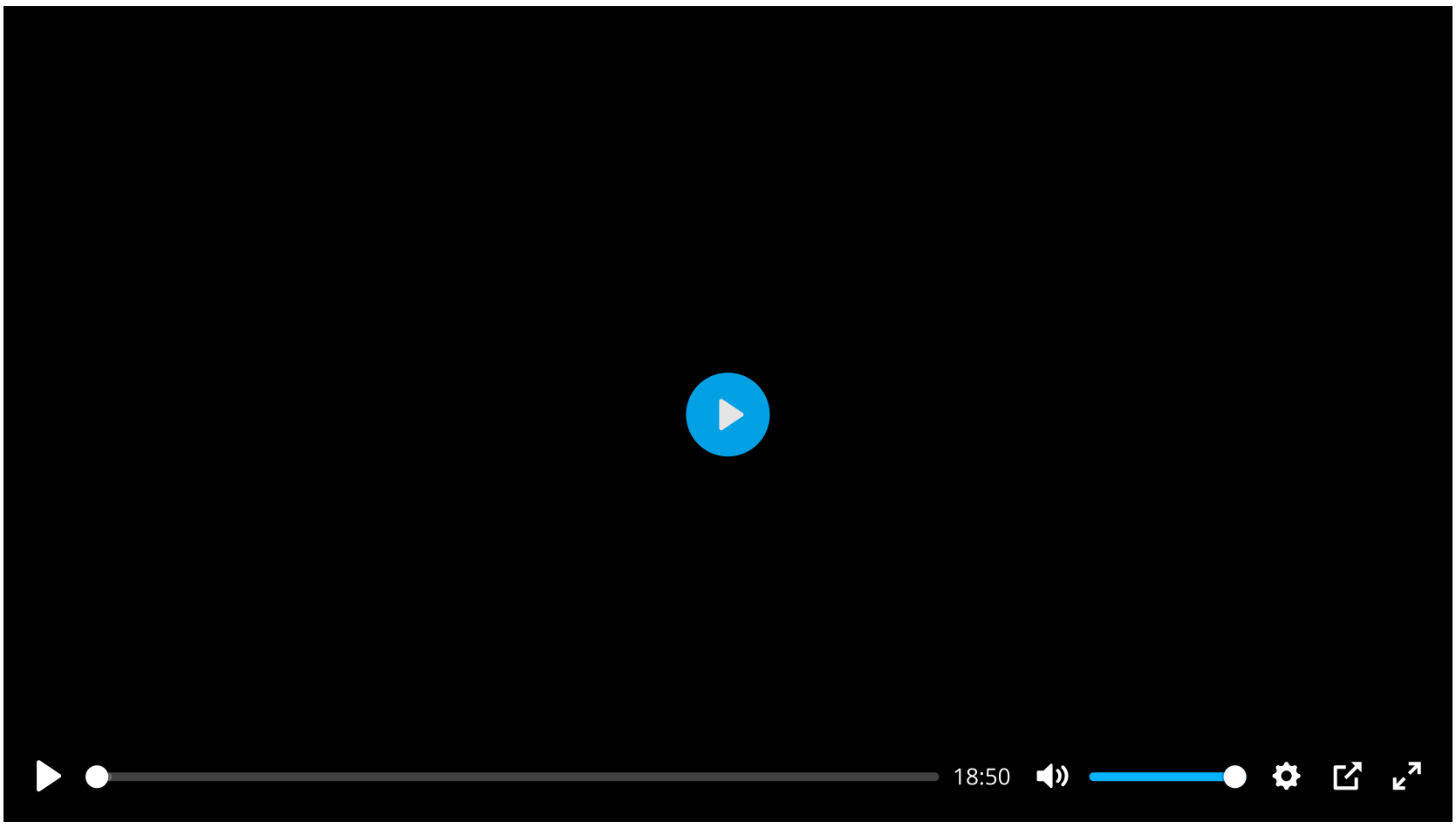
4.3 Dataframe'ы и преобразования



Текстовая расшифровка видео:

БАЗОВЫЕ ПРИМЕРЫ РАБОТЫ

План:

- Фильтрация;
- То ли Python, то ли SQL;
- Работа с колонками;
- GroupBy/Join.

