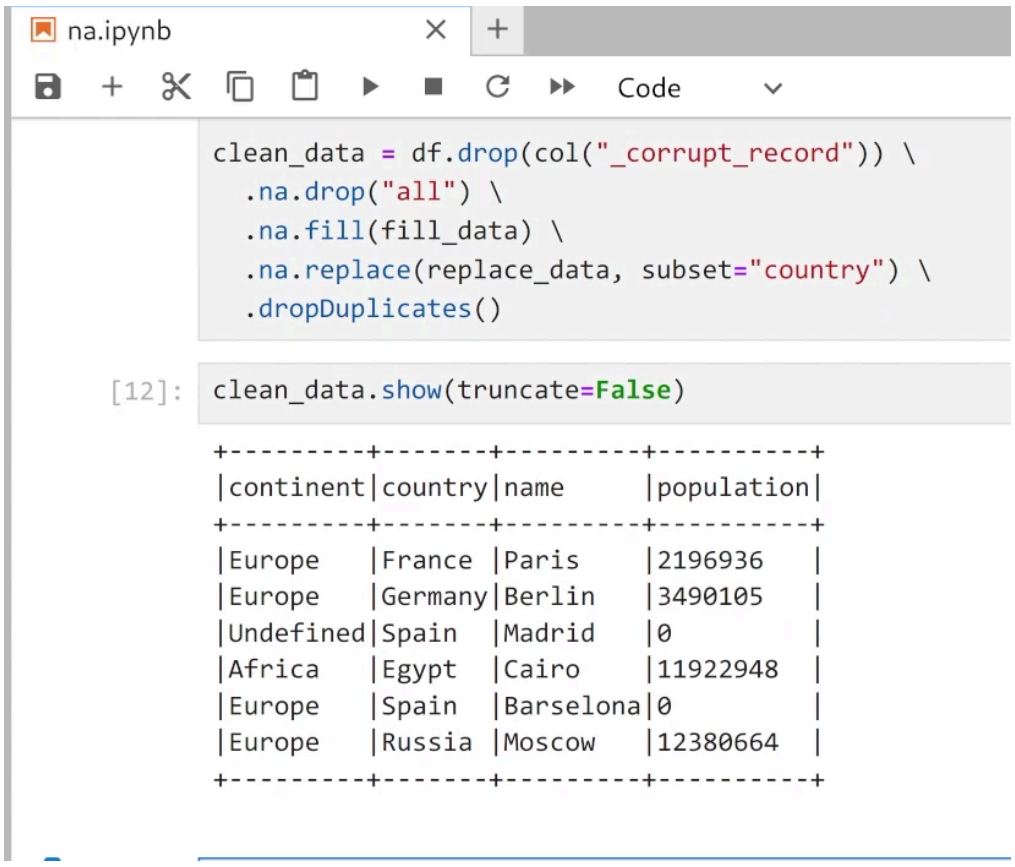
Возвращаясь к примеру, отметим, что среди записей есть дубликат (Каир встречается дважды):

```
na.ipynb
Code
spark = SparkSession(sc)

[2]: from pyspark.sql.functions import col, lower, lit, length, split, explode, from_json

[3]: testData = """{ "name":"Moscow", "country":"Rossiya", "continent": "Europe", "population": 12380664}
         { "name":"Madrid", "country":"Spain" }
         { "name":"Paris", "country":"France", "continent": "Europe", "population" : 2196936}
         { "name":"Berlin", "country":"Germany", "continent": "Europe", "population": 3490105}
         { "name":"Barselona", "country":"Spain", "continent": "Europe" }
         { "name":"Cairo", "country":"Egypt", "continent": "Africa", "population": 11922948 }
         { "name":"Cairo", "country":"Egypt", "continent": "Africa", "population": 11922948 }
         { "name":"New York, "country":"USA","""

[4]: raw = spark.range(0,1).select(lit(testData).alias("value"))

[5]: jsonStrings = split(col("value"), "\n").alias("value")

[6]: df_split = raw.select(explode(jsonStrings))

[7]: df = spark.read.json(df_split.rdd.map(lambda r: r.col))
```

В DataFrame есть параметр «**DropDuplicates**», который позволяет сносить совпадающие ряды, оставляя одну копию.

Выполним данный код:

Дедлайн 07 июля, 23:59 Мск



Появились очищенные данные. В столбце, где не указан континент, пишется «Undefinde», где не указан population, указан «0» и т.д.

Как вам урок?

Изучил, далее

Дедлайн 07 июля, 23:59 Мск