Фильтрация

Вспомним из уроков о RDD про трансформации и action'ы. В данном случае подход частично схож:

```
# один фильтр
df2.filter(df2.continent == 'AN')
# несколько фильтров
df2.filter((df2.continent == 'AN') & (df2.type == 'medium_airport'))
# колоночные выражения
from pyspark.sql.functions import col
df2[col("type") == "medium_airport"]
# буквально SQL
df2.createOrReplaceTempView("airport_codes")
spark.sql("""
 SELECT name FROM airport_codes
  WHERE continent = 'AN'
  AND type = 'medium_airport'
""")
```

У нас есть Dataframe со строчками и колонками. Например, мы хотим здесь что-либо отфильтровать. Проще всего сделать фильтрацию по значению отдельной колонки, сказав «верни новый Dataframe, где

Мы можем задать несколько фильтров, сказав «выведи все значения, которые находятся на континенте Антарктида с указанным типом “Medium airport”».

Рассмотрим на примере:

```
[5]: df2 = spark.read.option("header",True) \
       .csv("airport-codes.csv")
```

```
[8]: df2.filter((df2.continent == 'AN') & (df2.type == 'medium_airport'))

[8]: DataFrame[ident: string, type: string, name: string, elevation_ft: string, continent: string, iso_country: string, iso_regio
     n: string, municipality: string, gps_code: string, iata_code: string, local_code: string, coordinates: string]

[ ]:
```

Фильтр выполняется моментально, потому что он так же «лениво» создается и ничего не делает.

Выведем то, что нам нужно посчитать:

```
|ident|type          |name                          |elevation_ft|continent|iso_country|iso_region|municipality   |gps_code|ia
ta_code|local_code|coordinates                          |
+-----+--------------+------------------------------+------------+---------+-----------+----------+---------------+--------+--
-------+----------+-------------------------------------+
|NZIR |medium_airport|McMurdo Station Ice Runway    |1           |AN       |AQ         |AQ-U-A    |McMurdo Station|NZIR    |nu
ll     |null      |166.468994140625, -77.85399627685547 |
|NZPG |medium_airport|McMurdo Station Pegasus Field|18          |AN       |AQ         |AQ-U-A    |McMurdo Station|NZPG    |nu
ll     |null      |166.52499389648438, -77.9634017944336|
|NZSP |medium_airport|South Pole Station Airport    |9300        |AN       |AQ         |AQ-U-A    |null           |NZSP    |nu
ll     |null      |0, -90                               |
|NZWD |medium_airport|Williams Field                |68          |AN       |AQ         |AQ-U-A    |McMurdo Station|NZWD    |nu
ll     |null      |167.0570068359375, -77.86740112304688|
+-----+--------------+------------------------------+------------+---------+-----------+----------+---------------+--------+--
-------+----------+-------------------------------------+
```

Видим, что в Антарктиде есть четыре аэропорта.

Работа с Dataframe посредством написания кода на Python или на Scala и работа с ними посредством написания SQL-запросов во многом схожа.

Возвращаясь к нашему примеру, мы можем использовать трюк и зарегистрировать временное представление «CreateOrReplaceTempView» на основе Dataframe. Это создаст именованную запись метаданных, которую мы можем использовать дальше в SQL-запросах. Например, мы можем сделать несколько таблиц из нескольких Dataframʼов, а потом запустить SQL, который сделает joinʼы, преобразования и т.д. между этими таблицами. Все это будет схоже, если бы мы написали это кодом.

В случае, если мы пишем кодом, автоматические проверки типов и т.д. позволяют немного сократить количество вариантов «выстрела в ногу». Мы можем проверить правильность своих действий.

Мы можем создать временный View (временное представление) и выполнить SQL-запрос напрямую к ней, чтобы сделать тот же самый фильтр, который мы сделали кодом.

То ли Python, то ли SQL

Обычно нет «предпочтительного» варианта написания кода. Проще посмотреть, как он выглядит у коллег для удобства всех сторон.

В целом, вы можете писать на гибриде Pythonʼa и SQL, как в данном примере:

```
df2.select(
    col("*"),
    lower(col("name")).name("lower_name"),
    (length(col("name")) + 1).alias("length"),
    lit("foo").alias("bar")
)
```

Мы можем сделать фильтр и указать нужные колонки и т.д. В примере делаем select, сразу указывая пачку колонок. Мы забираем все колонки из начального Dataframe, создаем еще одну колонку «Lower name», являющуюся результатом применения колонки «Name» к приведению к нижнему регистру. Создаем еще одну колонку «Length», представляющую из себя числовую колонку (длина имени + 1), а также колонку «Bar», заполняющуюся литеральным значением foo.

Обратите внимание, что функции col, lower, length, lit — это функции, которые находятся в

Дедлайн 07 июля, 23:59 Мск

Рассмотрим на примере.

Вызвав данный код, видим следующее:

```
[11]: df2.select(
          col("*"),
          lower(col("name")).name("lower_name"),
          (length(col("name")) + 1).alias("length"),
          lit("foo").alias("bar")
      ).show()
```

```
+-----+-------------+--------------------+------------+---------+-----------+----------+------------+--------+---------+-----
-----+--------------------+--------------------+------+---+
|ident|         type|                name|elevation_ft|continent|iso_country|iso_region|municipality|gps_code|iata_code|local
_code|         coordinates|          lower_name|length|bar|
+-----+-------------+--------------------+------------+---------+-----------+----------+------------+--------+---------+-----
-----+--------------------+--------------------+------+---+
|  00A|     heliport|   Total Rf Heliport|          11|       NA|         US|     US-PA|    Bensalem|     00A|     null|
00A|-74.9336013793945...|   total rf heliport|    18|foo|
| 00AA|small_airport|Aero B Ranch Airport|        3435|       NA|         US|     US-KS|       Leoti|    00AA|     null|
00AA|-101.473911, 38.7...|aero b ranch airport|    21|foo|
| 00AK|small_airport|        Lowell Field|         450|       NA|         US|     US-AK|Anchor Point|    00AK|     null|
00AK|-151.695999146, 5...|        lowell field|    13|foo|
| 00AL|small_airport|        Epps Airpark|         820|       NA|         US|     US-AL|     Harvest|    00AL|     null|
00AL|-86.7703018188476...|        epps airpark|    13|foo|
```

Вводятся все колонки, также дополнительно появляется колонка «Lower name», «Length», «Bar».

Работа с колонками

Можно создавать и вычислять колонки через select, также можно использовать дополнительный метод «withColumn».

Поскольку датафреймы неизменяемые, мы можем с помощью withColumn создать новый датафрейм на основе предыдущего, который будет содержать нужную колонку.

Мы можем делать разные преобразования, например, удалить колонку, вызвав «Drop». Также на удаление можно задать несколько колонок, указав нужные через запятую. Нюанс: при вызове «Drop», Spark не будет ругаться, если мы попытаемся удалить колонку с несуществующим именем. Вызывайте «Drop» для тех колонок, которые существуют.

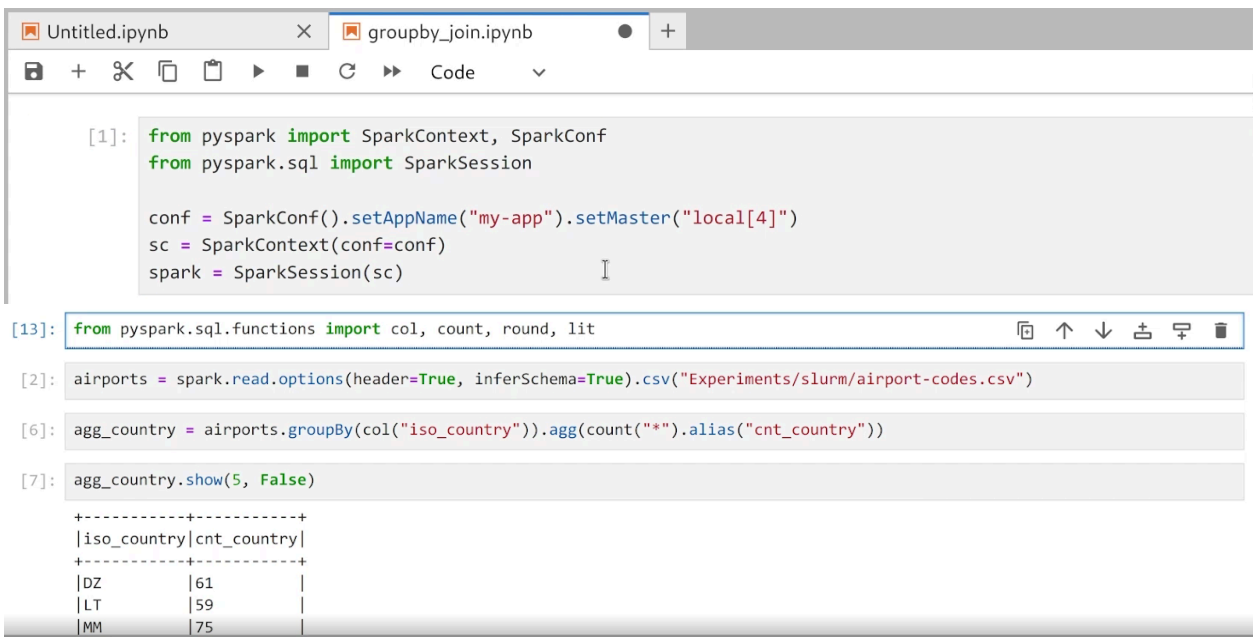
GroupBy/Join

Добавим новую колонку к датасету Airports, в которой будет процент заданного типа аэропорта ко всем типам аэропорта по каждой стране.

Для этого сделаем объединение:

- Посчитаем количество аэропортов с группировкой по типу и стране;
- Посчитаем количество аэропортов в целом;
- Объединим получившиеся датасеты;
- Подставим результат объединения в изначальный датасет.

Рассмотрим на примере:



Поскольку мы запускаем впервые, то создаем сессию и контекст и импортируем все, что нужно, после

Дедлайн 07 июля, 23:59 Мск

GroupBy будет во многом похож на SQL и на groupBy при работе в Pandas. Выглядеть это будет следующим образом:

```
[6]: agg_country = airports.groupBy(col("iso_country")).agg(count("*").alias("cnt_country"))

[7]: agg_country.show(5, False)

+-----------+-----------+
|iso_country|cnt_country|
+-----------+-----------+
|DZ         |61         |
|LT         |59         |
|MM         |75         |
|CI         |26         |
|TC         |8          |
+-----------+-----------+
only showing top 5 rows
```

Принцип работы groupBy:

Когда мы группируем значения по какому-то полю, у нас получается колонка с этим полем. Появляются группы элементов, которые нужно разагрегировать, то есть применить агрегирующую функцию. Можно посчитать сумму, количество, среднее значение и т.д. Есть множество агрегатных функций. В нашем случае мы делаем groupBy по стране и считаем количество аэропортов. В итоге мы получим следующее:

Untitled.ipynb | groupby_join.ipynb | Code

```
[2]: from pyspark.sql.functions import col, count, round, lit

[3]: airports = spark.read.options(header=True, inferSchema=True).csv("airport-codes.csv")

[4]: agg_country = airports.groupBy(col("iso_country")).agg(count("*").alias("cnt_country"))

[5]: agg_country.show(5, False)

+-----------+-----------+
|iso_country|cnt_country|
+-----------+-----------+
|DZ         |61         |
|LT         |59         |
|MM         |75         |
|CI         |26         |
|TC         |8          |
+-----------+-----------+
only showing top 5 rows
```

Видим аббревиатуру страны и количество аэропортов.

Далее, мы создаем еще один датасет, где делаем группировку сразу по двум полям: по типу аэропорта, по стране, после чего также считаем количество. Мы хотим отношения одного к другому. Получится следующий датафрейм:

```
[6]: agg_type_country = airports.groupBy(col("type"), col("iso_country")).agg(count("*").alias("cnt_country_type"))

[7]: agg_type_country.show(5, False)

+--------------+-----------+----------------+
|type          |iso_country|cnt_country_type|
+--------------+-----------+----------------+
|large_airport |GB         |27              |
|heliport      |CH         |19              |
|closed        |LT         |4               |
|medium_airport|SS         |3               |
|medium_airport|BE         |8               |
+--------------+-----------+----------------+
only showing top 5 rows
```

Теперь необходимо посчитанный датафрейм на первом этапе сджойнить с датафреймом, посчитанным на втором этапе. В данном случае вызывается метод «Join»:

```
[14]: percent = agg_type_country.join(agg_country, ["iso_country"], "inner") \
          .select(col("iso_country"), col("type"), (round(lit(100) * col("cnt_country_type") / col("cnt_country"), 2).alias("percent
```

В данном случае есть разные варианты того, по каким полям мы будем делать Join (объединение). В этой ситуации у нас есть общее поле, которое мы будем джойнить, это поле – iso_country. Оно под одинаковым названием присутствует в обоих датафреймах, поэтому мы можем указать в виде списка iso_country (оно автоматически подцепится Spark'ом). Если же поля были бы указаны по-разному, то пришлось бы указывать их названия.

Возвращаемся к примеру и делаем Join. Его тип – «Inner». Затем делаем select. Выбираем колонки, подсчитываем и выводим в виде процентов. Датасет, который должен получится:

Дедлайн 07 июля, 23:59 Мск

```
[9]: percent.show()
```

```
+-----------+-------------+-------+
|iso_country|         type|percent|
+-----------+-------------+-------+
|         DZ| small_airport|  36.07|
|         DZ|medium_airport|  59.02|
|         DZ| large_airport|   1.64|
|         DZ|       closed|   3.28|
|         LT|medium_airport|  10.17|
|         LT| large_airport|   1.69|
|         LT| small_airport|  77.97|
|         LT|     heliport|   3.39|
|         LT|       closed|   6.78|
|         MM|       closed|   1.33|
|         MM| large_airport|   2.67|
|         MM|medium_airport|  26.67|
|         MM| small_airport|  69.33|
|         CI|medium_airport|  26.92|
|         CI| small_airport|  73.08|
|         TC| small_airport|   50.0|
|         TC|medium_airport|   50.0|
```

Теперь приджойним полученную таблицу к изначальной таблице с аэропортами, чтобы информация в процентах тоже была включена в новый датафрейм на основе исходного.

Мы делаем Join на основе двух полей, которые есть: iso_country, type. Делаем левый джойн и выбираем, что нам необходимо (также делаем «sample», беря не все записи, а лишь кусочек). Мы объединили все в исходном датафрейме:

```
[10]: result = airports.join(percent, ["iso_country", "type"], "left")
      result.select(col("ident"), col("iso_country"), col("type"), col("percent")).sample(0.2).show(20, False)
```

```
+-----+-----------+-------------+-------+
|ident|iso_country|type         |percent|
+-----+-----------+-------------+-------+
|00AL |US         |small_airport|58.3   |
|00AZ |US         |small_airport|58.3   |
|00CA |US         |small_airport|58.3   |
|00FD |US         |heliport     |27.15  |
|00GA |US         |small_airport|58.3   |
|00IS |US         |small_airport|58.3   |
|00NK |US         |seaplane_base|2.47   |
|00OI |US         |heliport     |27.15  |
|00PA |US         |heliport     |27.15  |
|00S  |US         |small_airport|58.3   |
|00TE |US         |heliport     |27.15  |
|01CA |US         |heliport     |27.15  |
|01CO |US         |heliport     |27.15  |
|01IA |US         |small_airport|58.3   |
|01IN |US         |heliport     |27.15  |
|01K  |US         |small_airport|58.3   |
|01MA |US         |heliport     |27.15  |
|01NV |US         |small_airport|58.3   |
```

Join’ы работают по схожему принципу как в RDD, так и базах данных: мы указываем поля, по которым джойним, а на выходе получаем датафрейм, из которого делаем выборку.

Нужные дополнительные колонки, поля и т.д. высчитываем «на лету», используя функции, которые импортируем из р**ySpark.SQL.Functions**.

Как вам урок?

Изучил, далее

Дедлайн 07 июля, 23:59 Мск